KARABÜK ÜNİVERSİTESİ
TEKNOLOJİ FAKÜLTESİ MEKATRONİK MÜHENDİSLİĞİ BÖLÜMÜ

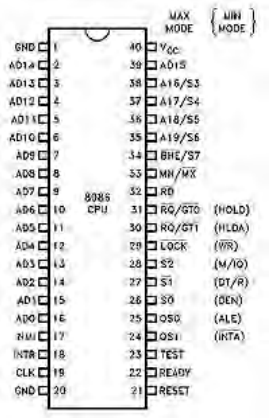
MTM 305 MİKROİŞLEMCİLER

Arş. Gör. Emel SOYLU
Arş. Gör. Kadriye ÖZ

Mikroişlemci Mimarisi

Mikroişlemcinin tanımı

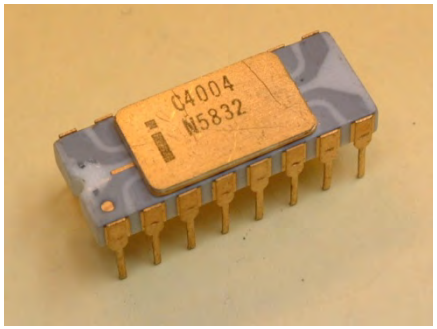
Mikroişlemci, işlemci ana işlem biriminin (CPU- Central Process Unit) fonksiyonlarını tek bir yarı iletken tümdevrede (IC-Integrated Circuit) birleştiren programlanabilir bir sayısal elektronik bileşendir. Kullanıcı ya da programcı tarafından yazılan programları meydana getiren komutları veya bilgileri yorumlamak ve yerine getirmek için gerekli olan tüm mantıksal devreleri kapsar. Bu devreler genelde transistörlerden meydana gelmektedir.

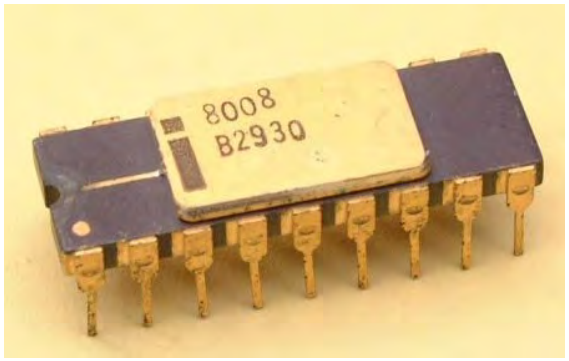


Mikroişlemciler

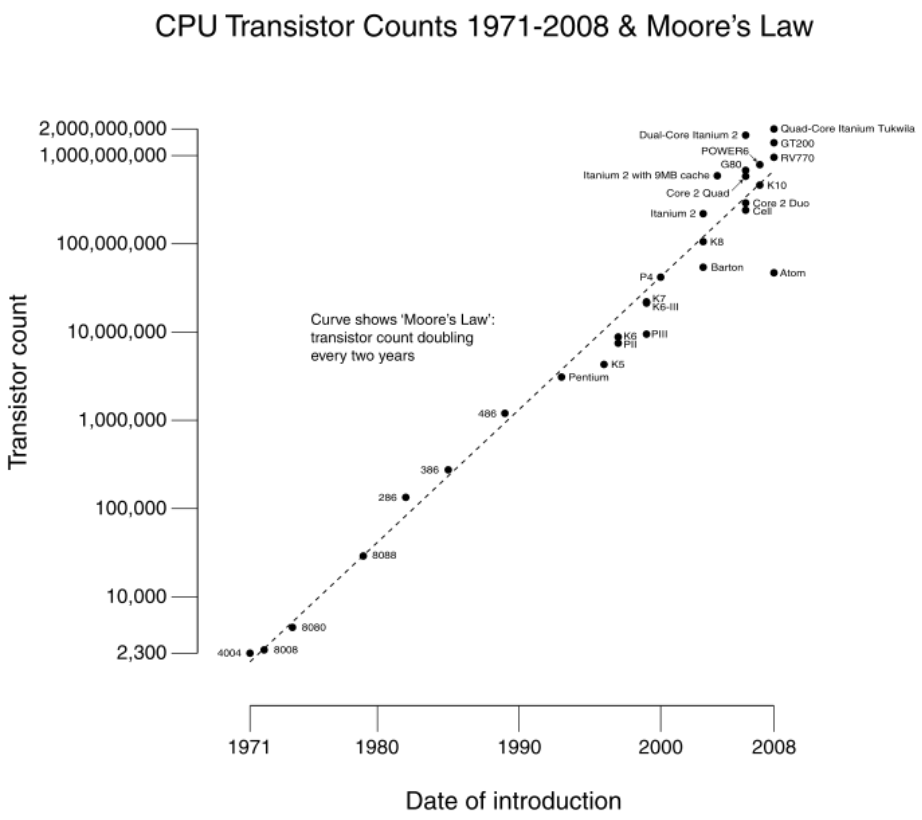
İlk mikroişlemci 1971 yılında hesap makinesi amacıyla üretilen Intel firmasının 4004 adlı ürünüdür. Bu kesinlikle hesap makinelerinde kullanılmak üzere üretilmiş ilk genel amaçlı hesaplayıcıdır. Bir defada işleyebileceği verinin 4 bit olmasından dolayı 4 bitlik işlemci denilmekteydi.

1974 ve 1976 yılları arasında 8 bitlik ilk genel amaçlı mikroişlemci denilebilecek mikroişlemciler tasarlanmıştır.





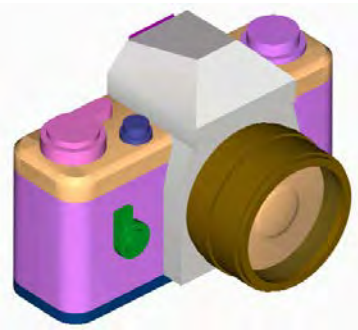
Mikroişlemciler



http://en.wikipedia.org/wiki/Transistor_count

Mikroişlemci kullanım alanları

Günümüzde, en büyük ana bilgisayarlardan, en küçük el bilgisayarlarına kadar her sistem çekirdeğinde mikroişlemci kullanılmaktadır.



Mikroişlemcinin Görevleri

- Sistemdeki tüm elemanlar ve birimlere zamanlama ve kontrol sinyali sağlar.
- Bellekten komut alıp getirir.
- Komutun kodunu çözer.
- Komutun operandına göre, veriyi kendisine veya G/Ç birimine aktarır.
- Aritmetik ve mantık işlemlerini yürütür.
- Program işlenirken, diğer donanım birimlerinden gelen kesme taleplerine cevap verir.

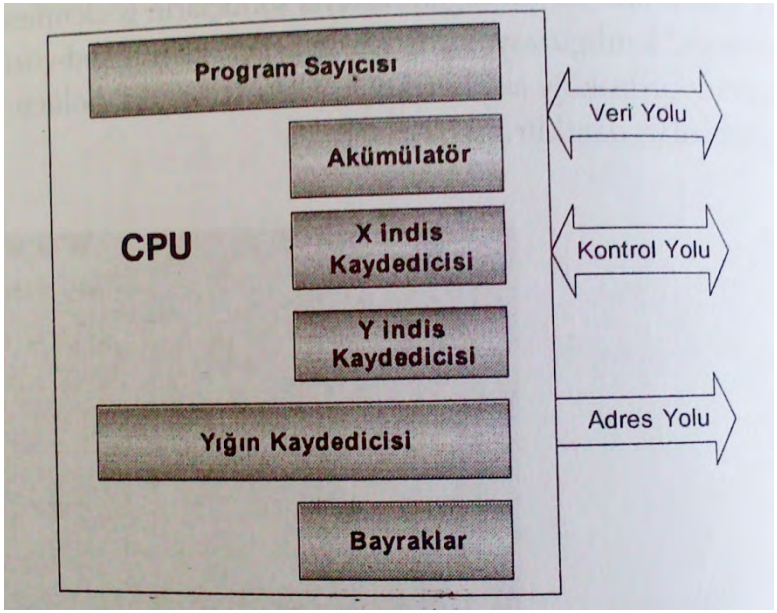
Mikroişlemcinin Yapısı

Mikroişlemcilerin yapısında aşağıdaki birimler bulunmaktadır.

- Kaydediciler
- Aritmetik ve Mantık Birimi
- Zamanlama ve Kontrol Birimi

Kaydediciler

İşlemci içerisinde ham bilgi girdisinin hızlı biçimde işlenerek kullanılabilir çıktıya dönüştürülmesi için sistemde verileri geçici olarak üzerinde tutacak bir gurup veri saklayıcıya ihtiyaç duyulmaktadır. Kaydediciler verinin manevrasında ve geçici olarak tutulmasında görevlidirler.

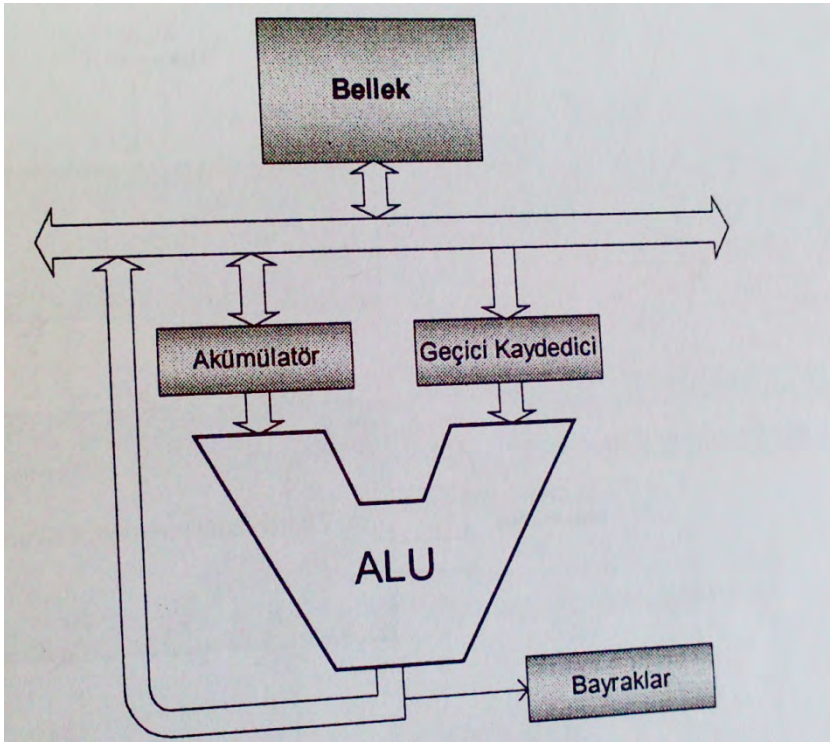


Aritmetik ve Mantık Birimi (ALU)

ALU mikroişlemcide aritmetik ve mantık işlemlerinin yapıldığı en önemli birimlerden birisidir.

Aritmetik işlemler denilince başta toplama, çıkarma, çarpma, bölme, mantık işlemleri denilince AND,OR, EXOR ve NOT gibi işlemler akla gelir. Komutlarla birlikte bu işlemleri mantık kapılarının oluşturduğu toplayıcılar, çıkarıcılar ve kaydıran kaydediciler gerçekleştirirler. ALU'da gerçekleşen bütün bu işlemler kontrol sinyalleri vasıtasıyla Zamanlama ve Kontrol Biriminin gözetiminde eş zamanlı olarak yapılır.

Aritmetik ve Mantık Birimi (ALU)

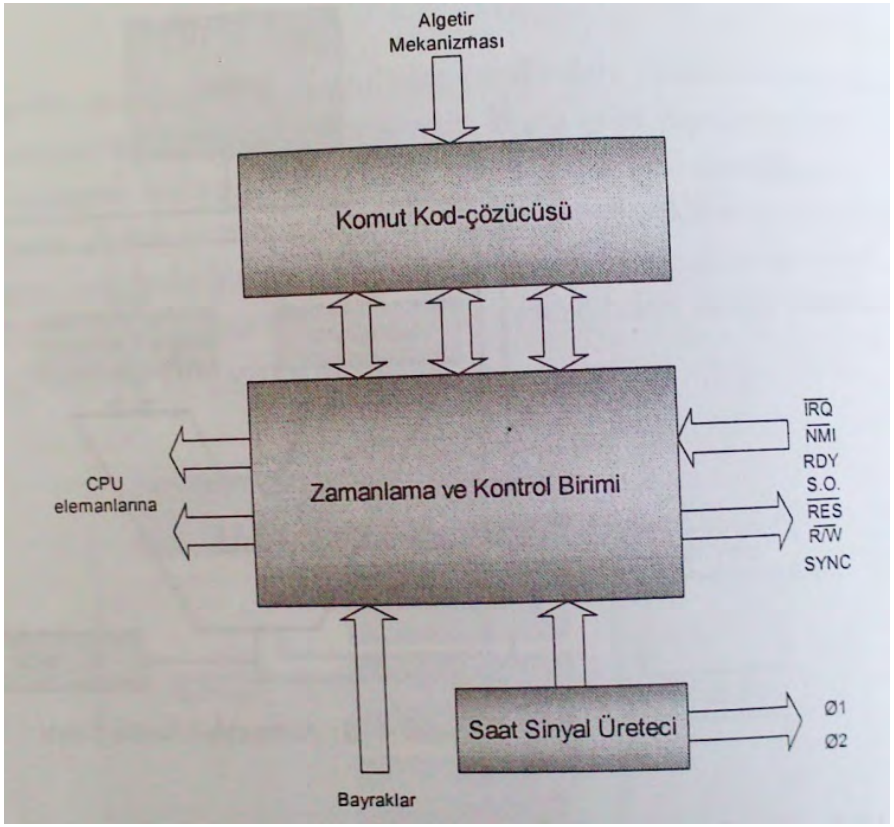


Aritmetik ve Mantık Birimi (ALU)

ALU'da basit matematik komutlar zorlanmadan işlenebilir fakat karmaşık aritmetik işlemleri (doğrudan bir komutla çarpma, bölme, karekök alma) gerçekleştirmek için ayrı altyordam gruplarına veya ek elektronik devrelere ihtiyaç duyulur. Eğer ek devre konulmamışsa mevcut devrelerle bu işlemleri gerçekleştirmek için birbiri ardına aynı komutu defalarca işlemek gereklidir, bu da zaman kaybı demektir. Gelişmiş mikroişlemcilerde bu devreler yerleşik vaziyettedir.

Gelişmiş işlemlerde kayan noktalı aritmetik işlemleri gerçekleştirmek üzere FPU (Floating Point Unit) bir işlemci daha yerleştirilmiştir. Bu sayede küçük haneli veya küçük kesirli sayılarla işlem yapılabilmektedir. Eğer sistemde FPU mevcutsa ağır matematiksel işlemler bu işlemci tarafından yapılırken ana işlemci diğer işlemlerle meşgul olacağından sistemde yavaşlama yok denecek kadar az olur.

Zamanlama ve Kontrol Birimi



Zamanlama ve Kontrol Birimi

Zamanlama ve kontrol birimi, bellekte program bölümünde bulunan komut kodunun alınıp getirilmesi, kodunun çözülmesi, ALU tarafından işlenilmesi ve sonucunun alınıp belleğe geri konulması için gerekli olan kontrol sinyalleri üretir. Bilgisayar sisteminde bulunan dahili ve harici bütün elemanlar bu kontrol sinyalleri ile denetlenir.

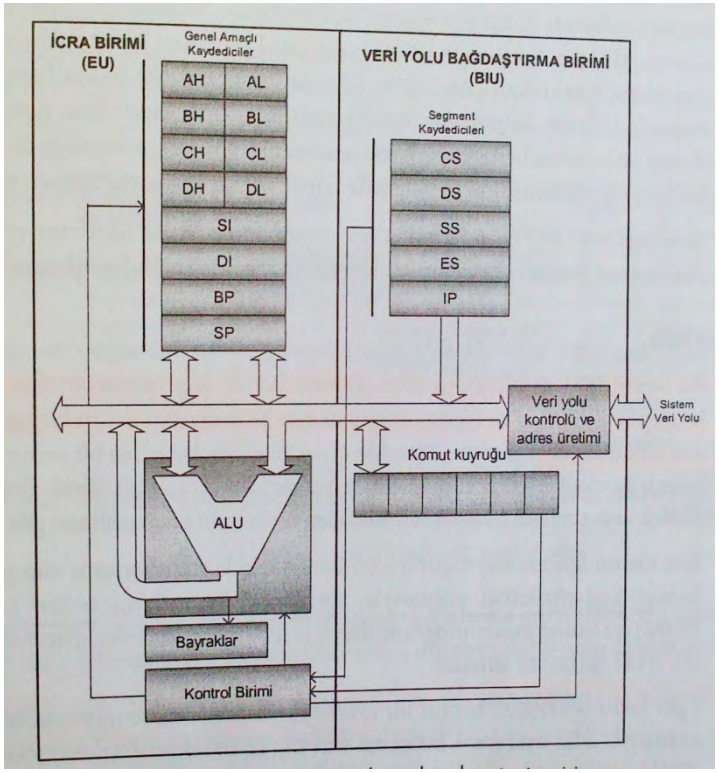
Basit bir mikroişlemcide bu bölüm 3 değişik işlevi yerine getirir:

1. Zamanlama Kontrolü: İşlemci, harici saat sinyali üreten bir birimden giriş alan iç-saat devresine sahiptir. Bir sinyal alınarak talebe göre zaman sinyallerine çevrilerek sisteme dağıtılır.
2. Komut kod çözücüsü: Bu devre komut kaydedicisinde tutulan komutları yorumlar ve ALU'ya kaydedicilerle çalışması için uygun sinyaller gönderilir.
3. Kesme ve Mantık Birimi:Bu birim diğer kontrol elemanlarına benzer.Gerekli durumlarda kesme sinyallerini alarak işlemciyi uyarır.

16 Bitlik İşlemciler

16 bitlik mikroişlemciler basit olarak 8 bitlik işlemcilerde olduğu gibi Kaydedici bölümü, ALU ve Zamanlama-Kontrol birimine sahiptir. Fakat mimari yapısı çoklu görev ortamına uygun hale getirildiğinden, işlemci içerisindeki bölümler de fonksiyonel açıdan iki mantıksal ana bölüm halinde daha ayrıntılı olarak açıklanmaktadır. 16 bitlik x86 tabanlı işlemciler veri yolu bağdaştırma birimi (BIU) ve icra birimi (EU) olmak üzere iki ana bölümde toplanabilir. BIU birimi, EU birimini veriyle beslemekten sorumluyken, icra birimi komut kodlarının çalıştırılmasından sorumludur. BIU bölümüne segment kaydedicileriyle birlikte IP ve komut kuyrukları ve veri alıp getirme birimleri dahilken, EU bölümüne genel amaçlı kaydediciler, kontrol birimi, aritmetik ve mantıksal komutların işlendiği birim dahildir. Bu birimler çip üzerinde birbirine fiziksel olarak bağlıdır.

16 Bitlik İşlemciler



Veri Yolu Bağdaştırma Birimi (BIU)

İşlenecek komutların kodları sistem tarafından bellekle ilgili segmentlerdeki adreslere yerleştirilir. Bellekteki bu komutlar çalıştırılacağı zaman doğrudan veri yolu bağdaştırma birimi tarafından bellekten alınarak kod çözme birimine getirilirler. Ayrıca icra edilen bir komutun sonucu belleğe yazılacağı zaman veriyolu bağdaştırma biriminden geriye yazma talep edilir.

İcra biriminin komutlarıyla verinin alınıp getirilmesi veya belleğe depolanması sırasında veri bağdaştırma birimi otomatik olarak getirilen verilerdeki komutları adına tampon da denilen işlemcinin tipine göre kapasitesi değişen komut kuyruğuna yerleştirir. Bellekten alınıp getirilecek ve işlenecek bu komutların yeri CS:IP kaydedici ikilisi tarafından belirlenir. Bellekte işlenecek programın ilk komutunun bulunduğu adres bu birim tarafından otomatik olarak CS:IP kaydedicilerine yerleştirilir.

Bellekten veya I/O birimlerinden her ne yöntemle olursa olsun bir şekilde komut kuyruğuna getirilen komut kodları ve operand bilgileri icra birimi tarafından işlenmek için hazır durumdadır.

İcra Birimi (EU)

Bu birim, BIU ile birlikte paralel çalışarak komut kuyruğuna sürülen makine dilindeki komutların kodunun çözülmesi ve işlemci içerisinde her bir komutun doğru bir biçimde ele alınarak işlenmesinden sorumludur. Eğer komutun işlenmesi sırasında herhangi bir veriye gerek duyulursa ve veri genel amaçlı kaydedicilerden birindeyse alınıp getirilmesi, eğer veri harici ortamdaysa (bellek veya çevresel cihazlarda) BIU'dan bu verinin talep edilmesi gibi işlemleri EU gerçekleştirir.

İcra birimi komut kuyruğunun en üstündeki komut kodunu alıp getirdikten sonra, kodalr bir komut-kod çözücü vasıtasıyla, içerisinde komutların mikro karşılıkları olan (mikro-kod ROM) ve adına mikroprogram denilen bir mikro-kod sıralayıcısının kontrolü altında çözülerek ALU birimine sürülür.

Eğer kodu belirlenen kod bir aritmetik ya da mantık komutuysa, icra birimindeki adres üreteci vasıtasıyla BIU uyarılarak bellekten veri alıp getirilerek icra birimindeki kaydedicilere veya doğrudan ALU'ya yollar ve kontrol biriminin denetiminde bu veriyle ne yapılması gerekiyorsa o yapılır.

İcra Birimi (EU)

Komutlar işlendikten sonra sonuç bilgisi nereye depolanacaksa (bir kaydediciye veya bellek alanına) oraya gönderilir. Bir komutun işlenmesinden sonra komut kuyruğunda bulunan sıradaki komutun ele alınması için kontrol birimi sinyal gönderir.

İlk mikroişlemcilerde bir komutun işlenmesi üç aşamada yapılmaktaydı.

1. Komut kodunun bellekten alınıp getirilmesi
2. Kodunun çözülmesi
3. İşlenmesi

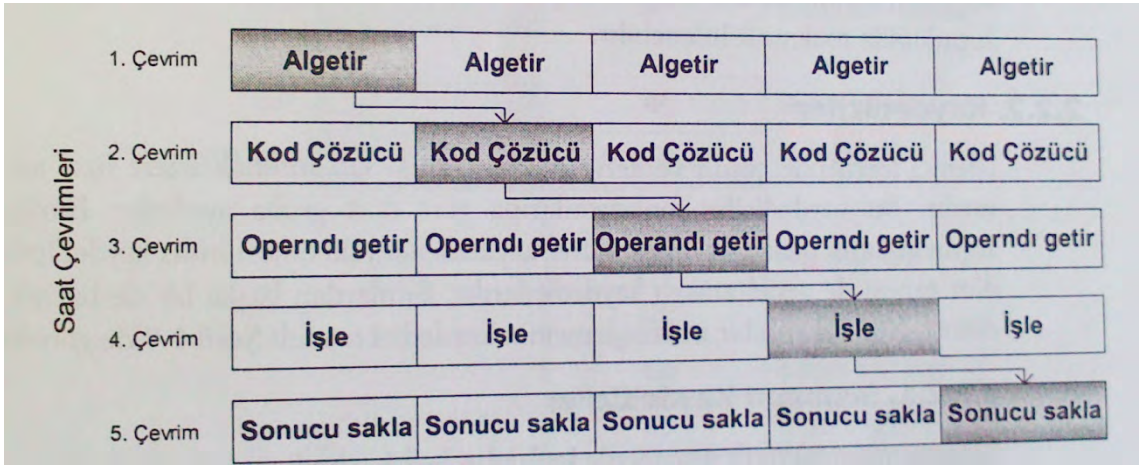
Bir komut kodu üç aşamalı iş hattına girerek belli saat çevrimlerinde işlenmekteydi. Bu teknikte iş hattına giren bir komut icra edilmeden ikinci bir komut hatta alınmıyordu. Bir komut bir kademede işlendikten sonra diğer kademeye geçtiğinde önceki kademe boş duruyordu ve böylece işlemci çevrimleri boşa harcanıyordu. Bu tip çalışma sistemine tek kademeli iş-hattı tekniği denmektedir.

İcra Birimi (EU)

Daha sonraları mikroişlemci mimarisindeki gelişimlerle (32 bit işlemciler) iş hattına alınan bir komut bir sonraki safhaya geçtiğinde boşalan safhaya başka bir komut alınmaktadır. Böylece her saat çevriminde iş hattındaki tüm kademeler bir iş ile meşgul olmakta ve toplamda birden çok işlem yapılmaktadır. Bu tip işlemci çalışma sistemine süper iş hatlı teknik denmektedir.

Kısaca iş hattı tekniği, mikroişlemcinin farklı kaynaklar kullanarak komutları ardışık adımlara bölmesi işlemine denir. Bundan dolayı işlemci bir çok işlemi paralel olarak aynı anda yapar.

İcra Birimi (EU)



İcra Birimi (EU)

Mikroişlemcinin her defasında komut alıp getirmesi ve onu işlemesi sistemi yavaşlatır. Bu yavaşlığı gidermek için iş hattı tekniği yeterlidir. Fakat tekrar belleğe erişilmek istendiğinde bu defada belleğin işlemci hızında olmaması ve ona gerektiği hızda cevap vermemesi işlemci tasarımcısına bir sorun daha ortaya çıkarmıştır. Burada verinin akışını hızlandıran ve sistemin bir bütün halinde aynı hızda çalışmasını sağlayan bir dizi tedbirler alınmıştır.

Günümüz işlemcilerinde bu tip sorunlar Dinamik Çalışma adı altında işlemciye kazandırılan yeteneklerle yok edilmeye çalışılmıştır. Dinamik çalışmaya dahil olan öğeler şunlardır:

- Mikroişlemcinin bir sonraki komutla birlikte ele alacağı komut gurubunun tahmin edilmesi işlemine Çoklu Dallanma Tahmini denir.
- Komutlar arasındaki bağımlılıkları analiz eden bir veri akış analizi ve
- İlk iki öğenin sonuçlarını kullanarak komutları spekülatif çalıştırma işlemidir.

Sistemdeki elemanların hızlarının birbirine yakın olmasının yanında ön bellekler ve süperskalar mimarisi hızda önemli rol oynamaktadır.

Aritmetik ve Mantık Birimi (ALU)

İcra birimindeki komutun kodu çözüldükten sonra yapılacak işlem aritmetik ya da mantık işlemi olacaktır. Bu işlemin yürütülmesi işi ALU (Arithmetic and Logic Unit) birimine verilmiştir. Bu birimde bayt veya word olarak basit dört işlemin yanında verinin bit olarak artırılması ya da azaltılması, sağa sola kaydırılması veya yönlendirilmesi işlemleri yapılabilir. Aynı zamanda ALU bayt veya word verisi üzerinde bit-bit mantıksal işlemlerin yapılması için de çok elverişlidir. AND,OR ve XOR komutlarıyla, iki bayt veya iki word verisi arasında bit eşlemesi işlemleri yapılabilir.

İlk mikroişlemcilerde çarpma bölme gibi aritmetik komutların yerine getirilmesinde sağa veya sola kaydırma komutlarından faydalanılmaktaydı, fakat 16 bitlik işlemcilerle birlikte mimarinin gelişmesine paralel olarak komut kümelerindeki artış, bu işlemleri doğrudan yapabilecek MUL ve DIV gibi komutları da beraberinde getirmiştir. Hangi komut kullanılırsa kullanılsın toplayıcı ve çıkarıcı devreleri çarpma ve bölme işlemlerinde çarpma için üstüste toplama, bölme için üstüste çıkarma şeklinde kullanılmaktadır.

Aritmetik ve Mantık Birimi (ALU)

ALU gerekli işlemi bitirdikten sonra adına bayrak kaydedicisi denilen hücreye yazılacak durum bitlerini kontrol eder. Bu bitler bir sonraki işlenecek komuta etki edebilir. Mikroişlemci tüm bu işlemlerin düzenli bir şekilde yürütülebilmesi için veri yolu bağdaştırma birimi ile icra biriminin paralel olarak çalışması gerekir. Herhangi bir kopuklukta makine kilitlenebilir.

Kaydediciler

İşlemci içerisinde çeşitli verilerin manevrasında kullanılmak üzere özel amaçlı 14 kaydedici vardır. Bu kaydediciler fonksiyonlarına göre dört gruba ayrılırlar: Bunlardan

4 tanesi segment kaydedicisi
3 tanesi işaret kaydedicisi
2 tanesi indis kaydedicisi
5 tanesi genel amaçlı kaydedicilerdir.

Bunlardan başka bir de bayrak kaydedicisi mevcuttur.

Segment Kaydediciler

Büyük kapasiteli belleklerde bilginin yönetimi (yüklenmesi, saklanması ve sırasını beklemesi) oldukça karmaşıktır. Bu sebeple büyük bellekler belli amaçlarla 64Kbaytlık küçük gruplara (segmentlere) ayrılarak daha kolay yönetilirler. Bellekte bu bölümlerin başlangıç adresi segment kaydedicileri tarafından tutulurlar. Bu bölümdeki verilerin adresleri ise, segment kaydedici içeriğine uzaklığıdır ve ofset adres olarak anılırlar.

Gösterdikleri bellek alanlarının oluşumuna göre segment adları özel amaçları olarak verilmiştir. Buna göre programcı tarafından yazılan komut kodlarının assemblera bağlı olarak sistem tarafından bellekte saklandığı bölüme kod segment adı verilir. Kod segmentteki komutlarla ilgili olan veya bu kodların işleyeceği verilerin saklandığı başka bir bölüme data segmenti denilir. Verilerin çok büyük olduğu istatistiksel bilgilerin tutulduğu data segmentinin yetmediği durumlarda ikinci bir bölüm vardır ki buna ekstra segment adı verilir. Networking ve korumalı mod işlemlerinde bu data segmentlerinin büyüklüğü de yetmezse, 386 ve daha sonraki işlemcilerde F segment ve G segment denilen bölümler taanımlanmıştır. Bunun anlamı bellek miktarının mikroişlemci mimarisi ile birlikte paralel olarak artması demektir.

Segment Kaydediciler

Mikroişlemcilerde komutların işlenmesi sırasında , kaydedicilerin yetersiz gelmesi durumunda verinin geçici olarak bellekte atıldığı özel bölüme Yığın Segmenti denilir. Bellekteki bu bölümlerin başlangıç adresi kendi adlarıyla işlemci içerisinde yer alan kaydedicilerde tutulur.

Kod segment kaydedicisi: Kod segment bellekte çalıştırılacak komutların sıralı bir şekilde bulunduğu bölümdür.

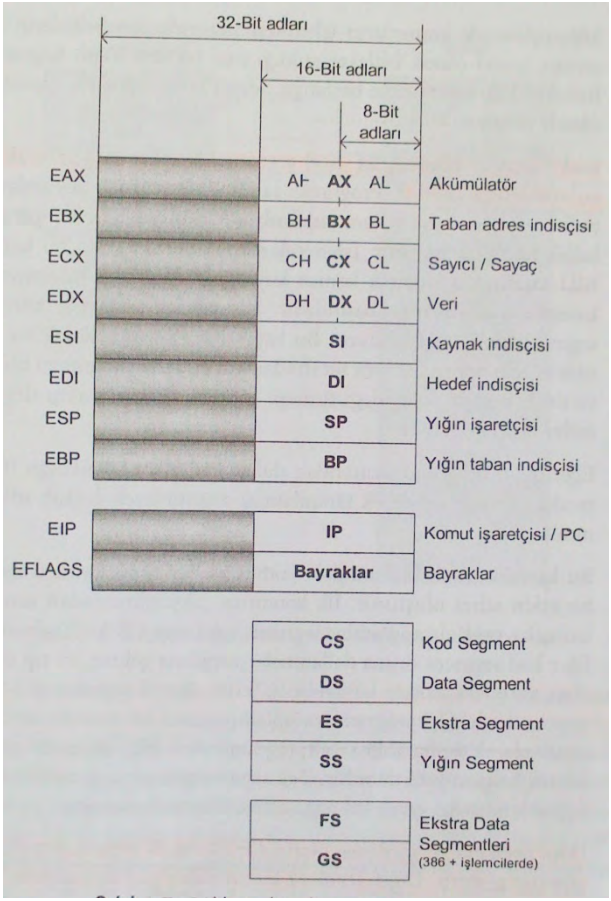
Data Segment Kaydedicisi: Normalde işlenecek verinin depolandığı bellek alanının başlangıç adresini gösterir. Diğer segment kaydedicileri gibi tam adresin segment tarafını gösterir.

Ekstra segment kaydedicisi: ES olarak adlandırılan bu kaydedici, programcı tarafından tanımlanmadıkça işlemci bunu kullanmaz. Genellikle string işlemlerinde hedef adresi olarak algılanır.

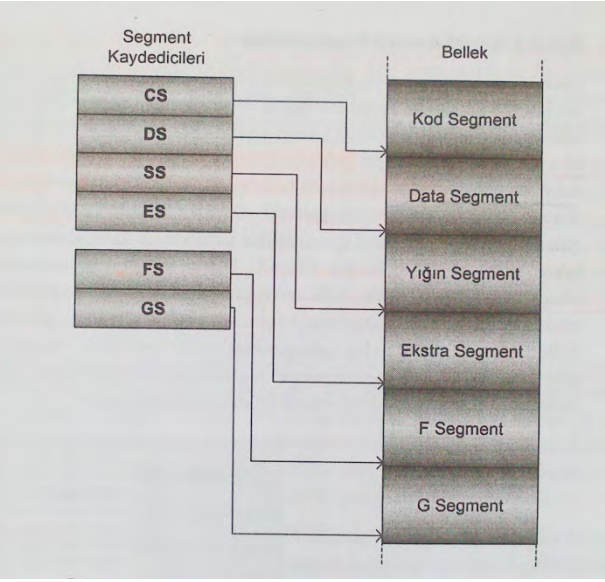
Segment Kaydediciler

Yığın Segment Kaydedicisi: Kısaca SS olarak bilinen bu kaydedicinin gösterdiği bellek alanına verilen ad adından da anlaşılabileceği gibi bir takım veri işlenirken yer yokluğundan veya kaydedici yetersizliğinden dolayı verinin geçici olarak yerleştirildiği yerdir. Yığına veriler geçici olarak atılabildiği gibi, program içerisinde altyordam çağrılmasında veya diğer yüksek düzeyli dillere parametre geçişleri yapılmasında çok sık kullanılmaktadır.

Segment Kaydediciler



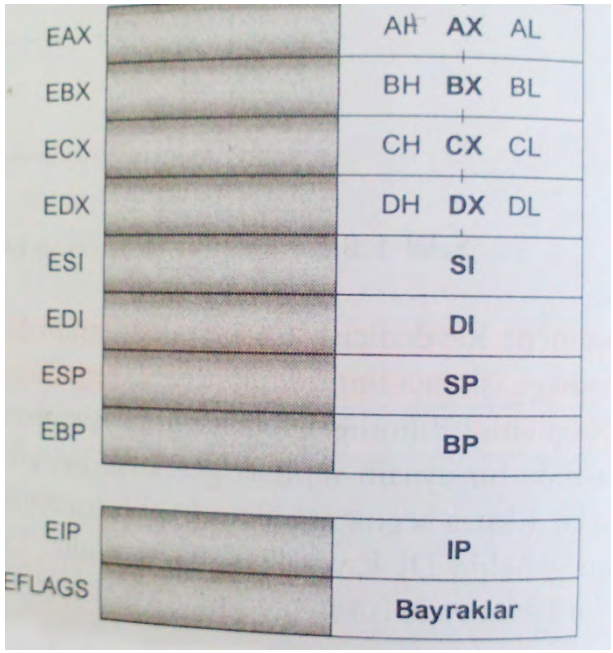
Segment Kaydediciler



Genel Amaçlı Kaydediciler

Genel amaçlı kaydediciler , mikroişlemcide program komutlarının icrası sırasında verinin manevrasında kullanılan ve yapısal olarak en küçük bölümü 8 bitlik bellek hücresine benzeyen elektronik elemanlardır. Genel amaçlı kaydediciler kendi aralarında yaptıkları işe göre iki gruba ayrılırlar. Birinci grupta çok amaçlı kaydedicilere EAX,EBX,ECX,EDX,ESI,EDI ve EBP dahilken, ikinci özel amaçlı kaydediciler grubuna ESP,EIP ve Bayrak kaydedicisi girer. 386'Ya kadar bu kaydediciler 16 bitlik AX,BX,CX ve DX olarak işlem görmüşlerdir. Daha küçük 8 bitlik verilerin (bayt) işlenmesinde kullanılmak üzere daha da ufak parçalarla tanımlanabilmektedir. AH,AL,BH,BL,CH,CL,DH ve DL gibi. AX serisi kaydediciler 16 bitlik verilerin saklanmasında EAX serisi kaydediciler 32 bitlik verilerin saklanmasında kullanılmaktadır. Kaydedici kısaltmasındaki X'in manası H (High) ve L (Low)'un birlikte kullanımı E ise Extenden (genişletilmiş) manasına gelmektedir.

Genel Amaçlı Kaydediciler



Genel Amaçlı Kaydediciler

AX kaydedicisi:Akümülatör AX koduyla tanımlanır ve verilerin ilk ele alınmasında başrol oynadığından baş kaydedici olarak düşünülebilir.8,16 ve 32 bitlik verilerle çarpma, bölme bazı I/O işlemlerinde ve bazı harfdizi işlemlerinde etkin bir biçimde kullanılmaktadır.

BX kaydedicisi: Taban adres kaydedicisi olarak bilinen ve BX koduyla tanımlanan kaydedici, bellekteki veri gruplarının ofsetinin tutulmasında bir indisçi gibi davranır. Ayrıca hesaplamalarda ve 32 bitlik işlemcilerde bellekteki verinin adreslenmesinde de kullanılmaktadır.

CX Kaydedicisi: Sayaç kaydedicisi olarak bilinen CX, string işlemlerinde bir sayaç elemanı veya döngü işlemlerinde tekrarlama sayıcısı gibi işlevleri yerine getirir.

DX kaydedicisi: Data kaydedicisi diye tanımlanan DX kaydedicisi, genellikle akümükatöre yardımcı olan bütün işlemlerde bir tampon gibi davranan kaydedicidir.

Genel Amaçlı Kaydediciler

İşaretçi ve İndis Kaydedicileri: Mikroişlemcili sistemlerde bellekteki ara adresleri gösteren kaydedicilere işaretçi adı verilir.

Bayrak kaydedicisi: Bayrak kaydedicisi bir işleminde sonucun ne olduğunu kaydedici bitlerine yansıtan bir bellek hücresini oluşturur. Bu kaydediciye bayrak denmesinin sebebi, karar vermeye dayalı komutların yürütülmesinde sonuca göre daha sonra ne yapılacağını bit değişimiyle bu kaydedicinin 1 bitlik hücrelerine yansıtmasıdır. Kaydedici bitlerindeki mantısal 1 bayrak kalktı, 0 bayrak indi demektir. Karşılaştırma ve artimetik komutların çoğu bayraklara etki eder.

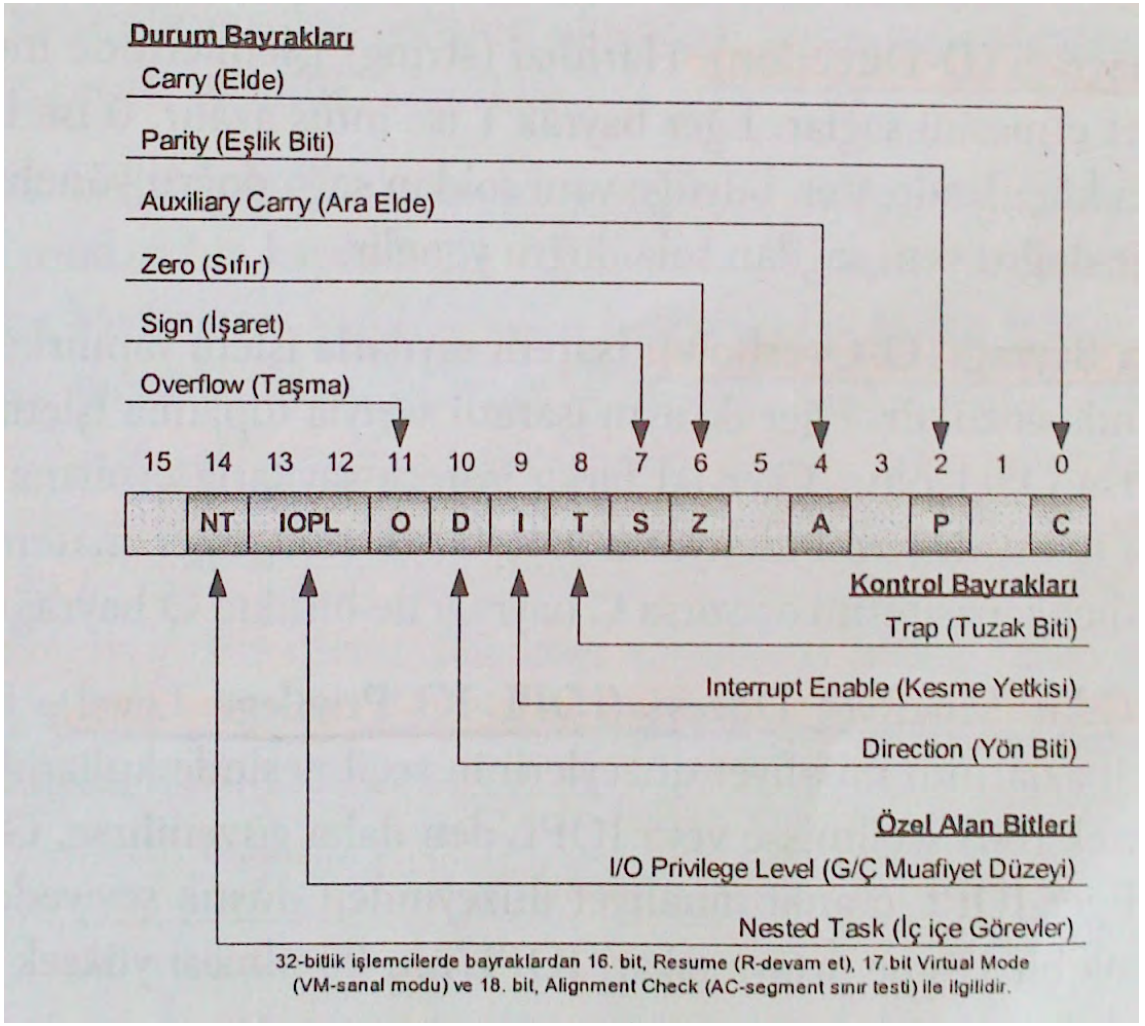
Bayraklar

Elde Bayrağı (C-Carry): Eğer toplama sonucunda elde, çıkarma sonucunda borç ortaya çıkıyorsa C=1 aksi taktirde 0 olur. Aynı zamanda C bayrağı kaydırma ve yönlendirme işlemleri sonucunda kaydedicinin MSB bitindenveya LSB bitinden düşen verileri üzerinde tutar ve karşılaştırma işlemlerinin sonucunu yansıtır. Ayrıca C bayrağı çarpma işlemi için sonuç göstericisi gibi hareket eder.

Eşlik biti (P-parity): İşlemin sonucunda kaydedicideki mantıksal birlerin sayısı çift ise P=1 aksi halde P=0 olur. Eşlik biti genelde veri iletişiminde karşılıklı verilerin güvenli iletilip iletilmediğinin kontrolünde kullanılır.

Yardımcı Elde Bayrağı: (AC-Auxilary Carry): Eled bayrağı ile aynı işlemi görür fakat sadece 3. bitten bir fazlalık ortaya çıkarsa bu bayrak 1 aksi durumda 0 olur. AC bayrağı paketlenmiş ondalık verilerin işlenmesinde çok kullanışlıdır.

Bayraklar


Durum Bayrakları
Carry (Elde)
Parity (Eşlik Biti)
Auxiliary Carry (Ara Elde)
Zero (Sıfır)
Sign (İşaret)
Overflow (Taşma)
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
NT IOPL O D I T S Z A P C
Kontrol Bayrakları
Trap (Tuzak Biti)
Interrupt Enable (Kesme Yetkisi)
Direction (Yön Biti)
Özel Alan Bitleri
I/O Privilege Level (G/Ç Muafiyet Düzeyi)
Nested Task (İç içe Görevler)
32-bitlik işlemcilerde bayraklardan 16. bit, Resume (R-devam et), 17.bit Virtual Mode (VM-sanal modu) ve 18. bit, Alignment Check (AC-segment sınır testi) ile ilgilidir.

Bayraklar

Sıfır Bayrağı (Z-Zero): İşlem sonunda sonuç 0 ise Z=1 aksi halde Z=0 olur. Mesela bu işlem sonunda AX kaydedicisindeki değer 0000 ise sıfır bayrağı 1 olur diğer durumlarda bayrak 0 kalır.

İşaret bayrağı (S-Sign): İşaretli sayılarla yapılan işlemlerde bu bayrak anlam ifade etmektedir. Eğer aritmetik mantık, kaydırma ve yönlendirme işlemleri negatif sonuç üretiyorsa S=1 aksi halde S=0 olur. Diğer bir deyimle S bayrağı sonucun 8 bit veya 16 bit olmasına bakılmaksızın MSB bitini yansıtır.

Tuzak Bayrağı (T-Trap): Hata ayıklama işlemlerinde komutların adım adım işlenmesi maksadıyla kullanılır. Bayrak 1 yapıldığında Debug işlemi yapmak için komutler tek tek çalıştırılır.

Kesme Yetkilendirme bayrağı (I-Interrupt Enable): Sisteme bağlı harici cihazlardan gelen kesme taleplerine izin verir. I bayrağının 0 olması kesme isteklerine cevap verilmemesini sağlar. Ancak I=1 olduğunda tekrar istekler göz önüne alınır.

Bayraklar

Yön Bayrağı (D-Direction): String işlemlerinde indis kaydedicisinin ileri yada geri hareket etmesini sağlar. Eğer bayrak 1 ise indis azalır, 0 ise indis değeri artar. Eğer D=0 ise, işlemci küçük adresten büyüğe yani soldan sağa doğru yönelir. Eğer D=1 ise, büyük adresten küçüğe doğru yani sağdan sola doğru yönelir. Eğer D=1 ise büyük adresten küçük adrese yani sağdan sola doğru yönelir.

Taşma bayrağı(O-Owerflow): İşaretli sayılarla işlem yapılırken bir hatanın ortaya çıkması durumunda gözükür. Eğer iki aynı sayıyla toplama işlemi yapılıyor ve sonuç farklı işaretli çıkıyorsa O=1 olur. Eğer matematik bir işlem sonucunda sonuç kaydedici kapasitesini aşıyorsa C bayrağı ile birlikte O bayrağı da 1 olur.

Giriş/Çıkış Muafiyet düzeyi (IOPL-IO Privilege Level): Korumalı mod operasyonlarında G/Ç cihazlarının muafiyet düzeylerinin seçilmesinde kullanılır. Eğer o andaki muafiyet düzeyi yüksek seçilmişse veya IOPL'den daha güvenilirse, G/Ç herhangi bir engellemesiz çalışır.

Bayraklar

İçiçe Geçmiş Görevler (NT-Nested Task) Korumalı mod operasyonlarında o andaki görevin başka bir görevle iç içe girmesi işlemidir. Görev başka bir görevle yazılım tarafından iç içe girdirildiğinde bu bayrak 1 olur.

İşleme devam (R-Resume): Hata ayıklama işlemlerinde (Debug), bir sonraki işlenecek komuta devam edilmesinin kontrolünde kullanılır.

Sanal Mod(VM-Virtual Mode) Korumalı mod sisteminde sanal mod işleminin seçilmesinde kullanılır. Sanal mod, DOS sisteminde belleğin birkaç parçaya bölümlenmesini sağlar.

Segment Sınır Tespiti(AC-Alignment Check): Eğer word veya doubleword tanımlamaları kendilerine uygun adres sınırlarında değilse bu bayrak 1'e kurulur. Bu bayrak sadece 486SX işlemcide kullanılmaktadır.

Bellek Adreslemesi

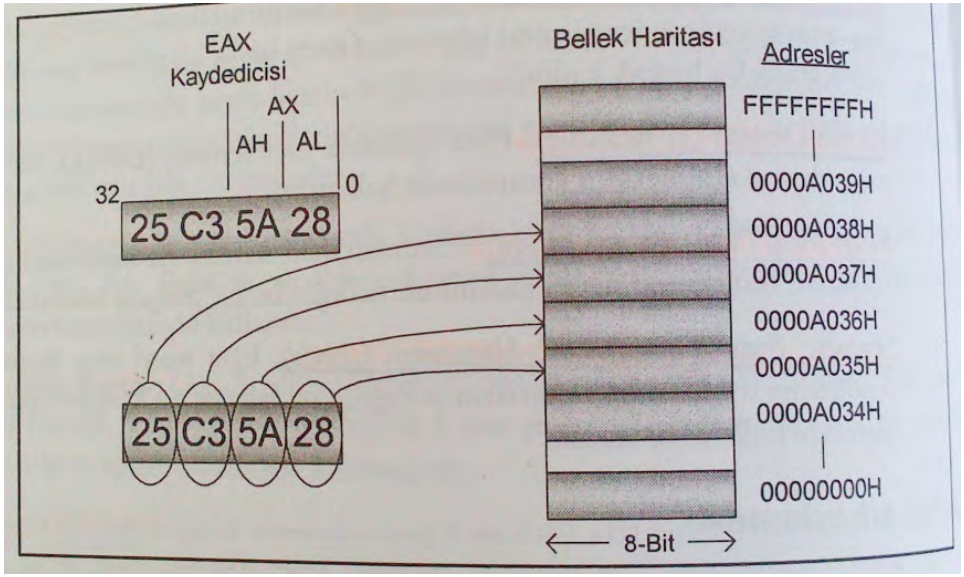
Mikroişlemciye dayalı sistemlerde adres uzayı fiziksel veya mantıksal bellek tanımlamasından birisiyle ilişkilendirilir. Çoğu durumlarda mantıksal bellek yapısı fiziksel bellek yapısından farklıdır. Fiziksel bellek, bellek sisteminin gerçek donanım yapısıyken mantıksal bellek, bunun programcıya gözüken tarafıdır.

Mantıksal Bellek Tanımlaması

Mantıksal bellek tanımlamasında bütün adresler bayt olarak numaralandırılır ve programcı buna göre programını yazar ve uygular. 16 adres hatlı işlemcilerin adres numaraları 0000H ile başlar ve FFFFH ile sona erer. Burada tanımlanan adres uzayı 64 KB'dır. 32 adres hattına sahip işlemcilerin adres numaraları 00000000H ile başlar ve FFFFFFFFH ile sona erer. Bu işlemcileri kullanan sistemin adresleme kapasitesi böylece 4 GB olur.

Programcı mantıksal bellek yapısına göre bir baytlık veriyi doğrudan bir adres göstererek oraya depolar veya oradan bir kaydediciye yükleyebilir. Fakat bir word'lük bir verinin üzerinde tanımlanması, bellekte iki ardışık adrese ulaşacak demektir. Bu durumda verinin az değerlikli kısmı küçük nolu adrese, çok değerlikli kısmı büyük nolu adrese yerleştirilir. Bu işleme ters bayt sıralaması denir.

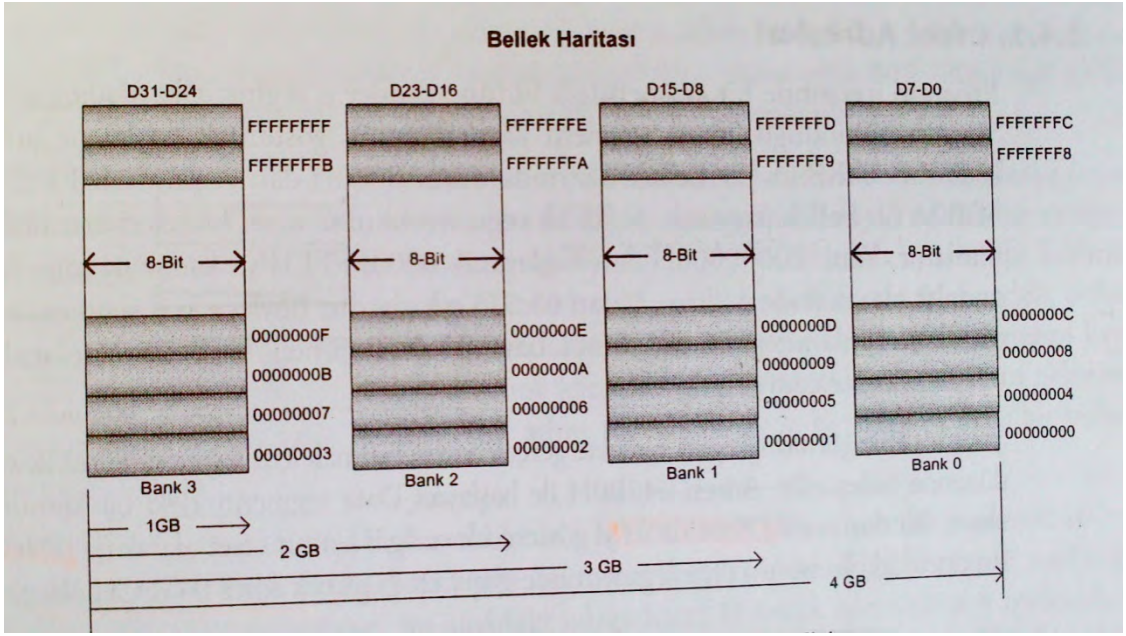
Mantıksal Bellek Tanımlaması



32 bitlik EAX kaydedicisindeki 25C35A28H verisi bellekte 0000A035H adresine saklanmak istenirse, dört ardışık bellek adresine erişim yapılması gereklidir. Bu işlem otomatik olarak işlemci tarafından yapılır programcı yalnızca başlangıç adresini belirtir.

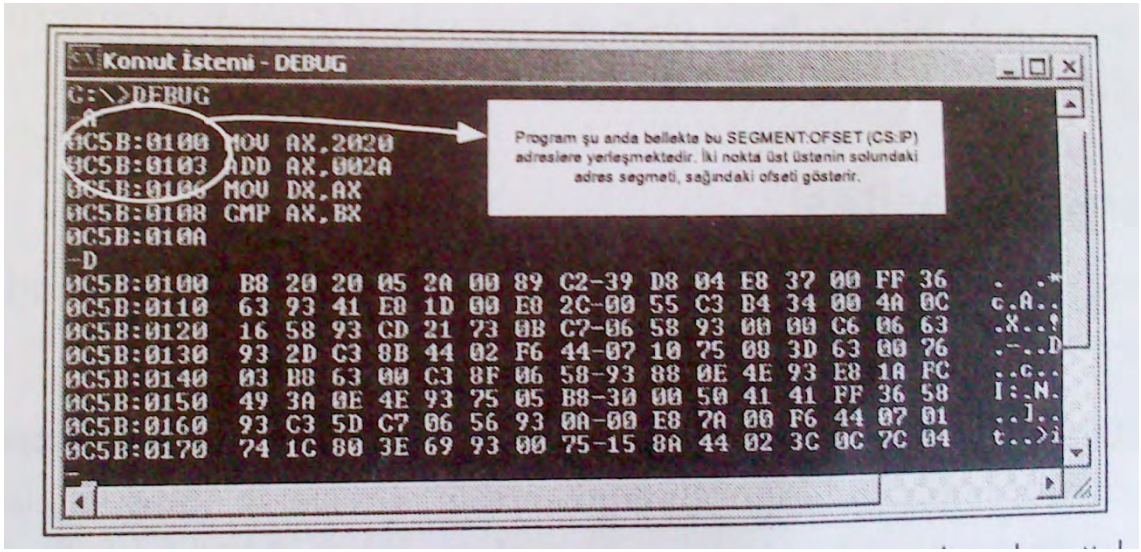
Fiziksel Bellek Tanımlaması

Belleklerin fiziksel tanımlanması donanımsal bir yaklaşımdır ve işlemci mimarisine bağlıdır. Günümüz işlemcilerinin bellek organizasyonu 8'er bitlik banketler halinde (sıralar) yapılır. 16 bitlik bellek düzeninde iki adet 8 bitlik banketle adresleme bayt yada word olarak yapılırken, 32 bitlik bellek düzeninde 4 adet 8 bitlik banketle adreslemeler bayt, word veya doubleword olarak yapılabilmektedir.



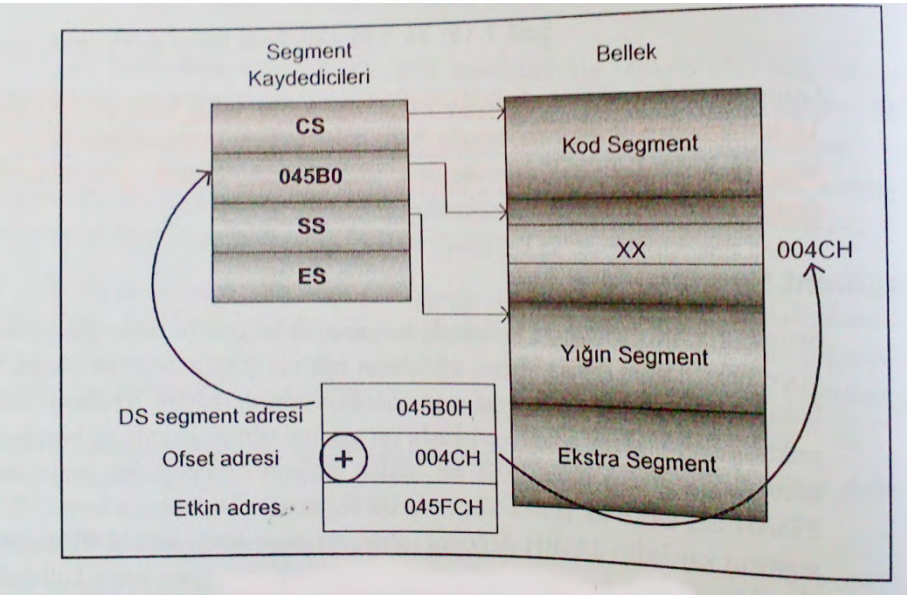
Segment Sınırları

Segmentler, adreslerin eşit bir şekilde ondalık olarak 16 ile bölünebilen ve adına paragraf sınırı denilen adresle başlar. Segment adres kaydedicileri daima 0H ile başlar. Mesela herhangi bir segment kaydedicisi içeriği 255AH ise, 64 KB'lık bellek alanında bu segmentin başlangıç adresi 255 A0H olacaktır.



Ofset adresleri

Program içerisinde bir segmentteki bütün adresler o segment adresine göre görecelidir. Verinin bulunduğu adres, segment kaydedicisinin gösterdiği başlangıç adresinden uzaklığı kadardır. 64 KB'lık bir bellek uzayında 0000H'dan başlayarak FFFFH'e kadar gider. Şekilde bellekte gerçek adresi bulmak için segment adres ile ofset adresi belli bir düzende birleştirilir. Adresi 045B0H ile başlayan data segmentindeki bir komutun adresi 004CH olsun. Bu durumda DS:045B0H'yi gösterecek ve ilgili komut ofset olarak ta 004CH^yi gösterecektir.Bu durumda komutun data segmentinde işaret ettiği gerçek adres 045FCH olacaktır.



CISC, RISC ve EPIC Esasları

Mikroişlemcinin temel unsurları kaydediciler, veri yolları ve iş hatlarıdır. Bu unsurların büyüklüğü, sayısı, yapısı o işlemcinin yeteneklerini belirler ve bir mimariyi diğer mimarilerden ayırır.

Bilgisayar tarihinin başlarında, donanım fiyatlarının yüksek oluşundan dolayı çoğu bilgisayar oldukça basit komut kümesine sahipti. Sonraki yıllarda donanımı oluşturan elemanların üretimindeki artış, fiyatların düşmesine bunun sonucunda sistemde yüksek miktarda eleman kullanılmasına sebep oldu. Böylece fazla donanım kullanımı, komut kümesinin büyümesini ve sistemi çok karmaşık yapan donanımlarda kullanılmasını sağlamıştır.

Bir bilgisayarın komut kümesi, programcının makineyi programlarken kullanabileceği ilkel emirleri veya makine komutlarının tamamının oluşturduğu kümeyi belirtir. Bir komut setinin karmaşıklığı, komut ve veri formatlarına, adresleme modlarına, genel amaçlı kaydedicilere, opcode tanımlamalarına ve kullanılan akış kontrol mekanizmalarına bağlıdır. İşlemci tasarımındaki komut seti mimarileri CISC ve RISC olmak üzere iki çeşittir.

CISC Mimarisi (Complex Instruction Set Computer-Karmaşık komut kümeli Bilgisayar)

Intel'in X86 mimarisine dayalı işlemci serisinin ortaya çıktığı 70'li yıllarda, RAM'lerin pahalı ve kısıtlı olması sebebiyle bu kaynakların tasarruflu bir şekilde kullanılarak yüksek seviyeli dillerin desteklenmesini savunan bazı tasarım mimarları bir araya gelerek CISC mimarisini geliştirmişlerdir. Bu mimari, programlanması kolay ve etkin bellek kullanımı sağlayan tasarım felsefesinin bir ürünüdür. Her ne kadar performans düşüklüğüne sahip olsa ve işlemciyi karmaşık hale getirse de yazılımı basitleştirmektedir.

CISC mimarisinin karakteristik iki özelliğinden birisi, değişken uzunluktaki komutlar, diğeri ise karmaşık komutlardır. Değişken ve karmaşık uzunluktaki komutlar bellek tasarrufu sağlar. Karmaşık komutlar iki ya da daha fazla komutu tek bir komut haline getirdikleri için hem bellekten hem de programda yer alması gereken komut sayısından tasarruf sağlar. Karmaşık komut karmaşık mimariyi de beraberinde getirir. Mimarideki karmaşıklığın artması, işlemci performansında istenmeyen durumların ortaya çıkmasına sebep olur. Ancak programların yüklenmesinde ve çalıştırılmasındaki düşük bellek kullanımı bu sorunu ortadan kaldırabilir.

CISC Mimarisi (Complex Instruction Set Computer-Karmaşık komut kümeli Bilgisayar)

Tipik bir CISC komut seti, değişken komut formatı kullanan 120-350 arasında komut içerir. Bir düzineden fazla adresleme modu ile iyi bir bellek yönetimi sağlar.

CISC mimarisi çok kademeli işleme modeline dayanmaktadır. İlk kademe yüksek düzeyli dilin yazıldığı yerdir. Sonraki kademeyi makine dili oluşturur ki, yüksek düzeyli dilin derlenmesi sonucu bir dizi komutlar makine diline çevrilir. Bir sonraki kademede makine diline çevrilen komutların kodları çözülerek, mikroişlemcinin donanım birimlerini kontrol edebilen en basit işlenebilir kodlara (mikrokod) dönüştürülür. En alt kademede ise işlenebilir kodları alan donanım aracılığıyla gerekli görevler yerine getirilir.

CISC Mimarisi (Complex Instruction Set Computer-Karmaşık komut kümeli Bilgisayar)

İlk mikroişlemci tasarımları, komut kümesindeki her bir komutun şifresini çözme ve sonra işleme şeklinde çalışan adanmış mantık kullandılar. Bu düzen birkaç kaydedici içeren basit tasarımlar için iyi bir çalışmaydı. Ancak yapımı oldukça zor ve daha karmaşık mimarilerin doğmasına sebep oldu. Bu yüzden tasarımcılar işlemcinin farklı birimleri arasındaki veriyollarını kontrol etmek için birkaç basit mantık geliştirdiler.

Veriyolu mantığını kontrol etmek için basitleştirilmiş komutlara mikrokod denilir ve bu tip bir uygulama mikroprogramlı uygulama olarak bilinir. Bir mikroprogramlı sistemde işlemcinin komut kodlarının her birine karşılık gelen mikrokod komut gruplarını içeren belleği (tipik olarak ROM) vardır. Bir makine kodu işlemciye eriştiğinde, işlemci kodun daha basit komutçuklara ayrılmış dizilerini icra eder.

Komutlar yerel bir ROM bellekte olduğundan ana bellekten on kat daha hızlı bulunup getirilebilirler. Bundan dolayı tasarımcılar mümkün olduğunca çok komutu mikrokod ROM'a koymaya çalışırlar. Gerçekte sık sık kullanılan ve belirli uygulamalarda yavaş yordamların yerine, bu yordamları içeren mikrokodlu mikroişlemciler üretilmektedir.

CISC Mimarisi (Complex Instruction Set Computer-Karmaşık komut kümeli Bilgisayar)

İçerisinde mikrokod bulunduran ROM bellek, ana bellekten çok daha hızlı olduğu için, mikrokod bellekteki komut serisi fazla hız kaybetmeksizin dahili sistemde yürütülebilir.

Aynı komut kümesini adanmış mantık üzerinde yürütmek yerine, yeni yongalarla yürütmek daha kolaydır ve daha az transistör gerektirir.

Bir mikroprogramlı tasarım yeni komut kümelerini işlemek için tamamen değiştirilebilir.

Yeni komutlar mikrokod halinde eskilerin üzerine eklenir. Böylece geriye dönük uyumluluk tam olarak sağlanabilir.

Bazı makineler ticari hesaplamalar için, bazıları da bilimsel hesaplamalar için elverişli hale getirildiler. Bununla birlikte tümü aynı komut kümesini paylaştığından programlar makineden makineye, temel donanımlara bağlı kalarak, performansın mümkün olan artırımı ve azaltımı ile birlikte yeniden derlenmeden taşınabilir. Bu esneklik ve güç, mikrokodlamayı yeni bilgisayarları geliştirmek için tercih edilen yol yapar.

CISC Mimarisi (Complex Instruction Set Computer-Karmaşık komut kümeli Bilgisayar)

Bir mikroprogramlı tasarımı kullanmanın sonuçlarından birisi, tasarımcıların her bir komuta daha fazla işlevsellik katabilmeleridir. Programları çalıştırabilmek için gerekli toplam komut miktarını azaltan sadece bu değildir. Bu yüzden fazla bellek kullanımı daha etkili hale gelmiş ve assembly dilinde program yazanların durumunu kolaylaştırmıştır.

Daha sonra tasarımcılar, Assembly dili programcısının amaçladığı komutlarla kendi komut kümesini artırmıştır. Bu tip artırımlar, manevra işlemleri, özel döngü yapıları ve özel adresleme modlarını içermektedir.

CISC Mimarisi (Complex Instruction Set Computer-Karmaşık komut kümeli Bilgisayar)

Devre tasarımcıları, programcıya yakın komut kümesi oluşturmaya başladıktan sonra, sıra mantıksal adımlarla yüksek-düzeyli dillerden oluşan komut kümelerini yapılandırmaya gelmiştir. Bu adım, derleyici yazarların işini kolaylaştırdığı gibi derleyicilere kaynak kodu dizisi başına daha az komut çıkarmayı sağlar.

CISC mimarili işlemciler, tek bir çağrı ile yığın çerçeveleme ve yok etme yordamlarının da dahil olduğu çeşitli tipte komutları yürütür.

CISC'ın doğuşu

CISC tasarım kararları:

- Mikrokod kullanmak
- Zengin komut kümesini oluşturmak
- Yüksek seviyeli komut kümesini oluşturmak

Bu üç karar, bütün bilgisayarları 80'lerin sonuna taşıyan CISC felsefesinin temelidir ve bu gün de hala geçerliliğini korumaktadır. CISC adı, RISC mimarisinin gelişimine kadar bilgisayar tasarımcılarının sözlüğüne girmemişti ve bilgisayarları tasarladıkları tek mimariydi.

Bundan sonra bütün CISC tasarımlarının paylaştığı genel karakteristikler ve bu karakteristiklerin CISC yapısına sahip bir makinenin işlemine etkileri görülecektir.

CISC Tasarımının Özellikleri

80'li yıllara kadar çıkarılan çipler kendine has tasarım yollarını takip ettiler. Bunların çoğu "CISC tasarım kararları" denilen kurallara uydular. Bu çiplerin hepsinin benzer komut kümeleri ve donanım mimarileri vardır. Komut kümeleri, assembly dili programcılarının rahatlığı için tasarlanırlar ve donanım tasarımları oldukça karmaşıktır.

Komut Kümeleri

CISC mimarisinin gelişimine izin veren tasarım sınırlamaları az miktarda yavaş bellek ve ilk makinaların Assembly dilinde programlanması komut kümelerine bazı ortak karakteristikler yüklemiştir:

- Komutların bir kaynak ve bir hedefe sahip olması
- ADD R1,#25 komut satırında, 'R11 hedef,'#25' ise kaynak verisidir.
- Kaydediciden kaydediciye, kaydediciden belleğe ve bellekten kaydediciye komutlara sahip olması
- Diziler yoluyla belleği indislemek için geliştirilmiş pek çok adresleme modları içermesi
- Değişken uzunlukta komutlar içermesi (uzunluk genellikle adresleme moduna göre değişir)
- İcrası için birden fazla çevrim gerektiren komutlar. Eğer bir komut çalıştırılmadan önce ek bir bilgiye ihtiyaç duyarsa, ekstra bilgiyi toplamak fazladan çevrim gerektirecektir. Sonuç olarak bazı CISC komutlarını yürütmek, diğerlerinden daha uzun zaman alacaktır.

Donanım Mimarisi

- Pek çok adresleme modunu desteklemesi amacıyla tek bir komut tarafından karmaşık komut ve deşifre mantığı yürütülür.
- Az miktarda genel amaçlı kaydedici: Bellek üzerine doğrudan işlem yapabildiğinde, fazla genel amaçlı kaydediciye ihtiyaç yoktur. Az miktarda kaydedici, çok miktarda bellek kullanımı demektir.
- Pek çok özel amaçlı kaydedici: Çoğu CISC tasarımı, yığın işaretçisi ve kesme yöneticisi gibi özel amaçlı kaydedicileri kendisi kurar. Bu işlem donanım tasarımını bir dereceye kadar basitleştirir. Komutlar karmaşık olduğundan her parametrenin tutulması ayrı bir önem taşır.
- Bayrak kaydedicisi: Bu kaydedici, son işlemin sonucunun ne olduğunu işlemciye bildirir ve sonraki işlemin buna göre yönlendirilmesini sağlar.

İdeal CISC Mimarisi

CISC işlemciler, bir sonraki komuta başlamadan önce elindeki komutu tamamen icra etmek üzere tasarlanmıştır. Ama gerçekte böyle olmaz, çünkü komutlar çok karmaşıktır ve tek saykılda işlenmez. İş-hattında beklemelere sebep olurlar. Bu sebeple çoğu işlemci bir komutun icrasını pek çok belirli aşamaya ayırır. Bir aşama biter bitme sonuç bir sonraki aşamaya aktarılır.

- Al getir: Bir komut ana bellekten alınıp getirilir.
- Kodunu çöz: Komutun şifresi çözülür. Eğer gerekliyse işlemci bellekten ek bilgi okur.
- Çalıştır: Komut işlenir. Mikroprogramın kontrol kodu işletimi yürütecek donanım çevrimini belirler.
- Tekrar yaz: Sonuçlar belleğe yazılır.

İdeal bir CISC makinasında, her bir komut sadece bir çevrim gerektirir. Gerçekte bu bir anda bir komutu icra eden makine için mümkün olan en yüksek hızdır.

Gerçekçi bir CISC Makinesi

Gerçekte bazı komutlar aşama başına birden fazla saat çevrimi gerektirir. Bununla birlikte, CISC oluşumunun temelindeki düşünce, toplam çevrim sayısını küçük tutmak olduğundan bir CISC tasarımı bu yavaşlamayı uygun görebilir.

Performansı belirlemek için aşağıdaki eşitlik kullanılmaktadır.

İcra süresi=$\sum_{i=1}^{N}$ komut başına çevrim sayısı[i] *çevrim süresi

N=komut sayısı

CISC Mimarisinin Üstünlükleri

- CISC makinalar ilk gelişim sıralarında bilgisayar performansını yükseltmek için mevcut teknolojileri kullandılar.

- Mikroprogramlama assembly dilinin yürütülmesi kadar kolaydır ve sistemdeki kontrol biriminden daha ucuzdur.

- Yeni komutlar ve mikrokod ROM'a eklemenin kolaylığı tasarımcılara CISC makinalarını ilk bilgisayarlar gibi çalıştırabilirler çünkü yeni bilgisayar önceki bilgisayarın komut kümelerini de içerecektir.

- Her bir komut daha yetenekli olmaya başladığından verilen bir görevi yürütmek için daha az komut kullanılır. Bu, nispeten yavaş ana belleğin daha etkili kullanımını sağlar.

- Mikroprogram komut kümeleri, yüksek seviyeli dillerine benzer biçimde yazılabildiğinden derleyici karmaşık olmak zorunda değildir.

CISC Mimarisinin Sakıncaları

- İşlemci ailesinin ilk kuşakları her yeni versiyon tarafından kabullenilmiştir. Böylece komut kodu ve çip donanımı bilgisayarların her kuşağıyla birlikte daha karmaşık hale gelmiştir.
- Mümkün olduğu kadar çok komut, mümkün olan en az zaman kaybıyla belleğe depolanabiliyor ve komutlar neredeyse her uzunlukta olabiliyor. Bunun anlamı farklı komutlar farklı miktarlarda saat çevrimi tutacaktır bu da makinanın performansını düşürecektir.
- Çoğu özel güçlü komutlar geçerliliklerini doğrulamak için yeteri kadar sık sık kullanılmıyor. Tipik bir programda mevcut komutların yaklaşık %20'si kullanılıyor.
- Komutlar genellikle bayrak (durum) kodunu komuta bir yan etki olarak kurar. Bu ise ek saykıllar yani bekleme demektir. Aynı zamanda sonraki komutlar işlem yapmadan önce bayrak bitlerinin mevcut durumunu bilmek durumundadır. Bu da yine ek saykıl demektir. Bayrakları kurmak zaman aldığı gibi, programlar takip eden komutun bayrağın durumunu değiştirmeden önce bayrak bitlerini incelemek zorundadır.

RISC Mimarisi

RISC mimarisi, CISC mimarili işlemcilerin kötü yanlarını gidermek için piyasanın tepkisi ile ona bir alternatif olarak geliştirilmiştir. RISC'ı IBM, Apple ve Motorola gibi firmalar sistematik bir şekilde geliştirmiştir. RISC felsefesinin taraftarları, bilgisayar mimarisinin tam anlamıyla bir elden geçirmeye ihtiyacı olduğunu ve neredeyse bütün geleneksel bilgisayarların mimari bakımından birtakım eksikliklere sahip olduğunu ve eskidiğini düşünüyorlardı. Bilgisayarların gittikçe daha karmaşık hale getirildiği ve hepsinin bir kenara bırakılıp en baştan geri başlamak gerektiği fikrindeydiler.

70'lerin ortalarında yarı iletken teknolojisindeki gelişmeler, ana bellek ve işlemci yongaları arasındaki hız farkını azaltmaya başladı. Bellek hızı artırıldığından ve yüksek seviyeli diller Assembly dilinin yerini aldığından, CISC'ın başlıca üstünlükleri geçersizleşmeye başladı. Bilgisayar tasarımcıları sadece donanımı hızlandırmaktan çok bilgisayar performansını iyileştirmek için başka yollar denemeye başladılar.

İlk RISC Modeli

IBM 70'lerde RISC mimarisini tanımlayan ilk şirket olarak kabul edilir. Aslında bu araştırma temel mimarisel modeller ortaya çıkarmak için Berkeley ve Standford üniversitelerince daha fazla geliştirildi. RISC'ın felsefesi üç temel prensibe dayanır.

Bütün komutlar tek bir çevrimle çalıştırılmalıdır: Performans eşitliğinin gerekli kısmı budur. Gerçekleştirilmesi bazı özelliklerin varolmasına bağlıdır. Komut kodu harici veri yoluna eşit ya da daha küçük sabit bir genişlikte olmalı, ilave edilmek istenen operandlar desteklenmemeli ve komut kodu çözümü gecikmelerini engellemek için dikey ve basit olmalı.

Belleğe sadece "load" ve "store" komutlarıyla erişilmelidir. Bu prensip ilkinin doğal sonucudur. Eğer bir komut direkt olarak belleği kendi amacı doğrultusunda yönlendirirse onu çalıştırmak için birçok saykıl geçer. Komut alınıp getirilir ve bellek gözden geçirilir. RISC işlemcisiyle belleğe yerleşmiş veri bir kaydediciye yüklenir, kaydedici gözden geçirilir ve son olarak kaydedicinin içeriği ana belleğe yazılır. Bu seri en az üç komut alır. Kaydedici tabanlı işlem gerçekleştirmeyle performansı iyi durumda tutmak için çok sayıda genel amaçlı kaydediciye ihtiyaç vardır.

İlk RISC Modeli

Bütün icra birimleri mikrokod kullanmadan donanımdan çalıştırılmalıdır: Mikrokod kullanımı, dizi ve benzeri verileri yüklemek için çok sayıda çevrim demektir. Bu yüzden tek çevrimli icra birimlerinin yürütülmesinde kolay kullanılmaz.

Günümüzün RISC yapısına sahip ticari mikroişlemcilerinde genel olarak iki tarz görülür. *Bunlar Berkeley Modeli* ve *Standford* modelidir. Aralarındaki esas fark kaydedici kümeleri ve bunların kullanımıyla alakalıdır. Her ikisi de veri ve komutların birbirinden ayrıldığı veriye erişimin paralel yapıldığı ve komut ve veri uyuşmazlığını ortadan kaldıran *Harward harici veriyolu mimarisine* sahiptir. Eğer bu iki akış tek bir yoldan yapılmaya kalkışılırsa, herhangi bir veri çağırması komut akışını durdurur ve işlemcinin tek çevrimde işleme hedefine ulaşmasının önüne geçer.

RISC Mimarisinin Özellikleri

RISC mimarisi aynı anda birden fazla komutun birden fazla birimde işlendiği *iş hatlı tekniği* ve *süperskalar* yapılarının kullanımıyla yüksek bir performans sağlamıştır. Bu tasarım tekniği yüksek bellek ve gelişmiş derleme teknolojisi gerektirmektedir. Bu mimari küçültülen komut kümesi ve azaltılan adresleme modları sayısı yanında aşağıdaki özelliklere sahiptir.

- Bir çevrimlik zamanda bir komut işleyebilme
- Aynı uzunluk ve sabit formatta komut kümesine sahip olma
- Ana belleğe sadece load ve store komutlarıyla erişim, operasyonların sadece kaydedici üzerinde yapılması
- Bütün icra birimlerinin mikrokod kullanılmadan donanımsal çalışması
- Yüksek seviyeli dilleri destekleme
- Çok sayıda kaydediciye sahip olması

İş Hattı Tekniği

Bilgisayar donanımının bir anda birden fazla komutu işlemcinin farklı alanlarında işleyebildiği tekniğe iş hattı tekniği denir. Bir komutu ele almaya başlamadan önce bir öncekinin tamamlanmasını beklemez.

CISC tabanlı makinelerde bir komutun işlenmesi 4 adımda yapılmaktadır. Bunlar komutu bellekten alıp getirmek, kodunu çözmek, işlemek ve yeniden belleğe yazmaktır. Bu kademeler RISC tabanlı makinelerde de bulunur. Fakat paralel olarak icra edilirler. Bir kademedeki işlem biter bitmez sonucu diğer kademeye aktarılır ve diğer komut üzerinde çalışmaya başlanılır. Tek iş-hatlı sistemin performansı, bir bölümün tamamlanması için gereken zamana bağlıdır. İş-hattı tekniği kullanmayan tasarımlarda olduğu gibi tüm safhalar için geçen toplam zamana bağlı değildir.

Bu tekniğe sahip RISC tabanlı işlemcilerde, her bir komut her kademede bir saat çevrimi harcar. Böylece işlemci her saat çevrimi başına yeni bir komut kabul edebilir.

İş Hatlı Sistemlerde Performans

İş hattı tekniği kullanan bir işlemci, bellekten veri okuma sırasındaki beklemeler, sınırlı komut kümesi tasarımı veya komutlar arasındaki uyumsuzluk gibi değişik durumlardan dolayı atıl durumda kalabilir.

Bellek Hızı

Bellek hızı sorunları çoğunlukla ön bellek kullanımıyla çözülmüştür. Statik RAM'den oluşan ön bellek *işlemci ile dinamik RAM'e sahip ana bellek arasına yerleştirilmiş* hızlı bir bellek türüdür. İşlemci ana bellekten bir alanı okumak istediğinde, bu alan aynı zamanda ön belleğe kopyalanır. Dolayısıyla, ana belleğe yazmak isterse de veri önce ön belleğe yazılır. Uygun bir zaman bulunduğunda da ana belleğe geçirilir.

Komut Gizliliği

Zayıf tasarlanmış bir komut kümesi, iş-hatlı tekniğe sahip işlemcinin sık sık durmasına sebep olabilir. Genel sorunların bazıları şöyledir:

Yüksek düzeyde şifrelenmiş komutlar: Bu tip komutlar CISC tabanlı makinelerde kullanılır ve çözmek için bir dizi testler uygulanır.
Değişken uzunlukta komutlar: Komutun tümünü getirmek belleğe çok yönlü başvurular gerektirir. Komutların bazıları bir kelime bazıları birden fazladır.
Ana belleğe giriş yapan komutlar: Ana bellek yavaş olabileceğinden işlemci durabilir.
İşlenmesi için çok fazla saat çevrimi gerektiren karmaşık komutlar: Mesela kayan noktalı işlemler
Aynı kaydediciye yazma ve okuma ihtiyacı olan komutlar: Kaydedicinin önceki okuma işleminden hala meşgul olması sebebiyle, kaydedicinin kullanılabilir olmasına kadar işlemcinin durmasına sebep olabilir.
Tek nokta kaynaklarına olan güvenilirlik: Bayrak kaydedicilerindeki durumları eğer bir komut kurarsa ve takip eden komut bu bitler okumaya çalışırsa, ikinci komut birinci komutun yazma işlemi bitene kadar beklemek zorundadır.

Güvenilirlik

RISC programcılarının problemlerinden bir tanesi, zayıf komut kümesinden dolayı işlemcinin yavaşlayabilmesidir.

Her bir komutun sonucunu depolamak için bir miktar zaman harcanmasından ve birkaç komutun aynı zamanda ele alınmasından dolayı sonraki komutların ilk komutların sonuçlarının depolanmasını beklemek zorundadır. Bununla birlikte bir program içindeki komutların basit olarak yeniden düzenlenmesi RISC programlarının bu performans kısıtlamalarını ortadan kaldırabilir.

Süper İş hattı Teknolojisi

Süper iş hatlı sistem, iş hattının her bir kademesini 2 alt devreye ayırır ve saat hızını dâhili olarak ikiye katlar. Her bir kademe saat başına 1 komut icra eder. Ancak dahili saat iki kat hızlı olduğundan iş hatlı saat darbesinin her vuruşunda iki komutu yükleyebilir.

Süperskalar Mimari

Süperskalar makineler aynı şeyi yapmakta yetenekli olan pek çok icra birimini içerir. Bu işlemcinin, birkaç benzer komutun her birini eldeki icra birimlerine dağıtarak işlemesine izin verir.

Mesela iki aritmetik birimi olan süperskalar makine iki çift sayıyı aynı anda (sonuçlarının aynı yere gitmek zorunda olmadığı durumlarda) toplayabilir. Bu makine iki çıkışlı makine olarak adlandırılır. RISC komut kümesi süperskalar mimariye tam uyumludur. Bunu sebebi çip üzerinde daha az yer tutan ve dolayısıyla bir veya daha fazla kez kopyası çıkartılabilen basit işlem birimlerinde her bir komutun icra edilebilmesidir çünkü, komutlar arasındaki bağımlılık önemsenmemektedir. (Eğer iki komut bir kaydedici veya bir bayrak kodları gibi aynı kaynakları isterse, bunlar aynı kaynakları isterse, bunlar aynı anda yürütülemezler).

RISC'de kaydedici sayısı çoktur. Dolayısıyla aynı kaydediciye nadir talep görülür. Bir çok-çıkışlı makine genellikle bir algetir kademesine ve bir kod-çözme kademesine sahiptir. Fakat bu kademeler her saat çevriminin bir parçası içinde işlenir. Böylece bunlar makinenin tüm hızını sınırlamazlar.

RISC Mimarisinin Üstünlükleri

RISC tasarımı olan bir mikroişlemciyi kullanmak, karşılaştırılabilir bir CISC tasarımını kullanamaya göre pek çok avantaj sağlar.

Hız: Azaltılmış komut kümesi, kanal ve süperskalar tasarıma izin verdiğinden RISC işlemciler genellikle karşılaştırılabilir yarı iletken teknolojisi ve aynı saat oranları kullanılan CISC işlemcilerinin performansının 2 veya 4 katı daha yüksek performans gösterirler.

Basit Donanım: RISC işlemcinin komut kümesi çok basit olduğundan çok az çip uzayı kullanırlar. Ekstra fonksiyonlar, bellek kontrol birimleri veya kayan noktalı aritmetik birimleri de aynı çip üzerine yerleştirilir.

Kısa Tasarım Zamanı: RISC işlemciler CISC işlemcilere göre daha basit olduğundan daha çabuk tasarlanabilirler ve diğer teknolojik gelişmelerin avantajlarını CISC tasarımlarına göre daha çabuk kabul edebilirler.

RISC Mimarisinin Sakıncaları

CISC tasarım stratejisinden RISC tasarım stratejisine yapılan geçiş kendi problemlerini de beraberinde getirmiştir. Donanım mühendisleri kodları CISC işlemcisinden RISC işlemcisine aktarırken anahtar işlemleri göz önünde bulundurmak zorundadır.

Kod Özelliği

Bir RISC işlemcisinin performansı işlediği kodun algoritmasına çok bağlıdır. Eğer programcı veya derleyici, komut programlamada zayıf iş çıkarırsa, işlemci atıl durumda kalarak bir parça zaman harcayabilir.

Programlama kuralları karmaşık olabileceğinden çoğu programcılar yüksek düzeyli bir dil kullanırlar ve komut düzenlemeyi derleyiciye bırakırlar.

RISC uygulamasının performansı, derleyici tarafından oluşturulan kodun özelliğine bağlı olduğundan dolayı geliştiriciler işlenmiş kodun özelliğine dayanan derleyicileri dikkatle seçmek zorundadırlar.

Hatalardan Arındırma

Komut planlaması dikkatli yapılmazsa hatalardan arındırmayı zorlaştırabilir. Makine dili komutlarının karıştırılması kodu okumayı zorlaştırır. Bunun için programcı kodlama yaparken dikkatli olmak zorundadır.

Kod Büyümesi

Kod genişlemesi CISC makinesi için derlenen ve RISC makinesi için tekrar derlenen bir programın aradaki göreceli uzunluk farkını işaret eder. Tam genişleme aslında derleyicinin niteliğine ve makinenin komut kümesi yapısına bağlıdır.

CISC tabanlı makinelerin karmaşık işlemlerinin tek bir komut ile yürütülmesinden dolayı kod genişlemesi problem olabilir.

Sistem Tasarımı

RISC makinelerin diğer bir problemi de komutlarını beslemek için çok hızlı bellek sistemleri gerektirmeleridir. CISC tabanlı sistemler genellikle kendi çipleri üzerinde L1 Ön bellek denilen bellekleri taşırlar.

EPIC Mimarisi (Explicitly Parallel Instruction Computing) Belirtilmiş Paralel Komutlarla Hesaplama

Çok uzun kelimeli (VLIW) bilgisayarlar, yazılımın paralelizme ilişkin kesin bilgi sağladığı mimari örneklerdir. Derleyici programdaki paralelliği tanımlar ve hangi işlemlerin bir başkasından bağımsız olduğunu belirterek donanıma bildirir. Bu bilgi, aynı çevrimde hangi işlerin başlatılabileceğiyle ilgili daha fazla denetim olmadan donanımla doğrudan değerlendirilir.

EPIC tarzı mimari, VLIW tekniğinin geliştirilmiş bir modelidir denilebilir. Süperskalar işlemcilerin en iyi yönlerinin bir çoğu EPIC felsefesine adapte edilmiştir. Çok belirgin RISC mimarileri olduğu gibi EPIC yapısı içinde bir komut kümesi mimarisinden fazlası vardır.

EPIC Mimarisinin Avantajları

- Paralel çalıştırma (çevrim başına birden çok komut çalıştırma)
- Tahmin kullanımı
- Spekülasyon kullanımı
- Derleme anında paralelizmi tanıyan derleyiciler
- 128 kayan nokta, 128 tamsayı, 64 tahminli büyük kaydedici kümesi
- Dallanma tahmini ve bellek gecikmesi problemlerine karşı üstün başarı
- Gelişme ve yeni birimlerin eklenmesine verilen doğal yapıdan kaynaklanan destek ve eskiye karşı uyumluluk

EPIC Mimarisinin Avantajları

- Paralel çalıştırma (çevrim başına birden çok komut çalıştırma)
- Tahmin kullanımı
- Spekülasyon kullanımı
- Derleme anında paralelizmi tanıyan derleyiciler
- 128 kayan nokta, 128 tamsayı, 64 tahminli büyük kaydedici kümesi
- Dallanma tahmini ve bellek gecikmesi problemlerine karşı üstün başarı
- Gelişme ve yeni birimlerin eklenmesine verilen doğal yapıdan kaynaklanan destek ve eskiye karşı uyumluluk

X86 Komut Yapısı

Komutlar ve talimatlardan meydana gelen assembly dilinde yazılmış kaynak kodlarının her bir satırında, dört ayrı alan tanımlanabilir. Bunlar etiket alanı, komut alanı, operand alanı ve açıklama alanlarıdır. Etiket, komut ve operand alanları birbirinden bir veya daha fazla boşluk ya da tab ile ayrılırlar. Açıklama alanı ; ile ayrılmalıdır.

Açıklama Alanları

Açıklama programın belli yerlerine ileride dikkat çekmek amacıyla kullanılan bir tanımdır. Programcı için seçimliktir. Eğer isterse programcı bu satırları kullanmayabilir.

```
MOV AX,15H ;15H sayısını AX kaydedicisine yükle
ADD BX,AX ;AX kaydedicisindeki veriyi BX'e yükle
```

Etiket Alanı

Bu alan sembolik bir isimdir ve komut satırının ilk başına konur. Etiketin ilk karakteri sayısal olmamak üzere tüm karakterleri içerebilir. Etiket adı maksimum 32 karakter uzunluğunda kullanılabilir.

BASLA: JMP ANA

- Etiket adları mümkünse kısa ve anlamlı olmalıdır.
- Etiket alanında bir birine benzeyen isimler kullanılmamalıdır.
- Etiket adında birbirine benzeyen karakterler bir arada kullanılmamalıdır. 0 ile O, S ile 5 gibi

Komut Alanı

Mikroişlemciye bir işi tarif eden ve mühendisler tarafından komut cümlesinin kısaltılarak mnemonik hale getirilmiş anlamlı kelimelerin kullanıldığı bu alana komut alanı, aksiyon alanı veya mnemonik alanı denilir. Etiket alanından sonra bir tab veya boşlukla girilen bu alanda bu komutlar bulunur.

CMP
JC
XCHG

Operand Alanı

Operand alanı işlemciye işlenecek verinin nerede oluğunu söyleyen kısımdır. Bu operandlara üzerinde iş yapılan veri denir. Bu alanda komut alanı ile arada bir veya iki karakterlik boşluk bırakılır. İki operand arasına virgül konulur.

```
CMP AX,BX
ADD AX,[BX]
MOV CX,00
NOT AX
JNE BASLA
```

İki operandın bulunduğu alanda ilki hedef operandı, ikincisi kaynak operandı temsil etmektedir.

Operand alanında kullanılan doğrudan verilerin sonunda B,H,D ve O gibi son takılar yazılır. Bunlar o sayıların hangi tabanda olduğunu gösterir. (B:binary, H:Hex,D: Desimal, O:Oktal)

Mikroişlemcilerin Tarihi

1978/1979 yıllarında üretilen ilk 8086/8088'den başlayıp 80286, 80386, 80486, Pentium, Pentium Pro, Pentium MMX, Pentium II, Pentium III ve Pentium IV mikroişlemcilerine uzanan geniş bir ürün yelpazesine sahip olan Intel x86 mikroişlemci ailesi tarihteki en başarılı mikroişlemci ailesi olmuştur.

Bunda çeşitli faktörler rol almıştır, fakat en büyük neden, şüphesiz 1981 yılındaki ilk PC'de IBM firmasının 8088 mikroişlemcisini seçmesi olmuştur. O tarihten itibaren, IBM ve bir çok firma bu işlemcileri PC'lerde kullanmaktadır. PC'lerin dünyada yaygın olarak kullanılması , bu işlemcilerin başarısında en büyük neden olmuştur.

Bir mikroişlemciyi anlatmanın en iyi yolu, işlemcinin veriyolu ve adres yolu genişliğini söylemektir. Veriyolu; sinyalleri taşımak için tasarlanmış bağlantılar dizisidir.
Mikroişlemcinin yolu denildiğinde ilk akla gelen, veri göndermek ve almak için kullanılan tel kümesidir. Birim zamanda ne kadar çok sinyal gönderilebilirse, o kadar çok veri transfer edilebilir ve veriyolu o kadar hızlı olur.

Veriyolundan tamamen farklı olan adres yolu, verinin gönderileceği veya alınacağı bellek konumunu bildiren adresleme bilgisini taşıyan bir dizi telden oluşur. Veriyolunda olduğu gibi adres yolundaki her bir telde yalnızca tek bir bitlik bilgi taşır. Bu tek bit, söz konusu adresin tek bir basamağını oluşturur. Bu nedenle, tel sayısı arttıkça adreslenecek toplam bellek miktarı da artar. Adres yolunun genişliği, bir yonganın adresleyebileceği maksimum RAM miktarını belirler.

8086 da 16-bit olan veriyolu Pentium işlemcisiyle x86 ailesinin veriyolu uzunluğu 64-bite çıkarılmıştır. Intel, Pentium ile RISC mimarisi tasarım kavramlarından olan Superscalar mimariyi kullanmaya başladı. Pentium da aynı anda bir saatte, iki tane iş-hatlı tam sayı birimi iki komutu ve bir tane iş-hatlı FPU birimi de bir tane FPU komutu yürütebilmektedir.Bu işlemcide ayrıca yürütme performansının önemli olarak etkileyen tümleşik devre üzerinde birinci seviye (L1) ayrı 8 KB kod ve 8 KB veri önbelleği bulunur.

Pentium Pro, 8086/8088, 80286, 80386, 80486 ve Pentium işlemcilerinden sonra gelen 6'ıncı nesil olduğu için, ilk çıkması sırasında P6 kod adıyla anılmış ve önemli mimari ekler sunmuştur. P6 mimarisi dinamik yürütme teknolojisi olarak belirtilen ve çoklu dallanma tahmini, veri akışı analizi ve tahmini yürütme olarak üç temel fonksiyonlu mimari yapıyı içermektedir. Pentium Pro'ya 4 yeni adres hattı daha eklenerek adres yolu 36-bit yapıldı. Intel firması ilk kez 256 K, 512 K veya 1 MB olabilen L2 önbelleğini Pentium Pro işlemcisinin üzerine yerleştirdi.

Intel firması bir PC'ye DSP özelliği kazandırmak için MMX olarak adlandırılan bir teknolojiyi, Pentium işlemcilerine 1997'den itibaren koymaya başladı. MMX teknolojisi Multimedya işlemleri için 57 tane yeni komut sunmaktadır.

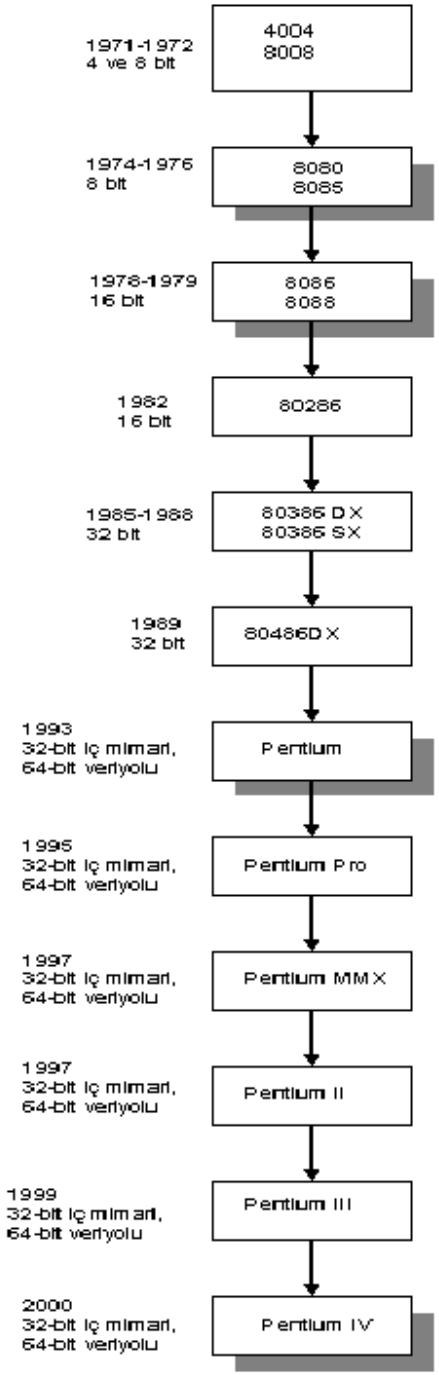
Intel Pentium II işlemcisi, Pentium Pro ve MMX teknolojilerinin birleşimi ile üretildi. Bu işlemcide bulunan 36 KB'lık L1 önbelleği yoğun olarak kullanılan veriye hızlı erişim sağlar. Ayrıca tümleşik devre üzerinde 512 KB'dan başlayan L2 önbelleği bulunur.

Pentium III mikroişlemcisi 1999 yılının başında Intel tarafından piyasaya sunulmuştur. Pentium III ile gelen önemli bir yenilik SIMD olarak adlandırılan bir yapıdır. Bu mimari yapı ile, ileri görüntü işleme, 3D, ses, video ve ses tanıma gibi uygulamalarda kullanılabilecek 70 tane yeni komut eklenmiştir. Pentium III ayrıca P6 mikromimarisini çok işlemli sistem yolu ve Intel MMX teknolojisini içerir.

Pentium IV, Intel'in 1995'ten beri ilk tamamen yenilenmiş x86 mikroişlemcisidir. Intel 1995'ten beri MMX, SSE, çip üzeri L2 kaşe bellek ve daha hızlı sistem veriyolları gibi pek çok gelişmeyi işlemcilerine ekledi. Fakat Pentium IV, gelecekteki pek çok çip'in temelini oluşturan gerçek bir yeni nesil tasarımdır. Pentium IV'ün sistem veriyolları 400 MHz'de çalışabilir. Bu durumda Pentium IV en hızlı Pentium III'ten yüzde 33 daha hızlı çekirdek frekansına ve 2 kat daha hızlı bir veriyoluna sahiptir.

Günümüzde x86 pazarı büyük bir endüstri olmuş ve her yıl milyonlarca işlemci satılmaktadır. X86 işlemcilerinin büyük popularitesi Intel'den başka firmaları da bu pazara sokmuş ve x86 uyumlu işlemciler üreten bir çok firma ortaya çıkmıştır. Günümüzde Intel'in yanında AMD, Cyrix ve Centaur hala aktif olarak yarışmaktadır.

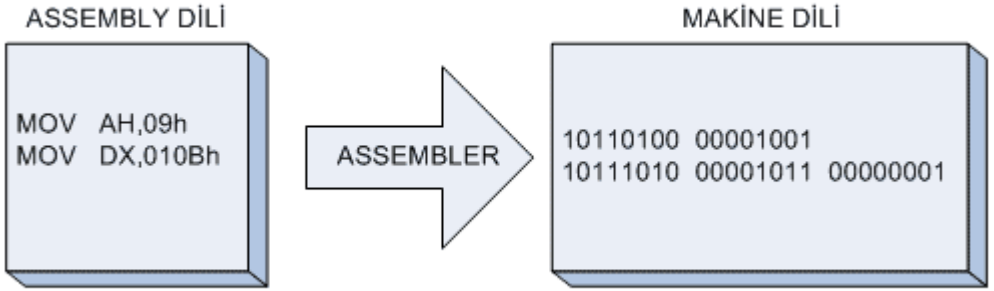
Intel Mikroişlemci gelişimi



Assembly Dili

Assembly programlama dili, kullanılan bilgisayar sisteminin yapısına ve işletim sistemi gibi platformlara sıkı-sıkıya bağımlı bir dildir. Assembly programlama dili düşük seviyeli bir dil olup C, C++, Pascal, C# gibi yüksek seviyeli programlama dillerine göre anlaşılması biraz daha zordur. Assembly dili ile program yazarken kullanılan bilgisayarın donanımsal özelliklerinin bilinmesi gerekir. Yazılan program kullanılan mikroişlemcinin yapısına bağlıdır. Assembly dili ile program yazarken programcı doğrudan bilgisayarın işlemcisi ve hafızası ile uğraşır. Anabellekteki (RAM'deki) ve işlemci kaydedicilerindeki değerleri doğrudan değiştirebilme imkanı vardır.

Mikroişlemci sadece ikili sayı sisteminde yazılan komut kodlarını, başka bir ifade ile makine dilinden anlar. Assembly dilinde yazılan programları makine diline çevirmek için Assembler adı verilen çevirici(derleyici) programlar kullanılır. Aşağıda verilen şekilde Assembly dili, Makine dili ve Assembler blok olarak görülmektedir.



Bilgisayarımızda çalıştırılan tüm programlar önce bilgisayarımızın RAM belleğ'ine yüklenir. Daha sonra RAM bellekten sırası ile mikroişlemci tarafından okunarak çalıştırılır. RAM'e yüklenen veri programın makine dili karşılığından başka bir şey değildir. Yani 0 ve 1 kümeleridir.

Makine dilinde program yazmak oldukça zordur. Buna karşılık makine dili ile birebir karşılığı olan ve komutları kısaltılmış kelimelerden (mnemonik) oluşan Assembly dilinden yararlanılır.

Assembly dilinde program yazmak makine dilinde program yazmaya göre daha hızlı ve daha kolay yapılabilir. Ayrıca yazılan programların bellekte kapladıkları yerde aynıdır. Başka bir ifade ile bellek kullanımları aynıdır.

Yüksek seviyeli dillerle karşılaştırıldığında assembly dilinde yazılan programlar daha hızlıdır ve bellekte daha az yer kaplar. Buna karşılık program yazmak yüksek seviyeli dillerde daha kolaydır.

Assembly programlama dili günümüzde daha çok sistem programcıları tarafından diğer programlama dilleri içerisinde kullanılmaktadır.

Assembly dilinin dezavantajları

- Assembly dilinde program yazmak için mikroişlemci içyapısı bilinmesi gerekir.
- Assembly dili mikroişlemci tipine göre değişir. Bir mikroişlemci için yazılan bir program başka bir mikroişlemcide çalışmayabilir. Program taşınabilir platformdan bağımsız değildir.
- Assembly dilinde program yazmak yüksek seviyeli dillere göre daha zor ve zaman alıcıdır.

Assembly dilinin avantajları

- Bigisayar donanımı üzerinde daha iyi bir denetim sağlar. İşlemcinizin gücünü en iyi şekilde ortaya koyabilecek tek programlama dilidir.
- Küçük boyutlu bellekte az yer kaplayan programlar yazılabilir. virüslerin yazımında kullanılırlar.
- Yazılan programlar daha hızlı çalışır. Çok hızlı çalıştıkları için işletim sistemlerinde kernel ve donanım sürücülerinin programlanmasında, hız gerektiren kritik uygulamalarda kullanılmaktadır.
- Herhangi bir programlama dili altında, o dilin kodları arasında kullanılabilir.
- İyi öğrenildiğinde diğer dillerde karşılaşılan büyük problemlerin assembly ile basit çözümleri olduğu görülür.

Assembly dilinde program yazma

Assembly dilinde program yazmak için Windows altında yer alan note pad, word pad gibi herhangi bir text editör kullanılabilir. Text editör yardımı ile Assembly dilinde program yazılır. Yazılan program TASM veya MASM assembler çevirici programları yardımı ile .obj uzantılı olarak makine diline çevrilir. Bu halde elde edilen program işletim sisteminin anladığı bir formatta değildir. TLINK bağlayıcı programı kullanılarak .exe veya .com uzantılı hale dönüştürülür. Bu haldeki program işletim sistemi üzerinde ismi yazılarak DOS ortamında çalıştırılabilir.

Bir Assembly dilinde yazılan programda temel olarak şu bölümler bulunur:

- Yorumlar
- Label (Etiketler)
- Talimatlar
- Komutlar

Yorumlar / Açıklamalar

Açıklamalar program satırlarının başına noktalı virgül konularak yapılır. Açıklama satırları assembler tarafından dikkate alınmaz. Program içinde daha detaylı bilgi vermek, kullanılan komutları izah etmek için kullanılır.

örnek:

```
; MOV    ES,AX          bu komut dikkat alınmaz
; AL ye SAYI1 değerini at
```

Etiketler

Etiketler program içinde kullanılan özel kelimelerdir. Sonuna ":" konularak kelimenin etiket olduğu anlaşılır. Etiketlerden program akışını belirli bir noktaya yönlendirmek istediğimizde yararlanırız.

Örnek:

```
Son:
Basla:    JMP ANA
```

Burada Son, Basla kelimeleri etikettir.

Talimatlar

Veri tanımlama talimatları

Veri tanımlama talimatları DB, DW, DD,DF, DQ, DT ve DUP dur.
DB (Define Byte): 1 Byte'lık veri tanımlanır.
DW (Define Word):2 Byte'lık veri tanımlanır.
DD (Define double word):: 4 Byte'lık veri tanımlanır.
DF (Define Far Word): 6 Byte'lık veri tanımlanır.
DQ (Define Quad Word): 8 Byte'lık veri tanımlanır.
DT (Define Ten Byte): 10 Byte'lık veri tanımlanır.
DUP: Duplicate

SAYI 3 DUP(0); Bellekten SAYI değişkeni için 3 byte'lık yer ayır, içini 0 ile doldur.
SAYI DW 10 DUP(5) Bellekten SAYI değişkeni için 10x2 byte'lık yer ayır, içlerini 5 ile doldur.

String verileri tanımlama

YAZI DB ‘KARABUK’
YAZI DB ‘K’,’A’,’R’,’A’,’B’,’U’,’K’

Dizi Tanımlama

DIZI DB 2, 4, 0, -5, 7
DIZI DB 12, 0FH, 01001001B
Sayıların sonunda B olması verinin ikilik sistemde olduğunu, H olması verinin hexadesimal olduğunu gösterir. Bir şey yazılmamışsa veri onluk sistemde yazılmış anlamına gelir.

Segment Talimatları

Segment talimatları bir segmentin başlangıcını tanımlamada kullanılır. Segmente hehangi bir isim verebilirsiniz.

SegmentAdı SEGMENT ParametreListesi

•

• .

Ver tanımları ve
Komutlar

•

•

SegmentAdı ENDS

Parametre listesi sırası ile ALIGN, COMBINE, CLASS parametrelerini alabilir. Bu parametrelerin kullanımı seçimliktir. Bu parametreler aşağıda verilen segment tanımlamasında olduğu gibi kullanılmasa da olur.

VeriSegment SEGMENT
•
• .
Ver tanımları ve
Komutlar
•
•
VeriSegment ENDS

Parametre listesi verildiğinde aşağıdaki gibi bir tanımlama yapılabilir. Bu tanımlamada para ALIGN parametresini, public COMBINE parametresini ve ‘Data’ CLASS parametresini ifade eder.

VeriSegment SEGMENT para public ‘Data”

-
- .

Ver tanımları ve
Komutlar

-
-

VeriSegment ENDS

Para: Bu alan segmentin paragraf başlarında (sonu 0 ile biten adreslerden) başlayarak yerleşeceğini ifade eder. Bu parametre belirtilmediğinde varsayılan değer para olarak belirlenir.

Combine Alanı: Bu alan assembler tarafından aynı adla meydana getirilen amaç programların segmentlerinin birbirleriyle nasıl bir bağ kuracağını ifade eder. Common, public, stack, memory ve at değerlerini alabilir.

Class Alanı: Segmentin hangi amaçla kullanılacağını ifade eder. Stack, Code ya da Data olabilir.

Örnekler :

```
KodSeg     SEGMENT  para public “Code”
DataSeg    SEGMENT  para public “Data”
StakSeg    SEGMENT  para Stack “Stack”
```

PROC talimatı

Assembly dilinde procedure(alt program) tanımlamak için kullanılır. Altprogram aşağıda verildiği gibi tanımlanır. Far veya Near parametresi Bu alt programın aynı veya farklı segmentlerden çağrılıp çağrılamayacağını belirtir. Far olursa farklı segmentlerden, Near olursa aynı segmentten çağrılabilr. CALL AltprogAdi şeklinde çağrılarak altprogramlar kullanılır.

AltprogAdi PROC Far/Near

- .

  Komutlar

-

AltprogAdi ENDP

Veri aktarım komutları

Komut kümesinde kullanılan kısaltmalar

acc	Akümülatör (EAX/AX/AL yazmaçlarından herhangi biri)
reg	8/16/32 bitlik herhangi bir yazmaç
regb	8 bitlik bir yazmaç
regw	16 bitlik bir yazmaç
reg32	32 bitlik bir yazmaç
sreg	herhangi bir kesim yazmacı
mem	herhangi bir bellek adresi
idata	8/16/32 bitlik herhangi bir değer (3,0Ah,012EFh,'A' gibi)
[..]	İşaret grubu yazmaç veya göreli konum değerini gösteren sayı ile erişilebilen bellek içeriği
disp8/disp16	8 bit(-128…0…127)/16 bit(-32768…0…32767)ile ifade edilebilecek büyüklükteki bir sayı kadar ifade
dest/src	Varış işleneni /Kaynak işleneni
opr/opr #	İşlenen/işlenen 1/işlenen 2/işlenen 3
italik yazı	değişken isimleri
0/1/x/?/--	İşlem sonucunda bayrak değeri clear(0),set(1), işleme göre değişmemiş (x), belirsiz (?), değişmemiş olabilir(-)
16p	16 bit korumalı kip

MOV (mov data)

MOV reg,idata
MOV mem,idata
MOV reg,reg
MOV mem,reg
MOV sreg,reg
MOV reg,sreg
MOV mem,sreg

İki işlenen ile kullanılan MOV komutu ikinci işlenendeki veriyi ilk işlenene aktarır. MOV komutunun bazı kısıtlamaları vardır. MOV komutunun her iki işleneni mem ve sreg olamayacağı gibi ilk işlenen sreg ise ikincisi idata olamaz.
MOV dest,src
Şeklinde ifade edilen komut src'nin değerini dest'e yerleştirir. Bayraklar üzerinde herhangi bir değişiklik yapmaz. MOV komutunda her iki işlenenin de aynı tipte olması gerekir. Aksi halde derleyici tip uyuşmazlığı hatası verir.

Örnek:

MOV *Mydata*,AX ; *Mydata* isimli bellek alanına AX yazmacında bulunan değer yerleştirilmektedir. *Mydata* isimli bellek alanı yazmaç ile aynı tipte tanımlanmış olmalıdır.

MOV AL,*Result* ; *Result* isimli bellek alanından, AL yazmacına veri aktarılmaktadır. *Result* isimli bellek alanının byte olarak tanımlanması gerekmektedir.

MOV Mydata,12 ; Mydata isimli bellek alanına 12 değeri yerleştirilmektedir. Kaynak işlenen olan 12 sayısı hem byte hem word bir büyük olarak ifade edilebileceği dikkate alındığında Mydata isimli değişkenin byte veya word olarak tanımlanmış olmasının işleyişe bir etkisi olmayacak, her iki durumda da atama işlemi yapılacaktır.

Örnek:

MOV WORD PTR[1001],120A; Bellek alanına adres ile erişilmesi durumunda tip uyuşmazlığı söz konusu olabilir. Programcı bellek üzerinde işlemleri hangi boyutta (byte/word) yapmak istediğine bağlı olarak gerekli tip tanımlamaları yapmalıdır. Bunu için doğru tipte adresleme modu kullanılmalıdır.

Örnek Program

```
#make_COM#

; COM file is loaded at CS:0100h
ORG 100h

X DW 35
mov ax,15                   ;reg,idata
MOV [1001H],5               ;mem,idata
mov bx,ax                   ;reg,reg
mov cx,[1001H]              ;reg,mem
mov [1002H],bx              ;mem,reg
mov SI,ax                   ;sreg,reg
MOV SI,[1002H]              ;sreg,mem
MOV cx,SI                   ;reg,sreg
MOV [1003h],SI              ;mem,reg

HLT
```

MOVSX (mov with sign extension)

MOVSX reg,reg
MOVSX reg,mem

Bu komut 8 bitlik değeri işareti ile birlikte 16 veya 32 bitlik alana, 16 bitlik değeri işareti ile birlikte 32 bitlik alana yerleştirir. İşaret, en anlamlı bitin yüksek anlamlı byte veya word ile tekrar edilmesi ile aktarılmaktadır. Genel olarak ilk işlenen ikinci işlenenden bir üst boyutta olmalıdır. Böylece ikinci işlenende bulunan değer daha üst büyüklüğe işaretli sayı olarak aktarılmış olur.

Örnek:

MOVSX EAX,AL ; AL yazmacındaki 8 bitlik değeri 32 bite uzatarak EAX yazmacına aktarır. Bu işlem sonucunda AL yazmacının işaret biti (7.bit), 8-31. bitler boyunca tekrarlanır. AL=67H ise bu işlemden sonra EAX=00000067H olacaktır.

MOVSX EDI,WORD PTR[ESI] ; ESI yazmacının belirlediği bellek gözündeki 16 bit değeri 32 bite uzatarak EDI yazmacına aktarır.15. bit ile ifade edilen işaret 16-31. bitlerde tekrarlanacaktır.

MOVSX CX,DL; DL yazmacındaki 8 bitlik değeri 16 bite uzatarak CX yazmacına yerleştirir. DL=80H ise sonuçta CX=0FF80H değerini alacaktır.

MOVZX (Move with zero extension)

MOVZX reg,reg
MOVZX reg,mem

Bu komut 8 bitlik değeri 16 veya 32 bitlik alana 16 bitlik değeri 32 bitlik alana yerleştirir. İşaret uzantısı ise en anlamlı bitlerin 0 ile doldurulması ile elde edilir. Genel olarak ilk işlenen ikinci işlenenden bir üst boyutta olmak zorundadır. Böylece ikinci işlenende bulunan değer üst büyüklüğe işaretsiz sayı olarak aktarılmış olur.

Örnek

MOVZX EAX,AL; AL yazmacındaki 8 bitlik değeri 32 bitlik değere uzatarak EAX yazmacına yerleştirir. 8-31. Bitler 0'dır. AL=80H ise sonuçta EAX=00000080H olacaktır.

MOVZX EDI,WORD PTR[ESI] ; ESI ile belirlenen bellek gözündeki değeri 32 bite uzatarak EDI yazmacına yerleştirir. 16-31. Bitler 0'dır. WORD PTR ifadesi ile bellekten 1 WORD alınacağı belirlenmektedir.

MOVZX CX,DL; DL yazmacının 8 bitlik değeri 16 bite uzatılarak CX yazmacına yerleştirilir. 8-15. Bitler 0'dır.

MOVZX BX, BL; BL yazmacındaki değeri işaret değeri 0 olacak şekilde BX yazmacına yayar. Ancak MOV BH,0 işlemi buna nazaran daha kısa sürede tamamlanacaktır.

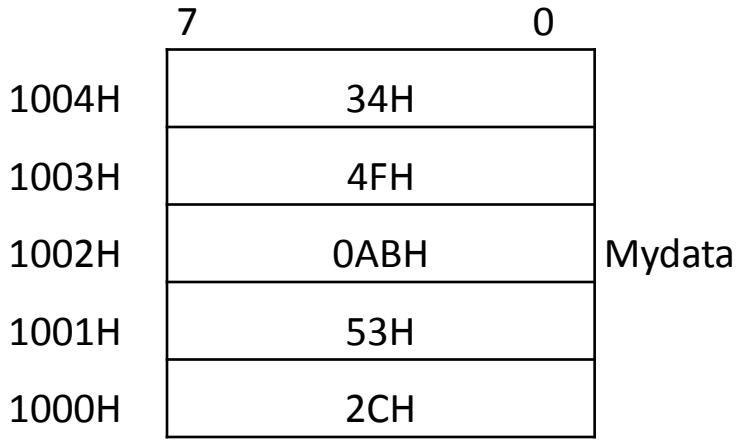
LEA (load effective adres)

LEA regw,mem

İşlem sonucunda regw de oluşan değer mem ile tanımlı bellek alanının, kesiminin başından itibaren kaç byte olduğunu (göreli konum) belirlemektedir. Elde edilecek adres 16 bit uzunluğunda olacaktır. Bu nedenle işlem sonucunun oluşacağı kısım mutlaka word tipinde bir yazmaç olmalıdır. Aynı işlem OFFSET komutu ile de gerçekleştirilebilmektedir.

Örnek:

LEA SI,Mydata ; Mydata isimli bellek alanının, tanımlı olduğu kesimin başından itibaren kaç byte uzakta olduğunu gösteren değer SI yazmacında oluşacaktır. Aşağıdaki bellek haritasında verilen değerlere göre SI=1002H olacaktır.



Örnek Kod:

```
#make_COM#

; COM file is loaded at CS:0100h
ORG 100h

X DW 35
LEA BX,X
```

LDS(load data segment register)

LDS regw,mem

Regw ile belirlenen yazmaç ile DS yazmacına, mem ile belirlenen bellek bölgesinde bulunan değerler yüklenir.
Yapılan işlem: regw=[mem]
DS=[mem+2]
Bu komut bir seferde DS:SI ikilisinin değerlerinin bellek alanından alınmasını sağlamaktadır. Genel olarak SI yazmacı ile kullanılmasının arkasında, bellek erişimlerinde DS ve SI yazmaçlarının birlikte kullanılması yatar. Benzer şekilde DI ve BX yazmaçları da bu komut ile birlikte kullanılabilir. Mem değeri doğrudan bir bellek adresi olarak verilebildiği gibi bir değişken ile de verilebilir. Bu durumda kullanılacak olan değişken 32 bit (doubleword) olacak şekilde tanımlanmış olması gereklidir.

Örnek:

LDS SI,[0000]
LDS SI,Mydata

Aşağıda belirtilen bellek haritası uyarıca SI=5341H ve DS=4553H değerlerini almış olacaktır.Mydata değişkeni kullanılarak işlem yapılmak istendiğinde değişkenin double word olarak tanımlanmış olması gerekir.

LES(load extra segment register)

LES regw,mem

LDS komutu ile aynı özellikleri göstermektedir. Ancak işlem diğerinden farklı olarak ES yazmacı kullanılarak gerçekleştirilir.

Yapılan işlem:

Regw=[mem]
ES=[mem+2]

Bu komut bir seferde ES:DI ikilisinin değerlerinin bellek alanından alınmasını sağlamaktadır. Genel olarak DI yazmacı kullanılmasının arkasında bellek erişimlerinde ES ve DI yazmaçlarının birlikte kullanılması yatmaktadır. Benzer şekilde SI ve BX yazmaçları da bu komut ile birlikte kullanılabilir. Mem değeri doğrudan bir bellek adresi olarak verilebildiği gibi bir değişken ile de verilebilir. Bu durumda kullanılacak olan değişkenin 32 bit olacak şekilde tanımlanmış olması gerekmektedir.

Örnek:

LES DI,[0001]

LDSSI,Mydata

Yukarıdaki bellek haritası uyarınca yapılan işlem sonucunda DI=5351H ve ES=4D45H değerlerini almış olacaktır. Mydata kullanılarak işlem yapılmak istendiğinde değişkenin 32 bit (double word) olacak şekilde tanımlanmış olması gerekmektedir.

XCHG(Exchange)

XCHG reg,reg
XCHG reg,mem
XCHG mem,reg

XCHG işlenenlerin değerlerinin değiş tokuş edilmesini sağlayan komuttur. Aynı sonucu, biraz daha uzun sürede ek bir yazmaç veya bellek alanı kullanarak daha fazla kod üreterek MOV komutları ile elde etmek te mümkündür. Ancak assembly dili mevcut kaynakların sınırlı olduğu durumlarda çözümler üretebilen bir dil olup bu gibi durumlarda XCHG komutu tercih edilmelidir.

Örnek:

XCHG AX,BX; AX yazmacının sahip olduğu değer ile BX yazmacının sahip olduğu değer değiştirilir.
XCHG AX,Mydata[SI]; Burada ise AX yazmacının değeri Mydata isimli bellek adresinin SI'ıncı adresinden başlayan word yer değiştirmektedir.MYdata word olarak tanımlanmalıdır.

Örnek Kod:

```
#make_COM#

; COM file is loaded at CS:0100h
ORG 100h

X DW 35
mov ax,10
xchg ax,x
hlt
```

XLAT/XLATB (Translate byte)

XLAT
XLATB
Doğrudan işleneni olmayan bir komuttur. AL yazmacını, başlangıcı DS:BX yazmaçları ile belirlenen adresteki bir tablo içerisinde indis olarak kullanmayı sağlar.

Yapılan işlem:
AL=DS:[BX+AL] veya AL=DS:[EBX+AL] dir.

Örnek:

```
LEA BX,Ascii2ebcdic
MOV AL,'A'
XLAT
```

Ascii2ebcdic isimli bellek alanında ASCII ve EBCDIC kodlama sistemi arasındaki dönüşüm yapmayı sağlayan 256 byte uzunluğunda bir dizi olduğunu, dizinin indislerinin ASCII kodlar, içeriğinin ise ASCII kodlara karşılık gelen EBCDIC değer olduğunu düşünelim. AL yazmacına 'A' yerleştirilip, XLAT komutu işlendiğinde diziden 'A' harfinin EBCDIC karşılığı alınarak AL yazmacına konacaktır.

ÖRNEK Kod:

```
#make_COM#

; COM file is loaded at CS:0100h
ORG 100h

dizi db 0,8,2,3,1,9,12,7,8,0,10,11,12,13,14
lea bx,dizi
mov al,5
xlat
hlt
```

8086 Mimarisi (Tekrar)
Adresleme Modları

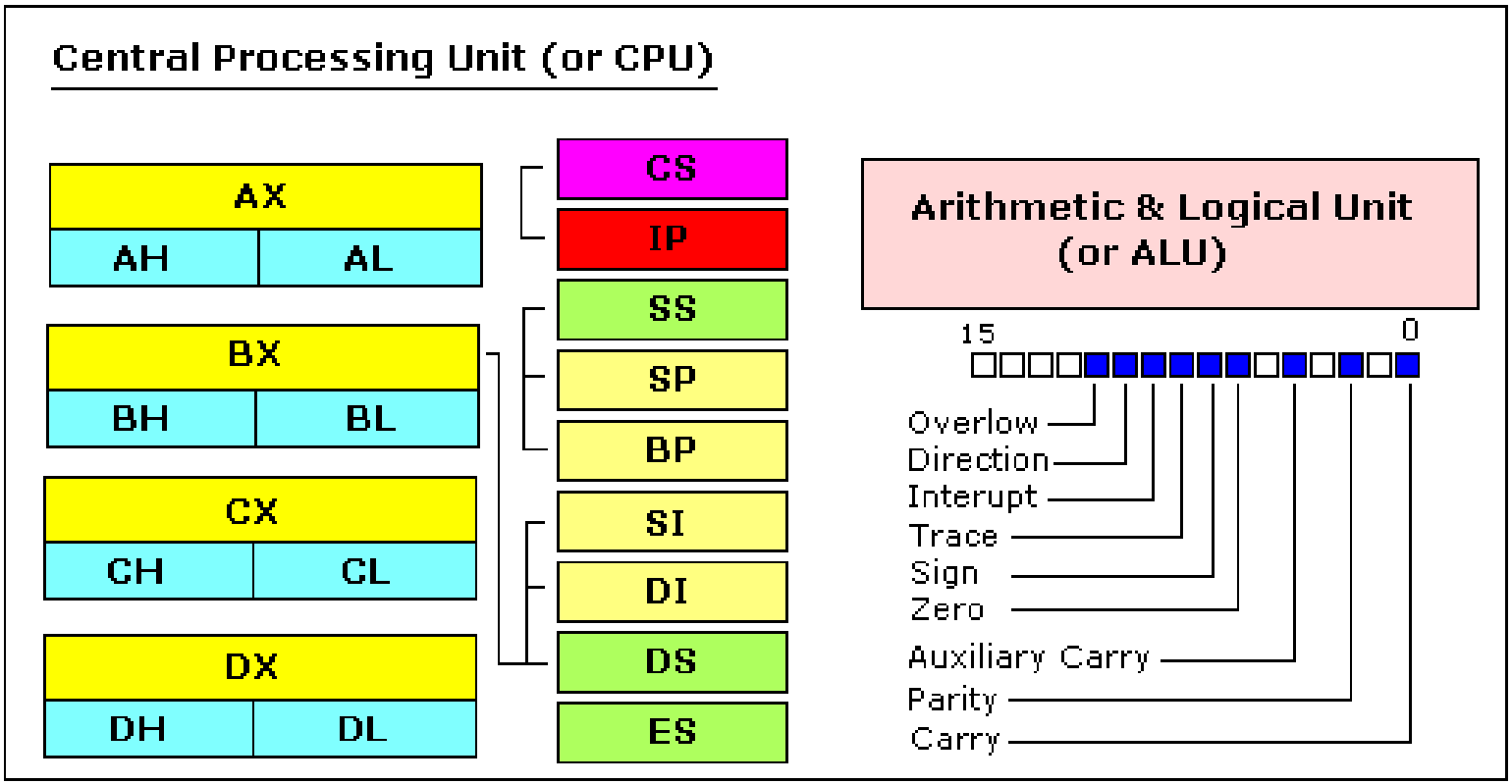
8086 Mimarisi

- 8086'da bulunan tüm iç register'lar ve veri yolları 16 bitlik genişliktedir.

Adresleme

- Veri yolu 16-bit genişliğindedir. Adres yolu ise 20-bit genişliğindedir
- 8086, 1MB'lık hafıza bloğunu adresleyebilir ($1M = 2^{20}$)
- Ancak, en fazla adreslenebilir hafıza uzayı 64 KB büyüklüğündedir. Çünkü tüm dahili register'lar 16-bit büyüklüğündedir. 64 KB'lık sınırların dışında programlama yapabilmek için extra operasyonlar kullanmak gerekir

8086 Bileşenleri



Register’lar

- Register’lar, CPU içerisinde bulunduklarından dolayı, hafıza bloğuna göre oldukça hızlıdırlar.
- Hafıza bloğuna erişim için sistem veri yollarının kullanılması gereklidir.
- Register’daki verilerin ulaşılması için çok çok küçük bir zaman dilimi yeterli olur.
- Bu sebeple, değişkenlerin, register’larda tutulmasına çalışılmalıdır.
- Register grupları genellikle oldukça kısıtlıdır ve çoğu register’ın önceden tanımlanmış görevleri bulunur. Bu nedende, kullanımları çok sınırlıdır. Ancak, yine de hesaplamalar için geçici hafıza birimi olarak kullanılmak için en ideal birimlerdir.

Register Tipleri

1. Genel Amaçlı Register'lar
2. İndis Register'ları
3. Özel Amaçlı Register'lar

1. Genel Amaçlı Register'lar

- 8086 CPU'da, 8 genel amaçlı register bulunur. Her register'ın ayrı bir ismi bulunur:
  - **AX** - accumulator register – akümülatör (**AH / AL**).
  - **BX** - the base address register – adres başlangıcı (**BH / BL**).
  - **CX** - the count register – sayma (**CH / CL**).
  - **DX** - the data register – veri (**DH / DL**).
  - **SI** - source index register – kaynak indisi.
  - **DI** - destination index register – hedef indisi.
  - **BP** - base pointer – temel gösterici.
  - **SP** - stack pointer – yığıt gösterici.

Genel Amaçlı Register'lar (devam)

- Bu register'lar, isminin belirttiği amaçlar için kullanılmak zorunda değildir. Programcı, genel amaçlı register'ları istediği gibi kullanabilir.
- Register'ların ana amacı, bir değişkeni tutmaktır.
- Yukarıdaki register'ların tamamı 16-bitliktir.
- 4 genel amaçlı register (AX, BX, CX, DX), iki 8-bitlik register olarak kullanılabilir.
  - Örneğin eğer **AX=3A39*h* ise**, bu durumda **AH=3A*h*** ve **AL=39*h*** olur.
  - 8-bitlik register'ları değiştirdiğiniz zaman, 16-bitlik register'lar da değişmiş olur.

2. Segment Register'ları

- **CS** – (Code Segment) Mevcut programın bulunduğu bölümü işaretler.
- **DS** – (Data Segment) Genellikle programda bulunan değişkenlerin bulunduğu bölümü işaretler.
- **ES** – (Extra Segment) Bu register'ın kullanımı, kullanıcıya bırakılmıştır.
- **SS** – (Stack Segment) yığının bulunduğu bölümü işaretler.

Segment Register'ları (devam)

- Segment register'larında herhangi bir veriyi depolamak mümkündür. Ancak bu, güzel bir fikir değildir.
- Segment register'larının özel amaçları vardır. Hafızada ulaşılabilir bazı bölümleri işaretler.

Segment Register'ları (devam)

- Segment register'ları, genel amaçlı register'ları ile birlikte çalışarak hafızada herhangi bir bölgeyi işaretleyebilir. Örneğin, fiziksel adres **12345h** (heksadesimal) işaretlenmesi isteniyor ise, **DS = 1230h** ve **SI = 0045h** olmalıdır.
- CPU, segment register'ı 10h ile çarpar ve genel amaçlı register'da bulunan değeri de ilave eder (1230h×10h + 45h = 12345h).

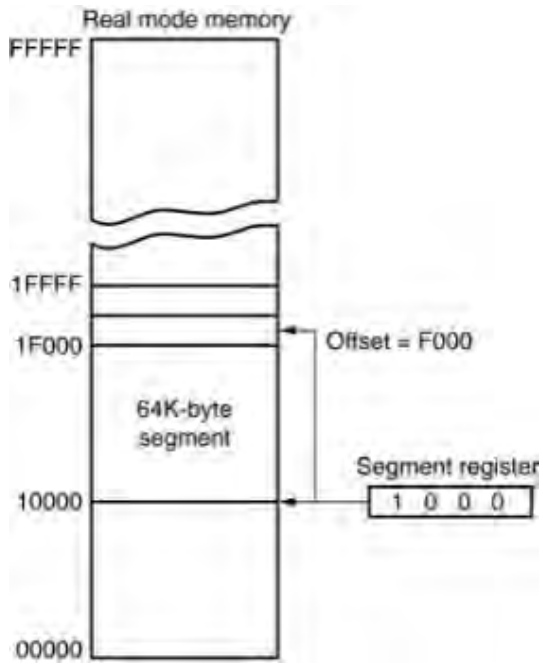
Segment Register'ları (devam)

- 2 register tarafından oluşturulmuş olan adrese, **effective address** (efektif adres) ismi verilir.
- **BX, SI** ve **DI register'ları**, **DS** ile birlikte çalışır; **BP** ve **SP** register'ları ise **SS** ile birlikte çalışır.
- Diğer genel amaçlı register'lar, efektif adres oluşturmak için kullanılmazlar. Ayrıca, BX efektif adres oluşturulmasında kullanılırken, BH ve BL kullanılmaz.

Segment ve Offset

- Tüm hafıza adresleri segment adresine offset adresi ilave edilmesi ile bulunur.
  - **segment adresi:** Herhangi bir 64 KB'lık hafıza bölümünün başlangıcını gösterir.
  - **offset adresi:** 64 KB'lık hafıza bölümünde herhangi bir satırı belirtir.

Segment ve Offset (devam)



- Segment register'ı 1000h değerine sahip ise, 10000h ile 1FFFFh aralığında bir hafıza adresine karşılık gelir.
  - 64KB'lık bir aralıktır
- Offset değeri F000h ise 1F000h adresindeki hafıza satırına karşılık gelir.

Segment ve Offset (devam)

- Başlangıç adresi belli ise, bitiş adresi FFFFh ilave edilerek bulunur.
  - Çünkü segment register'ı 64 KB'lık bir bölümü işaretler.
- Offset adresi, segment adresine ilave edilir.
- Segment ve offset adresleri 1000:2000 biçiminde de yazılabilir.
  - Bu durumda segment adresi 1000H ve offset'te 2000H'dir.

Öntanımlı Segment ve Offset Register'ları

- CS:IP
  - CS, kod bölümünün başlangıcına işaret eder. IP, kod bölümü içerisinde bir sonraki komutun bulunduğu hafıza adresine işaret eder.
  - Eğer CS=1400H ve IP=1200H, mikroişlemci, bir sonraki komutu 14000H + 1200H = 15200H adresinden okur.

Öntanımlı Segment ve Offset Register'ları (devam)

- SS:SP veya SS:BP
  - Yığın bölümünü kullanan komutlar kullanır.
- DS register'ı, BX, DI, SI, 8-bit'lik veya 16-bit'lik sayı ile birlikte
- ES:DI (string komutları için)

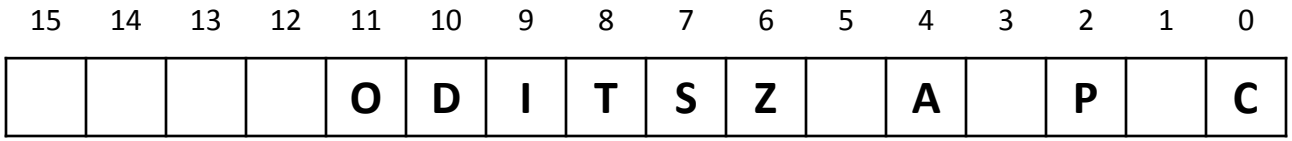
3. Özel Amaçlı Register'lar

- **IP** –instruction pointer – komut işaretleyicisi.
  - **IP** register'ı CS ile birlikte, halihazırdaki çalıştırılan komutu işaretler.
- **Flags (Bayrak) register**
  - **Flags register**, CPU tarafından, matematiksel operasyonlardan sonra otomatik olarak değiştirilir. Bu register sayesinde, elde edilen sonucun çeşidi ve durumu ile ilgili bilgiler, program tarafından kullanılabilir.

3. Özel Amaçlı Register'lar

- Genellikle, bu register'lara doğrudan erişim bulunmaz.
- Ancak, sistem register'larının değerleri, daha sonra öğreneceğiniz bazı metotlar sayesinde değiştirilebilir.

Flag Bit’leri



- **C (carry):** toplama işleminden oluşan elde ve çıkarma işleminden oluşan ödünçleri tutar.
  - Ayrıca, hata durumlarını gösterir
- **P (parity):** bir sayıda bulunan 1’lerin tek sayıda mı yoksa çift sayıda mı olduğunu belirtir.
  - 0 tek parity; 1 çift parity.
  - Eğer bir sayı 3 tane binary 1 bit var ise, sayı tek parity’ye sahiptir.
  - Eğer bir sayıda hiç 1 bit yok ise, sayı çift parity’ye sahiptir.

Flag Bit’leri (devam)

- **A (auxiliary carry):** 3 ve 4. bit pozisyonları için geçerli olmak üzere, toplama işleminden oluşan elde ve çıkarma işleminden oluşan ödünçleri tutar.
- **Z (zero):** aritmetik veya mantık operasyonunun sonucunun sıfır olup olmadığı bilgisini tutar.
- **S (sign):** çalıştırılan aritmetik veya mantık operasyonunun sonucunda elde edilen sayının aritmetik yönünü (pozitif veya negatif) belirtir.

Flag Bit'leri (devam)

- **T (trap):** çip üzeri debug özelliğine olanak tanır.
- **I (interrupt):** INTR (**int**errupt **r**equest – interrupt isteği) girişini kontrol eder.
- **D (direction):** DI ve/veya SI register'ları için arttırma veya azaltma modlarından birini seçer.
- **O (overflow):** iki yönlü sayının toplamı veya çıkarılması durumlarında kullanılır.
  - overflow olması, sonucun, 16-bitlik kapasiteyi aştığını gösterir.

Adresleme Modları

Adresleme modları belleğin nasıl kullanıldığını, belleğe nasıl erişileceğini ve verilerin belleğe nasıl yerleştirileceğini belirler. Aşağıda verilen anlatımda kullanılan kısaltmaları anlamaları şu şekildedir.

Register: Kaydedici (Yazmaç)

Memory: Bellek

İmmediate : Acil veri, doğrudan veri

Acil Adresleme(Immediate Addressing)
(Anlık Adresleme)

Doğrudan sabit bir değer bir kaydediciye aktarılır. Sabit değerin büyüklüğü ile register uyumlu olmalıdır. **Örneğin** 8 bitlik bir kaydediciye 16 bitlik bir değer yüklenemez.

Genel Kulanımı:

KOMUT register, immediate

Örnek:

MOV CL, 16h

MOV DI, 2ABFh

MOV AL, 4567h ; Yanlış

Kaydedici Adresleme (Register Addressing) (Yazmaç Adresleme)

Bu adresleme modunda her iki operand (işlenen) de kaydedicidir.

Genel kullanımı:

KOMUT register, register

Örnek:

MOV AL, BL

INC BX

DEC AL

SUB DX, CX

Doğrudan adresleme(Direct addressing)

Doğrudan bir adres değeri kullanılır. Bir adresten bir kaydediciye very aktarımı gerçekleştirilir. Bir başka ifade ile operandlardan birisi adres belirtir.

Genel kullanımı:

KOMUT register, memory veya

KOMUT memory, register

Örnek:

```
MOV AX, [1000]
TOPLAM   DW  20    ;  Burada TOPLAM bir adrestir. 20 sayısı bu adresin içindeki değerdir.
MOV AX, TOPLAM     ;  TOPLAM adresindeki değeri AX e at.
MOV TOPLAM, AX
```

Dolaylı adresleme(Indirect addressing)
(Kaydediciye dayalı dolaylı adresleme)

Etkin adres değeri (offset) BX, BP, SI, DI kaydedicilerinden birinde bulunur.

Genel kullanımı:

KOMUT register, [BX/BP/SI/DI] veya

KOMUT [BX/BP/SI/DI], register

Örnek:

```
MOV AX,[SI]
MOV BX,1000  ;
SUB DX, [BX]
MOV [SI], AL
TABLO DB 5,9,0,3,-7
MOV BX, OFFSET TABLO   ;   Bunun yerine LEA BX,TABLO kullanılabilir.
MOV AX, [BX]           ;  Kaydedici dolaylı adresleme var
```

Dolaylı adresleme(Indirect addressing) (Kaydediciye dayalı dolaylı adresleme) (2)

LEA komutu (Load Efective Addres) ofset adresi bir keydediciye yüklemek için kullanılır.

Yukardaki iki komut yerine aşağıdaki komut kullanılabilir.

MOV AX, TABLE ; doğrudan adresleme var. İlk iki değer AX e yüklenir.

Herhangi bir bellek bölgesi dolaylı bir adresleme ile adreslenip içine sabit bir değer atanmak istendiği zaman atanacak değerin uzunluğu byte ptr ve word ptr ile belirtilmelidir.

```
MOV  [BX], 12                ; yanlış 00
MOV  byte ptr [BX],12        ; doğru  12 sabit değeri bayt olarak BX in
                                      gösterdiği adrese yerleşecek
MOV  word ptr [BX],1234      ;doğru  1234 sabit değeri word olarak BX in
                             gösterdiği adrese yerleşecek
```

İndisli adresleme (Indexed addressing)

Dolaylı adreslemede köşeli parantez içinde kalan BX/BP/SI/DI kaydedicilerine bir indis değeri eklenerek kullanılır.

Genel kullanımı:

KOMUT register, [BX/BP/SI/DI+indis] veya

KOMUT [BX/BP/SI/DI+indis], register

Örnek:

```
MOV AX, [SI+4]
ADD [DI-6],CX
MOV CX, [SI+DI+7] ;
MOV AX, [BX+DI]
MOV AX, [SI-7]          ; bu komut yerine MOV AX, [SI]-7  de kullanılabilir.

MOV   DI, 2
MOV   AL, TABLE [DI]   ;AL  kaydedicisine TABLE dizisinin 2. Elemanını yükler.
```

DİKKAT

1-) KOMUT memory, memory

Şeklinde bir kullanım geçerli değildir. Bir bellek bölgesinde başka bir bellek bölgesine veri aktarımı yoktur.

2-) Sabit bir değer doğrudan Segment Registerine atanamaz.

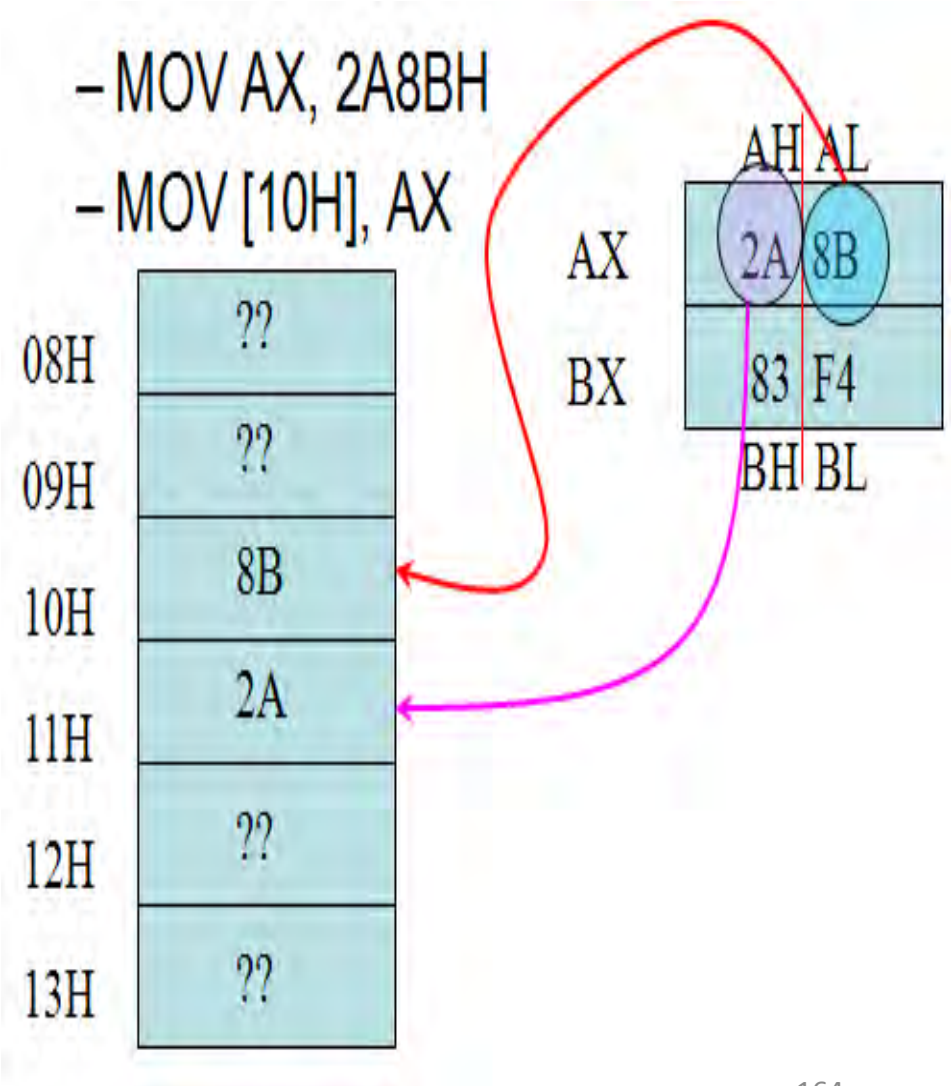
MOV DS,1234 Yanlıştır. Bunun yerine aşağıdaki kullanım geçerlidir.

MOV AX, 1234 ; önce kaydediciye

MOV DS, AX ; sonra segment kaydedicisine değer atanır.

16 Bitlik bir verinin belleğe yerleşmesi

Belleğin her bir hücresi 8 bit olduğundan 16 bitlik verinin düşük değelikli 8 bitlik kısmı adresin düşük değerlikli kısmına, yüksek değerlikli 8 bitlik kısmı adresin yüksek değerlikli kısmına yerleşir. Aşağıda verilen örnekte AX kaydedicisinde 2A8BH değeri yer almaktadır. Düşük değerlikli değer AL de yer almakta ve 10H adresine yerleşmektedir. Yüksek değerlikli kısımda yer alan AH daki değer 2AH değeri ise 11H adresine yerleşmektedir.



Benzer olarak 32bitlik bir veriyi(doubleword) bu şekilde düşünerek yerleştirebilirsiniz.
Örneğin:
SAYI DB 1A2F56FFH verisini 100H adresinden itibaren yerleştirecek olursa şu şekilde olur.

Adres	veri
100	FF
101	56
102	2F
103	1A
104	
105	
106	

Dizi Tanımlaması ve Elemanlarına Erişimi

DIZI DB 5, 6, 7, 8, 9, 0, -6, -9,3 şeklinde tanımlaması yapılır. LEA SI, DIZI ; şeklinde dizinin başlangıç adresini SI kaydedicisine alınır.

Örnek: Bir dizide bulunan elemanların toplamını bulan program kodunu yazınız.

```
.MODEL SMALL
      .STACK 64
      .DATA
         DIZI DB 5, 6, 7, 8, 9, 0,-6,-9, 3, 8
         SONUC DW ?
      .CODE
 ANA PROC FAR
      MOV AX,@DATA
      MOV DS, AX
     MOV AL,0
   MOV CX,10
     LEA SI, DIZI   ;   DIZI nin baslangic ofset adresi SI ya
yüklenir
BAS:
     MOV BL,[SI]
     ADC AL, BL
     INC SI
     LOOP BAS
     MOV SONUC, AL
     MOV AH,4CH
     INT 21H
ANA ENDP
  END ANA
```

Örnekler

Aşağıdaki komutların adresleme modlarını bulun

- MOV DH,[BX+DI+20H]
- MOV AL,BL
- JMP etiket1
- MOV SP,BP
- MOV AX,dizi
- MOV CH,[BP+SI]
- MOV AX,dosya[BX+DI]
- MOV [DI],BH
- MOV AX,44H
- MOV [BX+SI],SP
- MOV AL,sayı
- MOV AX,[DI+100H]
- MOV BL,44
- MOV dizi[SI],BL
- MOV liste[SI+2],CL
- MOV CX,[BX]

Aritmetik İşlemler

- Aritmetik işlemler toplama, çıkartma, çarpma ve bölme işlemlerini kapsamaktadır. Tüm aritmetik ve mantık komutları bayraklara etkilemektedir. Bu sayede aritmetik işlem sonucunda bayrak kaydedicisine bakılarak, sonuç sıfır mı(sıfır-zero bayrağı), elde var mı(carry bayrağı) büyük mü, küçük mü gibi kontrollerde yapılabilmekte ve işlem akışı buna göre yönlendirilebilmektedir.

TOPLAMA ve ÇIKARMA İŞLEMİ

X86 tabanlı işlemcilerin kendi bünyesindeki elektronik kapıları kullanarak gerçekleştirdiği aritmetik işlemlerden toplamayla ilgili iki komut vardır. Bunlar sırası ile toplama ADD ve çıkarma SUB komutlarıdır. Elde işleme dahil edilmek istenirse ADC ve SBB komutları kullanılır.

ADD : Toplama(eldesiz) – Addition

ADC : Toplama(eldeli - Carry – C) – Add with carry

SUB : Çıkarma(eldesiz) – Subtraction

SBB : Çıkarma(eldeli - Carry – C) – Subtract with borrow – Elde bayrağı borç için kullanılır.

Genel Kulanımı

ADD/SUB register, register	Ör: ADD AL, BL
ADD/SUB memory, register	Ör: ADD [SI], BL
ADD/SUB register, memory	Ör: ADD AL, [BX]
ADD/SUB register, immediate	Ör: ADD CL,12
ADD/SUB memory, immediate	Ör: ADD [SI], 25

Add ah,25h; ah’ deki değere 25 sayısını ekle

Add ah,bh; ah’a bh’deki değeri ekle

Add al,[bx]; bx’in gösterdiği yerdeki veriyi al’ye ekle

Add ax,alan; alan adresindeki Word değerini ax’e ekle

Add ax,[bx+4]; adresten ax’e ekle

Add bos,al; al’deki değeri bos adlı adresteki değere ekle

Adc dx,cx; cx’deki değeri c(carry) ile birlikte dx’e ekle

Adc [bx],25; 25+c’ yi bx’in gösterdiği adresteki değere ekle

Örnek:

```
mov ah,10
mov al,20
x db 85

add ah,al
add ah,x

mov ah,255
mov al,255
add ax,1
add ax,1
```

```
mov ah,255
mov al,255
adc ax,1
adc ax,1

mov ax,0
sub ax,1
sub ax,1

mov ax,0
sbb ax,1
sbb ax,1
hlt
```

SAYI1 DD 0123BC62H
SAYI2 DD 0012553AH
MOV EAX,SAYI1
ADD EAX,SAYI2

Yukarıda verilen son iki komutun yerine tek bir komut
ADD SAYI1,SAYI2 yazılabilir mi?

Eğer EAX gibi 32 bitlik bir kaydedici kullanmadan 16 bit olarak toplamak istersek ne yapacağız. Bunun için sayıları 16 bit olarak tanımlamalıyız. Öncelikle sayıları 2 ye ayıracağız.

Bu sayıların düşük sıralı ilk 16 bitlik kısmını (düşük değerli – az önemli tarafı – LSB – Least Significant Bit/Byte) tarafını SAYI1LSB de, büyük sıralı 16 bitlik kısmını (en değerli – En önemli tarafı – LSB – Most Significant Bit/Byte) SAYI1MSB de saklayalım. Ve işlemi 2 adımda gerçekleştirelim.

```
SAYI1LSB   DW 0BC62H
SAYI1MSB  DW 0123H
SAYI2LSB   DW 553AH
SAYI2MSB  DW 0012H
SONUCLSB   DW    ?
SONUCMSB  DW    ?

MOV AX,SAYI1LSB
ADD AX, SAYI2LSB
MOV SONUCLSB, AX
MOV  AX, SAYI1MSB
ADC AX, SAYI2MSB
MOV SONUCMSB , AX
```

Genelleştirilmiş bir toplama programı yazarsak şu şekilde olmalıdır.

```
LEA Sı,SAYı1LSB
LEA Dı,SAYı2LSB
LEA BX,SONUCLSB

CLC                 ; eldeyi sıfırla C=0
MOV   CX,02         ; döngü sayısı 2
TEKRAR:
MOV AX, [Sı]
ADC AX, [Dı]
MOV [BX],AX
INC Sı
INC SI
INC Dı
INC DI
INC BX
INC BX
LOOP TEKRAR
```

ÇARPMA İŞLEMİ

Bir çarpma işleminde, işlemci iki operand tanımı ister. Eğer bayt operandı tanımlanmış ise, işlemci diğer operandın **AL** olduğunu varsayar. Eğer Word operandı kullanılıyorsa, **AX** varsayar. Bu komutla işaretsiz sayılar için **MUL,** işaretli sayılar için ise **IMUL**' dur.

```
MUL     BH      ;     BH içeriğini AL ile çarp
MUL     ALAN    ;     bellek içeriğini AL ile çarp
MUL     DX      ;     DX içeriğini AX ile çarp
IMUL    BX      ;     işaretli BX değerini AX ile çarp
IMUL    VERI    ;     işaretli bellek verisini AL ile çarp
```

İki Word değeri çarpıldığında, işlemci daima 4 baytlık (32 bit) sonuç üretir. Sonucun önemli 16 bitlik kısmı(MSB) DX de ve az önemli 16 bitlik kısmı (LSB) ise **AX**' de tutulur. Burada **DX**, çarpma sonrasında oluşan genişleme değerini tutar.

Komut	Çarpan değer	Çarpılan	Sonuç
MUL CL	Bayt (8 bit)	AL	AX
MUL BX	word (16 bit)	AX	DX:AX
MUL EBX	Double word(32 bit)	EAX	EDX:EAX

Örnekler

```
SAYI1 DB 25
MOV AL,15
MUL SAYI1    ;    Sonuç=AX=AL*SAYI1


MOV AX,15FAH
MOV BX,21CEH
MUL BX    ;    Sonuç=DX:AX=AX*BX
```

ÖRNEK:

```
mov al,2
mov bl,3
mul bl

mov ax,259
mov bx,300
mul bx

mov bl,0
sub bl,3
mov al,2
imul bl

mov ax,189
mov bx,0
sub bx,10
imul bx
```

BÖLME İŞLEMİ

Bölme işlemlerinde de iki komut kullanılır. İşaretsiz sayılarda **DIV**, işaretliler de ise **IDIV** kullanılır.

Komut	**Bölen değer**	**Bölünen değer**	**Bölüm**	**Kalan**
DIV CL	Bayt (8 bit)	AX	AL	AH
DIV CX	word (16 bit)	DX:AX	AX	DX
DIV EBX	Double word(32 bit)	EDX:EAX	EAX	EDX

Bölünen (dividend) **AX** de iken, eğer kaynak operand bayt ise, bölüm (quotient) değeri **AL**' ye dönerken, kalan (remainder) değer **AH** 'da tutulur.

Bölünen **DX** ve **AX** çiftinde iken, eğer kaynak operand Word ise, bölüm değeri **AX**' e dönerken, kalan değer **DX**' de tutulur.

```
Div cx                ;dx:ax' deki değeri cx' deki değere böl
Div alan      ;ah: al 'deki değeri alan'daki değere böl
Idiv dh       ;ah:al' deki değeri dh'deki değere böl
Idiv kon      ;dx:ax' deki değeri kon'daki değere böl

Örnek:
MOV CL,12
MOV AX,45AEH
DIV CL      ;     AL=Bölüm,   AH=Kalan
```

ÖRNEK:

```
MOV AX,11
MOV CL,2
DIV CL

MOV AX,55

MOV CL,0
SUB CL,10
IDIV CL
```

Artırma Azaltma Komutları

INC: Increment
INC reg
INC mem
Tek işlenenli bir komuttur ve işlenenin değerini 1 artırmaktadır.
INC AX
INC WORD PTR[1002]
INC SAYI

DEC: Decrement
DEC reg
DEC mem
Tek işlenenli bir komuttur ve işlenenin değerini 1 azaltmaktadır.
DEC AX
DEC WORD PTR[1002]
DEC SAYI

Mantıksal Komutlar

AND

OR

XOR

NOT

TEST

And Komutu

Yapı olarak AND (VE) mantığıyla; 1 ve 0'lar ile ifade edilirse;

AND reg,idata
AND mem,idata
AND reg,reg
AND reg,mem
AND mem,reg

1 ve 1 = 1

1 ve 0 = 0

0 ve 1 = 0

0 ve 0 = 0

Sonuçları üretir. Genelde maskeleme amaçlı kullanılır.

And Komutu

Örnek:

MOV AL, A5H

AND AL, 0FH

Bu işlemlerden sonra AL' in yüksek değerlikli 4 biti (nibble) sıfırlanacaktır yani AL binary olarak ifade edilirse 0000 0101 olacaktır. Buna düşük değerlikli 4 bite dokunmadan diğer bitleri sıfırlamakta denilebilir.

OR Komutu

Mantıksal veya işlemini gerçekleştirir.

1 veya 1 = 1

1 veya 0 = 1

0 veya 1 = 1

0 veya 0 = 0

OR mem,reg

OR komutu da AND komutu gibi çalışır ve maskeleme işlemi için kullanılabilir.

Örnek:

OR AL,00010000B; Burada 4. Biti 1 yapmıştır.

XOR Komutu

Mantıksal özel veya işlemini gerçekleştirir. Aynıysa sıfır, farklıysa 1 üretir.

1 ^ 1 = 0

1 ^ 0 = 1

0 ^ 1 = 1

0 ^ 0 = 0

NOT Komutu

Mantıksal değil işlemini gerçekleştirir. Birse sıfır, sıfırsa 1 üretir.

NOT reg

NOT mem

Test komutu

AND işlemi yapar. Ancak sonuç hedefi etkilemez. Sadece bayrakları etkiler. AND komutunda ise sonuç hedefi etkiler.

Test komutunun yalnızca bayrakları etkilemesi sebebiyle hemen ardından şartlı dallanma komutu kullanılarak işlem yapılır.

Örnekler

Mod Alma:

AND SAYI, 1; Sayının 2 ye bölümünden kalanı alır. Yani 2 ye mod alır.

Küçük-Büyük Harf Çevrimi:

Yol: (A-Z) 41h-5Ah arası; (a-z) 61h-7Ah ‘ tır. Yani küçük harf ile büyük harf arasında 20h’lık bir fark vardır.

Yol: MSB bitinin birinci biti 1 yapılırsa küçük harfe çevrilir.

Örnek: Yazı DB ‘Karabuk Universitesi&’ tanımlanıyor bu string ifadeyi büyük harfe çeviren program kodunu yazınız.

Örnekler (devam...)

```
.MODEL SMALL
      .STACK 64
      .DATA
  YAZI DB 'Karabuk Universitesi&'
        .CODE
 ANA PROC FAR
      MOV AX,@DATA
      MOV DS, AX

    MOV AL,0
    MOV CX,20
    LEA SI, YAZI
BAS:
      MOV AL,[SI]
      CMP AL, 61H
      JA GIT
DON:
        INC SI
        LOOP BAS
GIT:
      SUB AL,20H ; 20H çıkar ve büyük harfe
dönüştür.
       MOV [SI],AL
      JMP DON

MOV AH,4CH
      INT 21H
ANA ENDP
  END ANA
```

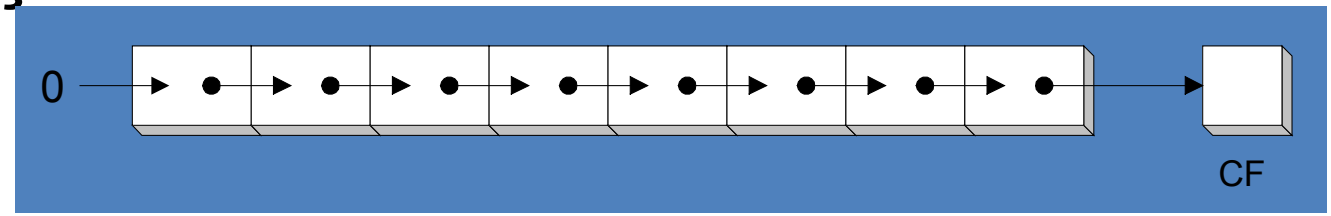
KAYDIRMA VE DÖNDÜRME KOMUTLARI

- **Kaydırma Komutları**
  - **SHR(Shift Right)**
  - **SHL(Shift Left)**
  - **SAR(Shift Aritmetik Right):**
  - **SAL(Shift Aritmetik Left):**

- **Döndürme Komutlari**
  - **ROR(Rotate Right):**
  - **ROL(Rotate Left):**

SHR(Shift Right):

Bitler sağa doğru kayar ve 1. bit Carry'e düşer.

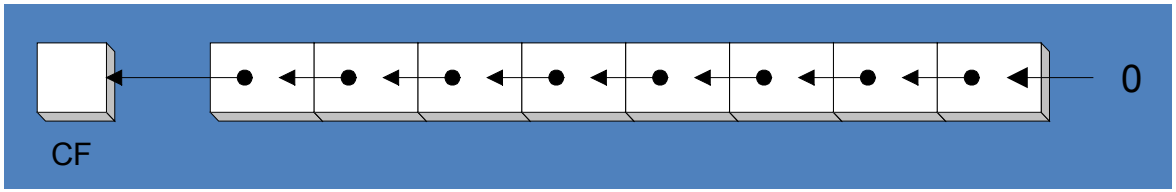


n bit sağa kaydırma openandı $2^{n'}$e böler.

```
MOV DL,85
SHR DL,1                    ; DL = 42, CF=1
SHR DL,2                    ; DL = 10, CF=1
```

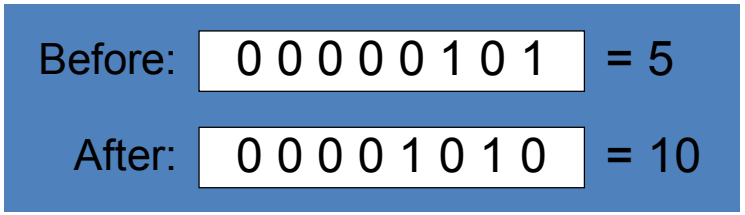
SHL(Shift Left)

Bitler sola doğru kayar ve 8. Bit Carry'e düşer.



Örnekler:

```
MOV DL,5
SHL DL,1
```



N bit sola kaydırma işlemi ile operand 2^n ile çarpılmaktadır.

Ör: $5 * 2^2 = 20$

Örnek

```
MOV AL,11011011B
SHR AL,1 ; 01101101
SHR AL,1 ; 00110110
SHR AL,1 ; 00011011
SHR AL,1 ; 00001101
SHR AL,1 ; 00000110
SHR AL,1 ; 00000011
SHR AL,1 ; 00000001
SHR AL,1 ; 00000000
```

```
MOV AL,00110011B
SHL AL,1 ;01100110
SHL AL,1 ;11001100
SHL AL,1 ;10011000
SHL AL,1 ;00110000
SHL AL,1 ;01100000
SHL AL,1 ;11000000
SHL AL,1 ;10000000
SHL AL,1 ;00000000
RET
```

Örnek

```
MOV AL,30H          ;AL=30H
SHR AL,01                ;AL=18H
SHR AL,01                ;AL=0CH

MOV AL, 00000111b
SHR AL, 1                ; AL = 00000011b,
   CF=1
```

Örnek

```
MOV DL,5
SHL DL,2                ; DL = 20

MOV AL, 11100000b
SHL AL, 1               ; AL = 11000000b, CF=1
```

Çarpma

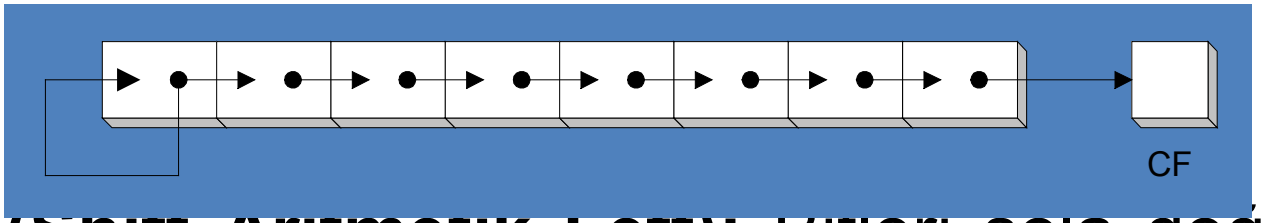
Bildiğimiz gibi SHL komutu 2^n ile işaretsiz çarpma yapabilmektedir. 2 nin kuvveti olmayan bir sayıyı 2 nin kuvvetlerine bölerek çarpma yapabiliriz. Örneğin EAXx36 işlemini yapmak istiyoruz. 36 yı 32+4 şeklinde parçalarız. Ve iki adet kaydırma işlemi yaparak sonuçlarını topladığımızda 36 ile çarpmış oluruz.

```
EAX * 36
= EAX * (32 + 4)
= (EAX * 32)+(EAX * 4)
```

```
MOV EAX,123
MOV EBX,EAX
SHL EAX,5          ; 2^5 ile çarp
SHL EBX,2          ; 2^2 ile çarp
ADD EAX,EBX
```

SAR & SAL

- **SAR(Shift Aritmetik Right):** Bitleri sağa doğru 1'er bit kaydırır. En soldaki bitin değeri de kaydırılır fakat eski değeri korunur. Bu bit işaretli sayılarda sign biti olarak kullanılır.



- **SAL(Shift Aritmetik Left):** Bitleri sola doğru 1'er bit kaydırır. En soldaki bitin değeri de kaydırılır fakat eski değeri korunur.

Örnek

```
MOV AL,11011011B
SAR AL,1 ; 11101101
SAR AL,1 ; 11110110
SAR AL,1 ; 11111011
SAR AL,1 ; 11111101
SAR AL,1 ; 11111110
SAR AL,1 ; 11111111
SAR AL,1 ; 11111111
SAR AL,1 ; 11111111
```

```
MOV AL,11011010B
SAL AL,1 ; 10110100
SAL AL,1 ; 01101000
SAL AL,1 ; 11010000
SAL AL,1 ; 10100000
SAL AL,1 ; 01000000
SAL AL,1 ; 10000000
SAL AL,1 ; 00000000
SAL AL,1 ; 00000000
```

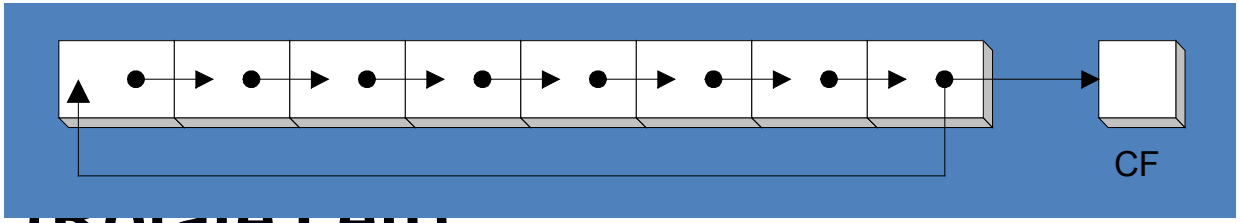
Örnek

```
MOV AL, 0E0h        ; AL = 11100000b
SAL AL, 1           ; AL = 11000000b, CF=1
MOV AL, 0E0h        ; AL = 11100000b
SAR AL, 1           ; AL = 11110000b, CF=0
MOV BL, 4Ch         ; BL = 01001100b
SAR BL, 1           ; BL = 00100110b, CF=0
MOV DL,-85
SAR DL,1    ; DL = -43, CF= 1
SAR DL,2    ; DL = -11, CF= 0
```

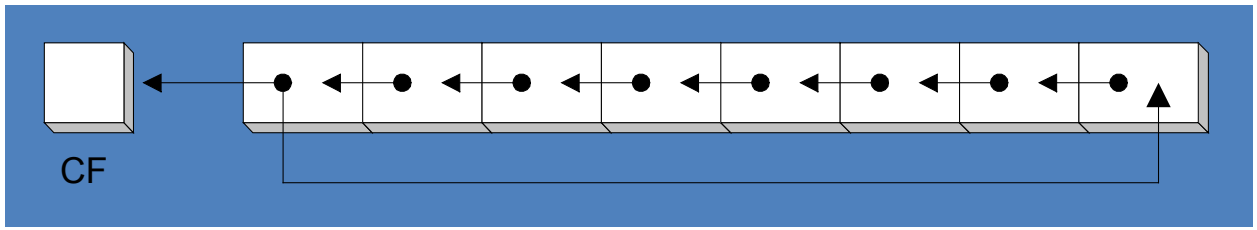
ROR & ROL

Bu komutlar ile sayı kaybolmaz.Sağdan-Sola ya da Soldan-Sağa bitlerdeki değerler Carry bayrağına düşer ve Carry bayrağındaki değer son bite yerleşir.

ROR (Rotate Right):



ROL (Rotate Left):



Örnek

```
MOV AL,11011010B
ROR AL,1 ;01101101
ROR AL,1 ;10110110
ROR AL,1 ;01011011
ROR AL,1 ;10101101
ROR AL,1 ;11010110
ROR AL,1 ;01101011
ROR AL,1 ;10110101
ROR AL,1 ;11011010
```

```
MOV AL,11011010B
ROL AL,1 ; 10110101
ROL AL,1 ; 01101011
ROL AL,1 ; 11010110
ROL AL,1 ; 10101101
ROL AL,1 ; 01011011
ROL AL,1 ; 10110110
ROL AL,1 ; 01101101
ROL AL,1 ; 11011010
```

ROR & ROL Örnek

```
MOV AL,11110000b
ROL AL,1                ; AL = 11100001b, CF = 1
MOV DL,3Fh              ;DL= 00111111b
ROL DL,4                ; DL = 11110011b, CF=1
MOV AL,11110000b
ROR AL,1                ; AL = 01111000b, CF = 0
MOV DL,3Fh              ;DL= 00111111b
ROR DL,4                ; DL = 11110011b h, CF=1
MOV AL, 1Ch             ; AL = 00011100b
ROL AL, 1               ; AL = 00111000b, CF=0
MOV AL,1Ch              ; AL = 00011100b
ROR AL, 1               ; AL = 00001110b, CF=0
```

Ödev-1

Her kaydırmadan sonraki AL kaydedicisi ve elde bayrağının değerini belirleyiniz.

MOV AL,6Bh

SHR AL,1

SHR AL,3

MOV AL,8Ch

SAR AL,1

SAR AL,3

Ödev-2

Her kaydırmadan sonraki AL kaydedicisinin hexadesimal karşılığını yazınız.

MOV AL,6BH
ROR AL,1
ROL AL,3

Ödev-3

AX’e 2H değerini yükleyiniz. AH’ı 26 ile kaydırma ve toplama komutlarını kullanarak çarpınız.

(26=16+8+2)

KONTROL KOMUTLARI

Program Kontrol Komutları

Program akışını bir noktadan bir başka noktaya yönlendirmek amacı ile kullanılan komutlardır. Bu komutlar aşağıda listelenmiştir.

- Şartsız Dallanma Komutu; JMP
- Döngü Komutları; LOOP
- Karşılaştırma Komutu : CMP
- Şartlı Dallanma; JE,JZ,JNZ….
- Alt Program Çağrısı; CALL
- Bayraklar İle İlgili Komutlar; CLC,STC...

Şartsız dallanma komutu

- JMP, programı belirtilen etiketin olduğu yere dallandırmakta ve program buradan çalışmaya devam etmektedir.

JMP Hedef

Döngü komutu

- LOOP komutu, genellikle bir iş birden fazla yapılacağı zaman CX registeri ile birlikte döngü kurmayı sağlamaktadır.

CX registerı içinde döngünün adedi tutulur. Her loop komutu çalıştığında CX'in değeri 1 azalır ve CX sıfırlandığında döngü biter.

```
        Mov CX,5
Topla: .....
        ......
        Loop Topla
```

Karşılaştırma Komutu

- CMP komutunun kullanımı
- CMP *deger1* , *deger2*

CMP register, register;

CMP AX, BX

CMP register, memory;

CMP AX, SONUC

CMP register, immediate;

CMP AX,5

CMP memory, register;

CMP SONUC, AX

CMP memory, immediate;

CMP SONUC,5

- CMP komutu kullanıldığı zaman aşağıdaki etkilenen bayrakların durumu verilmiştir.
- **Not**: CMP 50(veri),50(veri) şeklinde kullanılamaz aşağıda küçük ve büyük olma durumlarını açıklamak için o şekilde örnek verilmiştir.

	C	**Z**	**S**
CMP 50,60	1	0	1
CMP 50,50	0	1	0
CMP 50,40	0	0	0

Şartlı Dallanma Komutları

- Şartlı Dallanma Komutları genellikle bir CMP komutunu takiben program akışını başka bir noktaya kaydırmak amacıyla kullanılır. Şartlı dallanma komutları bayrakların durumuna bakarak hangi noktaya (etiket-label) gidileceğini belirlemektedir.

- Koşullu dallanmada dallanma aralığı 8 bit ile sınırlıdır. –128 veya +127 bayt'dan daha uzak noktalara dallanma söz konusuysa koşulsuz dallanma komutları kullanılmalıdır.

Bayrakların Durumunu Test Etmek İçin Kullanılan Koşullu Dallanma Komutları

Komut	Tanımlama	Durum	Karşıtı
JZ, JE	Sıfır (eşit) - Jump if Zero (Equal)	Z = 1	JNZ, JNE
JC, JB, JNAE	Carry (küçük; eşitten büyük değil) - Jump if Carry (Below, Not Above Equal)	C = 1	JNC, JNB, JAE
JS	Yönlü - Jump if Sign	S = 1	JNS
JO	Overflow - Jump if Overflow	O = 1	JNO
JPE, JP	Çift parity - Jump if Parity Even	P = 1	JPO, JNP
JNZ, JNE	Jump if Not Zero (Not Equal) (sıfır değil ise)	Z = 0	JZ, JE
JNC, JNB, JAE	Jump if Not Carry (küçük değil; eşitten büyük - Not Below, Above Equal)	C = 0	JC, JB, JNAE
JNS	Yönlü değil - Jump if Not Sign	S = 0	JS
JNO	Overflow değil - Jump if Not Overflow	O = 0	JO
JPO, JNP	Tek parity - Jump if Parity Odd (No Parity)	P = 0	JPE, JP

İşaretsiz Sayılarda Şartlı Dallanma Komutları

Instruction	Description	Condition	Opposite
JE, JZ	Eşit ise - Jump if Equal (=) Sıfır ise - Jump if Zero	Z = 1	JNE, JNZ
JNE, JNZ	Eşit değil ise - Jump if Not Equal (≠) Sıfır değil ise - Jump if Not Zero	Z = 0	JE, JZ
JA, JNBE	Büyük ise - Jump if Above (>) Küçük veya eşit değil ise - Jump if Not Below or Equal	C = 0 and Z = O	JNA, JBE
JBE, JNA	Küçük veya eşit ise - Jump if Below or Equal (<=) Büyük değil ise - Jump if Not Above	C = 1 or Z = 1	JNBE, JA
JB, JNAE, JC	Küçük ise - Jump if Below (<) Büyük veya eşit değil ise - Jump if Not Above or Equal Carry ise - Jump if Carry	C = 1	JNB, JAE, JNC
JAE, JNB, JNC	Büyük veya eşit ise - Jump if Above or Equal (>=) Küçük değil ise - Jump if Not Below Carry değil ise - Jump if Not Carry	C = 0	JB, JNAE, JC

İşaretli Sayılarda Şartlı Dallanma Komutları

Instruction	Description	Condition	Opposite
JE, JZ	Eşit is e - Jump if Equal (=) Sıfır ise - Jump if Zero	Z = 1	JNE, JNZ
JNE, JNZ	Eşit değil ise - Jump if Not Equal (≠) Sıfır değil ise - Jump if Not Zero	Z = 0	JE, JZ
JG, JNLE	Büyük ise - Jump if Greater (>) Küçük veya eşit değil ise - Jump if Not Less or Equal (not <=)	Z = 0 and S = O	JNG, JLE
JL, JNGE	Küçük ise - Jump if Less (<) Büyük veya eşit değil ise - Jump if Not Greater or Equal	S ≠ O	JNL, JGE
JGE, JNL	Büyük veya eşit ise - Jump if Greater or Equal (>=) Küçük değil ise - Jump if Not Less	S = O	JNGE, JL
JLE, JNG	Küçük veya eşit ise - Jump if Less or Equal (<=) Büyük değil ise - Jump if Not Greater	Z = 1 or S ≠ O	JNLE, JG

Alt Program Çağrısı

- Adına prosedür denilen program parçaları ana program içerisinde her çağrılmak istendiğinde Şartsız dalma komutu CALL kullanılır.

- Prosedürün sonunda bulunan RET komutuyla proram kaldığı yere geri döner.

Bayraklar İle İlgili Komutlar

Bazı komutların istenen şekilde çalışabilmesi için önkoşul olarak bayrakların değerlerinin ayarlanması gerekir.

Komut	Etki
CLC	C=0
CMC	C=C’
STC	C=1
CLD	D=0
STD	D=1
STI	I=1
CLI	I=0
LAHF	AH=bayrak
SAHF	bayrak=AH

Sık ihtiyaç duyulan bayrakların değerini bağımsız olarak değiştiren komutlara karşılık, seyrek ihtiyaç duyulan bayrak değerlerinde değişim için AH registeri kullanılır.

S	Z	?	A	?	P	?	C
S	Z	0	A	0	P	0	C

Örnek1: 1'den 100 'e kadar olan sayıların toplamını bulup sonucu SONUC değişkenine atan programı yazınız.

1. yol şartlı dallanma komutları ile

```
        .MODEL SMALL
       .STACK 64
       .DATA
        SONUC DW  ?
       .CODE
 ANA PROC FAR
       MOV AX,@DATA
       MOV DS, AX
      MOV AX,00
      MOV CX,100
BAS:
      ADD AX, CX
      DEC CX
      JNE BAS; Sonuç sıfır değilse BAS a git
      MOV SONUC, AX
      MOV AH,4CH
      INT 21H
ANA ENDP
  END ANA
```

2.yol LOOP komutu ile

```
       .MODEL SMALL
       .STACK 64
       .DATA
          SONUC DW  ?
       .CODE
 ANA PROC FAR
       MOV AX,@DATA
       MOV DS, AX
      MOV AX,00
      MOV CX,100
BAS:
      ADD AX, CX
      LOOP BAS;   CX'i 1 azalt sıfıra eşit değilse BAS' a git
      MOV SONUC, AX
      MOV AH,4CH
      INT 21H
ANA ENDP
  END ANA
```

Örnek2: 5 ile 100 arasındaki sayıların toplamını bulup, sonucu SONUC değişkenine atan program kodunu yazınız.

```
        .MODEL SMALL
        .STACK 64
        .DATA
            SONUC DW  ?
        .CODE
  ANA PROC FAR
        MOV AX,@DATA
        MOV DS, AX
       MOV AX,5
BAS:
       ADD BX, AX
       INC  AX
       CMP AX, 100
       JBE BAS              ; AX 100 den küçük ve eşitken BAS' a git
       MOV SONUC, BX
       MOV AH,4CH
       INT 21H
ANA ENDP
   END ANA
```

Örnek 3:

```
ORG    100h

MOV    AX, 5
MOV    BX, 2
JMP    hesapla

geri:  JMP dur          ; dur etiketine git

Hesapla:
ADD    AX, BX          ; AX'e BX'i ekle
JMP    geri            ; geri etiketine git

dur:
RET                    ; İşletim sistemine dön
END                    ; derleyiciyi sonlandır
```

Örnek 4:

```
ORG    100h

MOV    AL, 25          ; AL=25
MOV    BL, 10          ; BL=10
CMP    AL, BL          ; AL ile BL'yi karşılaştır
JE     esit            ; eğer AL = BL (ZF = 1) ise esit'e git
MOV CL,'H'             ; Buraya gelirse AL <> BL demektir
JMP    dur             ; Bu yüzden CL'ye 'H' yükle ve dur'a git

esit:                  ; buraya gelirse
MOV CL,'E'             ;  AL = BL demektir bu yüzden CL'ye 'E' yaz

dur:
RET
END.
```

Örnek 5:

```
ORG    100h

MOV    AL, 25            ; AL=25
MOV    BL, 10            ; BL=10
CMP    AL, BL          ; AL ile BL'yi karşılaştır
JNE    esit degil      ; AL <> BL (ZF = 0) ise dallan
JMP    esit

esitdegil:
MOV CL,'H'             ; buraya geldiyse AL <> BL demektir.
JMP    dur             ; bu yüzden CL='H' ve dur'a git

esit:                  ; buraya geldiyse
MOV CL,'E'             ;  AL = BL demektir bu yüzden CL='E'

dur:
RET
END
```

Örnek 6:

```
org 100h
mov cx, 100                    ; Bloklardaki eleman sayısı
mov bx, 0                      ; index'i başlat
L1:
mov ax, BLOCK1[bx]             ;BLOCK1'deki sıradaki sayıyı al
Add ax,BLOCK2[bx] ;BLOCK2'deki sıradaki sayıyı ekle
mov BLOCK2[bx], ax             ; sonucu sakla
add bx, 2                      ; Bir sonraki elemana geç
loop L1
Ret
BLOCK1 DW 100 DUP (1)
BLOCK2 DW 100 DUP (2)
```

Örnek 7:

```
org 100h
mov cx, 100 ; Bloklardaki eleman sayısı
mov SI, offset BLOCK1
mov DI, offset BLOCK2
L1:
mov ax, [SI] ; BLOCK1'deki sıradaki sayıyı oku
add ax, [DI] ; BLOCK2'deki sıradaki sayıyı ekle
mov [DI], ax ; Sonucu sakla
add SI, 2    ; Bir sonraki elemana geç
add DI, 2
loop L1
Ret
BLOCK1 DW 100 DUP (1)
BLOCK2 DW 100 DUP (2)
```

Örnek 8 (İç içe döngü):

```
ORG 100H
MOV BX, 0 ; toplam adım sayıcısı
MOV CX, 5
k1:
INC BX
; kodlar
PUSH CX
MOV CX, 5
k2:
INC BX
 ; kodlar
PUSH CX
MOV CX, 5
 k3:
 ADD BX, 1
     ; kodlar
 LOOP k3
 POP CX
 LOOP k2 ; iç döngü
 POP CX
 LOOP k1 ; dış döngü
 RET
```

Örnek 8 (c dilinde yazılırsa):

```
int i,j,k,t;
t=0;
for(i=0;i<5;i++)
   {
    t=t+1;                          //bu satır 5 kez çalıştırılıyor
     for(j=0;j<5;j++)
       {
         t=t+1;                     //bu satır 25 kez çalıştırılıyor
            for(k=0;k<5;k++)
              {
              t=t+1;                //bu satır 125 kez çalıştırılıyor
              }
       }
   }

//t=155 olmaktadır.
```

Örnek 9: Örnek 8 aşağıdaki gibi düzenlenirse BX'te en son hangi sayı bulunur?

```
ORG 100H
MOV BX, 0
MOV CX, 5
k1:PUSH CX
MOV CX, 5
k2: PUSH CX
MOV CX, 5
k3:
ADD BX, 1
LOOP k3
POP CX
LOOP k2
POP CX
LOOP k1
RET
```

String Komutarı

MOVS

CMPS

SCAS

LODS

STOS

String Komutları

MOVS, bir bellek bölgesinin başka bir bellek bölgesine aktarılması
CMPS, farklı iki bellek bölgesinin içeriklerinin karşılaştırılması
SCAS, bellek bölgesinin içeriğinin EAX/AX/AL yazmacı ile karşılaştırılması
LODS, bellek bölgesindeki değerlerin EAX/AX/AL yazmacına yüklenmesi
STOS, Bellek bölgesinin EAX/AX/AL yazmacının değeri ile doldurulması

String Komutları

String komutları bellek alanı üzerinde byte, word ve double word olmaz üzere, farklı büyüklükler ile işlem yapmayı mümkün kılar. Bu komutlar genel olarak tekrarlamalı işlemleri gerçekleştirmek için tasarlanmıştır. İşlemlerin tekrar sayısı CX yazmacı ile belirlenirken dizi komutlarının önüne konulan REP öneki çevrimin yapılmasını mümkün kılmaktadır.

Bu gruba dahil komutlarda DF bayrağının değeri yapılan işlemin yönünü belirlediğinden oldukça önemlidir. DF=1 ise indis yazmacı azalır, 0 ise artar.
İndislerin artış azalış miktarı
Byte ise 1
Word ise 2
Double word ise 4'tür.

MOVSB (Move String Byte)

Byte bazında aktarma yapan bir komuttur. Komutun görevi DS:SI ikilisinin gösterdiği adresteki byte'ı ES:DI ikilisinin gösterdiği adrese aktarmaktır. MOVS komutu bayrakları etkilememektedir.
Yapılan işlem:
[ES:DI]←[DS:SI]
DI ←DI-1
SI ←SI-1

Ör:
LEA SI,*eskiyer*
LEA DI,*yeniyer*
MOV CX,10
CLD
REP MOVSB ;REP her seferinde CX'i bir azaltır. Her turda SI, ve DI yazmaçlarının değeri işlem byte türünde olduğundan 1 arttırılacaktır.

MOVSB (Move String Byte)

```
#make_COM#
ORG 100h
LEA SI, a1
LEA DI, a2
MOV CX, 5
REP MOVSB
RET
a1 DB 1,2,3,4,5
a2 DB 5 DUP(0)
```

MOVSW Move String Word

Word bazında aktarma işlemi yapan komuttur. Komutun görevi DS:SI ikilisinin gösterdiği adresteki wordü ES:DI ikilisinin gösterdiği adrese aktarmaktır. Word olması dolayısıyla DF bayrağının durumuna bağlı olarak artım ve azaltmalar ikişer olarak yapılmaktadır.

Yapılan işlem:
[ES:DI]←[DS:SI]
DI ←DI-2
SI ←SI-2

(DF=1 ise azaltma 0 ise artırma yapılır)

MOVSW Move String Word

```
#make_COM#
ORG 100h
LEA SI, a1
LEA DI, a2
MOV CX, 5
REP  MOVSW
RET
a1 DW 1,2,3,4,5
a2 DW 5 DUP(0)
```

MOVSD Move String Doubleword

Doubleword bazında aktarma işlemi yapan komuttur. Komutun görevi DS:SI ikilisinin gösterdiği adresteki doubleword'ü ES:DI ikilisinin gösterdiği adrese aktarmaktır. Doubleword olması dolayısıyla DF bayrağının durumuna bağlı olarak artım ve azaltmalar dörder olarak yapılmaktadır.

Yapılan işlem:
[ES:DI]←[DS:SI]
DI ←DI-4
SI ←SI-4

(DF=1 ise azaltma 0 ise artırma yapılır)

CMPSB Compare String Byte

Belleğin farklı adreslerindeki byte büyüklüğündeki değerlerin karşılaştırılmasında kullanılmaktadır. CMP komutunda olduğu gibi iki işlenenin birbirinden çıkarılması sonucu bayraklar etkilenmektedir.

[DS:SI]-[ES:DI] işlemden bayraklar etkilenir.

DI ←DI-1

SI ←SI-1

Örnek

```
LEA SI,DIZI1
LEA DI,DIZI2
MOV CX,10
CLD
REPE CMPSB
JNE UYMAZ
```

DIZI1 ve DIZI2 isimli bellek alanlarında bulunan 10 byte'lık verinin aynı olup olmadığı kontrol edilmek istenmektedir. CLD komutu ile DF=0 yapıldığı için SI ve DI değerleri her seferinde bir artırılacaktır. REPE öneki işlemin eşitlik söz konusu olduğu sürece devamını sağlar ve her turda CX sıfır oluncaya kadar SI ve DI değerleri otomatik olarak artırılır. Herhangi bir noktada eşitsizlik söz konusu olursa REPE koşulu bozulur ve işlem JNE komutu ile devam eder.

CMPSW Compare String Byte

Belleğin farklı adreslerindeki word büyüklüğündeki değerlerin karşılaştırılmasında kullanılmaktadır. CMP komutunda olduğu gibi iki işlenenin birbirinden çıkarılması sonucu bayraklar etkilenmektedir.

[DS:SI]-[ES:DI] işlemden bayraklar etkilenir.

DI ←DI-2

SI ←SI-2

CMPSD Compare String Byte

Belleğin farklı adreslerindeki doubleword büyüklüğündeki değerlerin karşılaştırılmasında kullanılmaktadır. CMP komutunda olduğu gibi iki işlenenin birbirinden çıkarılması sonucu bayraklar etkilenmektedir.

[DS:SI]-[ES:DI] işlemden bayraklar etkilenir.

DI ←DI-4

SI ←SI-4

SCASB Scan String Byte

SCASB komutu AL yazmacı ile ES:DI ikilisinin gösterdiği adreste bulunan byte'ı karşılaştırır. Bu karşılaştırmadan sadece bayraklar etkilenir. Bu komutta diğer benzerleri gibi REP ön ekleri ile bağlantılı olarak kullanılmaktadır.

Yapılan işlem:

AL-[ES:DI] Bayraklar etkilenir.

DI=DI-1

(DF 0 ise artırma olacaktır.)

Örnek

```
LEA DI,mesaj
MOV CX,000CH
CLD
MOV AL,'*'
REPNE SCASB
JE buldu
```

Mesaj isimli bellek alanının içinde '*' karakteri aranmaktadır. Bunun için DI yazmacına mesaj değişkeninin başlangıç adresi, AL'ye aranacak karakter,CX'e tekrar sayısı verilmiş, CLD ile DF=0 yapılarak DI yazmacının artan yönde değişeceği belirlenmiştir.
Buna göre işlem dizi içinde '*' karakteri bulunana kadar REPNE çevrimi içinde dönecek bu aşamada DI yazmacının gösterdiği adresteki veri ile AL'deki eşit olduğundan ZF=1 olacak REPNE sağlanamayıp çevrimden çıkılacaktır. Çevrimin normal yolla mı yoksa ZF=1 olduğu için mi bittiğini anlamak için koşullu dallanma komutu ile ZF bayrağının durumu kontrol edilerek arananın bulunup bulunmadığına karar verilmektedir.

SCASW Scan String Word

SCASW komutu AX yazmacı ile ES:DI ikilisinin gösterdiği adreste bulunan word'ü karşılaştırır. Bu karşılaştırmadan sadece bayraklar etkilenir. Bu komutta diğer benzerleri gibi REP ön ekleri ile bağlantılı olarak kullanılmaktadır.

Yapılan işlem:

AX-[ES:DI] Bayraklar etkilenir.

DI=DI-2

(DF 0 ise artırma olacaktır.)

SCASD Scan String Doubleword

SCASD komutu EAX yazmacı ile ES:DI ikilisinin gösterdiği adreste bulunan doubleword'ü karşılaştırır. Bu karşılaştırmadan sadece bayraklar etkilenir. Bu komutta diğer benzerleri gibi REP ön ekleri ile bağlantılı olarak kullanılmaktadır.

Yapılan işlem:

EAX-[ES:DI] Bayraklar etkilenir.

DI=DI-4

(DF 0 ise artırma olacaktır.)

LODSB Load String Byte

DS:SI ikilisinin belirttiği adresten bir byte'ı AL yazmacına yerleştirir. SI yazmacının değeri ise DF bayrağının durumuna bağlı olarak değiştirilir. LOADSB komutu bayraklar üzerinde bir değişikliğe neden olmaz.
Yapılan işlem:
AL=[DS:SI]
SI=SI-1;(DF=0 ise artma olacaktır)

Örnek

```
LEA SI,input
LEA DI,output
LEA BX,ascii2ebcdic
CLD
L1: LODSB
OR AL,AL
JZ L2
XLAT
STOSB
JMP L1
L2:.....
```

Örnekte input isimli bellek alanındaki ASCII karakterlerin EBCDIC karşılıkları output isimli bellek alanına yazılmaktadır. İşlem input alanında 00H görünceye kadar devam edecektir. Bu amaçla SI,DI ve BX yazmaçları sırası ile input, output ve ascii2ebcdic isimli bellek alanlarının başını gösterecek şekilde ayarlanmıştır. DF=0 yapıldığı için her işlemden sonra SI ve DI yazmacı artırılacaktır. L1 etiketi ile başlayan çevrimde LODSB ile AL yazmacına alınan değer OR işleminden geçirilerek (bu işlem yazmaç üzerinde bir değişikliğe neden olmaz) sadece bayraklar düzenlenir. ZF=1 olmuşsa L2 etiketine gidilerek akış sonlandırılmaktadır. Aksi halde XLAT komutu ile AL=[BX+AL] işlemi yapılmış ve daha önce ASCII değerini barındıran AL yazmacına bunun karşılığı olan EBCDIC değeri ascii2ebcdic tablosundan alınmıştır. STOSB komutu ise [DI]=AL işlemini yaparak hedef alanda EBCDIC kodlarından oluşan diziyi oluşturmaktadır.

LODSW Load String Word

DS:SI ikilisinin belirttiği adresten bir word'üı AX yazmacına yerleştirir. SI yazmacının değeri ise DF bayrağının durumuna bağlı olarak değiştirilir. LOADSW komutu bayraklar üzerinde bir değişikliğe neden olmaz.
Yapılan işlem:
AL=[DS:SI]
SI=SI-2;(DF=0 ise artma olacaktır)

LODSD Load String Doubleword

DS:SI ikilisinin belirttiği adresten bir word'üı EAX yazmacına yerleştirir. SI yazmacının değeri ise DF bayrağının durumuna bağlı olarak değiştirilir. LOADSD komutu bayraklar üzerinde bir değişikliğe neden olmaz.
Yapılan işlem:
AL=[DS:SI]
SI=SI-4;(DF=0 ise artma olacaktır)

STOSB Store String Byte

LODSB işleminin yaptığının tam tersini yapan bir işlemdir. AL yazmacındaki değeri ES:DI ikilisinin gösterdiği bellek alanına yerleştirir.DI yazmacının değeri DF bayrağının durumuna bağlı olarak değiştirilir.
Yapılan işlem:
[ES:DI]=AL
DI=DI-1 (DF=0 ise artan)

Örnek

```
MOV DI,0
MOV CX,100
XOR AL,AL
CLD
REP STOSB
```

Başlangıç adresi olarak 0 seçiliyor. İşlem tekrar sayısı CX yazmacındaki değer ile belirleniyor.
AL yazmacı sıfırlanıyor. DF=0 yapılarak DI yazmacının artarak ilerleyeceği belirleniyor. Böylece 0 adresinden itibaren 100 byte'lık bellek alanı sıfır ile doldurulуyor.

STOSW Store String Word

LODSW işleminin yaptığının tam tersini yapan bir işlemdir. AX yazmacındaki değeri ES:DI ikilisinin gösterdiği bellek alanına yerleştirir.DI yazmacının değeri DF bayrağının durumuna bağlı olarak değiştirilir.
Yapılan işlem:
[ES:DI]=AL
DI=DI-2 (DF=0 ise artan)

STOSD Store String Doubleword

LODSD işleminin yaptığının tam tersini yapan bir işlemdir. EAX yazmacındaki değeri ES:DI ikilisinin gösterdiği bellek alanına yerleştirir.DI yazmacının değeri DF bayrağının durumuna bağlı olarak değiştirilir.
Yapılan işlem:
[ES:DI]=AL
DI=DI-4 (DF=0 ise artan)

CBW Convert Byte to Word

İşleneni olmayan bu komut AL yazmacında bulunan verinin AX yazmacına yerleştirilmesini sağlar. Bu işlem yapılırken işaret biti yüksek anlamlı byte boyunca tekrarlanır.

CBW; AL=10001111B ise sonuç AX=1111111110001111B olur
AL=01111000B ise sonuç AX=0000000001111000B olur.

CWD Convert Word To Doubleword

İşleneni olmayan bu komut AX yazmacında bulunan bilginin DX ve AX yazmaç ikilisine yerleştirilmesini sağlar. Bu işlem yapılırken işaret biti yüksek anlamlı word boyunca tekrarlanır. Yani DX yazmacı işaret bitinin sahip olduğu değer ile doldurulur.
CWD; AX=1000000011111111B ise sonuç DX=1111111111111111B ve AX=1000000011111111B olacaktır.

CWDE Convert Word To Doubleword Extended

Bu komut ile AX yazmacındaki değerin işaret biti EAX yazmacının yüksek anlamlı wordü boyunca tekrarlanır.

CDQ Convert Doubleword to Quadword

İşaret biti kalmak kaydıyla doubleword bir sayıyı quadword sınırına getirmektedir. Bu komut yanına işlenen almaz, giriş verisi olarak EAX yazmacını kabul eder ve sonucu EDX:EAX ikilisinde oluşturur.

Ön Ekler

Ön ekler CMPS,LODS,MOVS,SCAS,STOS türü string komutlarının önünde, işlemlerin istenen sayıda veya gerek koşul sağlanıncaya kadar tekrarlanmasını sağlamak üzere kullanılırlar. İşlemin tekrar sayısı CX yazmacındaki değer yardımıyla belirlenir. Her işlemden sonra CX yazmacı otomatik olarak 1 azaltılacaktır. Özellikle tek komutun tekrarlandığı türde işlemlerde LOOP komutu kullanarak işlemleri yapmak yerine ön eklerden yararlanmak bir alışkanlıktır.

REP Repeat String Prefix

CX <>0 olduğu sürece REP komutunu takip eden işlem tekrarlanır. Diğer deyişle CX=0 olunca durur.

Örnek:

```
LEA DI,dizi
CLD
XOR AL,AL
MOV CX,1024
REP STOSB
```

Dizi isimli bellek alanının içeriğinin sıfırlanması için bellek adresi DI yazmacında tutulmaktadır. İşlemin artan adrese doğru yapılmadı için DF=0 yapılır. İşlem sayısı CX yazmacına aktarılır. STOSB işlemi yapıldıktan sonra REP öneki gereği CX azaltılır ve sıfırdan farklı olduğu sürece STOSB komutu tekrar işletilir.

REP Repeat String Prefix

Örnek:

```
LEA SI,dizi1
LEA DI,dizi2
CLD
MOV CX,8
REP MOVSB
```

Dizi1 ile belirlenen bellek alanından 8 byte veri dizi2 ile belirlenen alana aktarılmaktadır. Bu işlem MOVSB yerine MOVSW ile yapılacak olursa CX yazmacında 8 değeri kullanmak yerine 4 değeri kullanılmalıdır.

REPE/REPZ Repeat Equal/Repeat Zero

ZF bayrağının değeri 1 olduğu sürece işleme devam edilecektir. Diğer deyişle CX=0 veya ZF bayrağının değeri 0 olduğunda işlem sona erer.

Örnek:

```
XOR BL,BL
CLD
MOV CX,8
LEA SI,dizi1
LEA DI,dizi2
REPE CMPSB
JNE son
INC BL
....
```

Dizi1 ve dizi2 isimli bellek alanlarının içerikleri karşılaştırılmaktadır. CX=0 veya ZF=0 ise işlem biter. İşlemin hangi koşula bağlı olarak bittiği koşullu dallanma komutuyla belirlenir. İşlem bir eşitlik sonucu sona ermiş ise BL yazmacının değeri 1 artırılmaktadır.

REPNE/REPNZ Repeat not Equal/Repeat not Zero

ZF bayrağının değeri 0 olduğu sürece işleme devam edilecektir. Diğer deyişle CX=0 veya ZF bayrağının değeri 1 olduğunda işlem sona erer.

Örnek : Aşağıda verilen programda BASSON alt programı kaynak verisini önce B harfinden başlayarak Hedef değişkenine kopyalamaktadır. SONBAS alt programı ise R harfinden başlayarak kopyalamaktadır

```
. MODEL SMALL
.STACK 64
.DATA
KAYNAK DB 'BILGISAYAR'
HEDEF DB 'ELEKTRONIK'
HEDEF2 DB 10 DUP (' ')

.CODE
ANA PROC FAR
CALL BASSON
CALL SONBAS
MOV AH,4CH
INT  21H
ANA ENDP

BASSON   PROC NEAR
CLD    ; soldan sağa doğru
MOV CX,10
LEA SI, KAYNAK
LEA DI, KAYNAK
REP MOVSB
RET
BASSON   ENDP

SONBAS   PROC NEAR
STD   ; sağdan sola doğru
MOV CX,10
LEA SI,KAYNAK+10
LEA DI,HEDEF+10
REP MOVSB
RET
SONBAS   ENDP
END ANA
```

Örnek : Kopyalama

```
.MODEL SMALL
.CODE
ORG 100H
BAS: JMP SHORT KOPYALA
KAYNAK DB ‘BILGISAYARCILAR’
HEDEF DB ‘ELEKTRONIKCILER’
KOPYALA   PROC NEAR
CLD
MOV CX,15
LEA SI,KAYNAK
LEA DI,HEDEF
TEKRAR:
LODSB
STOSB
LOOP TEKRAR
MOV AH,4CH
INT 21H
KOPYALA   ENDP
END BAS
```

Örnek: Aşağıda verilen iki string eşit ise BH kaydedicisine 1 değil ise BH kaydedicisine 0 atan programı yapınız.

```
NAME1 DB ‘ASSEMBLERS’
NAME2 DB ‘ASSEMBLERS’

CLD
MOV CX,10
LEA SI,NAME1
LEA DI,NAME2
REPE CMPSB     ;  Eşit olduğu müddetçe karşılaştır.
JNE ESITDEGIL
MOV BH,01
JMP SON
ESITDEGIL:
MOV BH,0
SON:
 MOV AH,4CH
INT 21H
```

Örnek program verilen ‘LDA#305A’ Stringindeki elemanlar arasında ‘#’ karakterini arayıp, yerine ‘$’ karakteri ile değiştirir.

```
.MODEL SMALL
.STACK 64
.DATA
DIZI  DB ‘LDA #305A’
.CODE
ANA PROC FAR
MOV AX,@DATA
MOV ES,AX
CLD
MOV AL,’#’
MOV BH,’$’
MOV CX,9
LEA ES:DI,DIZI
REPNE SCASB
JNE CIK
MOV BYTE PTR [DI-1], BH
CIK:
MOV AH,4CH
INT 21H
ANA ENDP
END ANA
```

Yığın Komutarı

POP

POPA

POPAD

POPF

POPFD

PUSH

PUSHA

PUSAD

PUSHF

PUSHFD

Yığın Komutları

Yığın çağırılan yordamların dönüş adreslerini, yazmaçların değerlerini ve yordamlar arasında aktarılan parametreleri saklamak amacıyla kullanılmaktadır. 32 bitlik mimari kullanılmadığı sürece yığın yapısı üzerinde çalışan komutlar her seferinde yığının üstünden bir word alır veya koyarlar. 80386'dan itibaren 32 bitlik işlemcilerin devreye girmesi ile yığın işlemleri 32 bit olarak yapılabilmektedir. 8086 işlemcisinde SP yazmacı yığın üzerinde bulunan dolu gözü göstermektedir.

POP src src←SS:[SP] SP ←SP+2

PUSH src SP ←SP-2 SS:[SP] ←src

Bu ara işlemler PUSH ve POP komutlarının kullanımı ile otomatik olarak gerçekleştiğinden dolayı kullanıcı müdahalesi gerekmez. POP ve PUSH komutları bayraklar üzerinde değişiklik yapmazlar. Ancak bayrak değerlerini yığından çeken POPF komutu doğası gereği bayrakları değiştirecektir.

POP Pop word of stack

POP regw
POP mem
Yığın üzerinden alınan veriyi hedef olarak belirlenen işlenene aktarıp yığın işaretçisi (SP yazmacının değeri) artırılacaktır.

Yapılan işlem:
Hedef=SS:[ESP]
İf(sizeof(hedef)=16) then
ESP=ESP+2
Else
ESP=ESP+4
Endif

Örnek:
POP ECX;Yığından 32 bitlik değer ECX yazmacına alınır. ESP=ESP+4 olur (32 bit işlemciler)
POP AX ;Yığından 16 bit değer AX yazmacına alınır. SP=SP+2 olur.

POPA Pop all general registers

SP yazmacı haricindeki tüm genel amaçlı yazmaçların yığından alınmasını sağlar. 80286 ve üstü işlemcilerde kullanılabilir.

POPA yapıldığında aşağıdaki işlemler gerçekleşir:

```
POP DI
POP SI
POP BP
ADD SP,2; SP yazmacının değerinin değişmesini engellemek için
POP BX
POP DX
POP CX
POP AX
```

POPAD Pop all general registers-double

ESP yazmacı hariç 32 bitlik tüm genel amaçlı yazmaçların yığından alınmasını sağlar.

POPADyapıldığında aşağıdaki işlemler gerçekleşir:

```
POP EDI
POP ESI
POP EBP
ADD ESP,4; SP yazmacının değerinin değişmesini engellemek için
POP EBX
POP EDX
POP ECX
POP EAX
```

POPF Pop flag from stack

Bu komut ile bayrak (PSW) yazmacına yığından 16 bitlik değer yüklenir.SP yazmacının değeri 2 artırılır.

Yapılan işlem:

FLAG=SS:[SP]

SP=SP+2

POPFD Pop eflag from stack-double

EFLAG yazmacına yığından alınan 32 bitlik değer yüklenirESP yazmacının değeri 4 artırılır.
Yapılan işlem:
EFLAG=SS:[SP]
ESP=ESP+4

PUSH Push Word To The Stack

PUSH idata
PUSH register
PUSH memory
PUSH sreg

Yığını üzerine konulması istenen veriyi yerleştirir. Daha sonra yığın işaretçisi uygun şekilde azaltılır.
Yapılan işlem:
İf(sizeof(src)=16) then
ESP=ESP-2
Else
ESP=ESP-4
Endif
SS:[ESP]=src

Örnek
PUSH EAX ;Yığına EAX yazmacındaki 32 bit değer konulur.
PUSH AX; Yığına AX yazmacındaki 16 bit değer konulur.

Örnek

```
MOV AX, 1234h
PUSH AX
POP DX ; DX = 1234h
RET
```

PUSHA Push all general registers

PUSHA 16 bitlik tüm yazmaç değerlerinin belli bir sıraya uygun olarak yığına saklanmasını sağlar. Ancak yazmaçlar arasında yığın üzerinde işaretçi olarak kullanılan SP yazmacının durumu farklılık göstermektedir. Gerçekleştirilen her PUSH işleminin sonucu olarak SP yazmacının değeri 2 azalmaktadır. Bu durumda SP yazmacının yığına saklanması sırasında farklı bir yöntemin kullanılması gerekmektedir. Bunun için PUSA işlemine başlarken SP nin sahip olduğu değer geçici bir değişkende saklanır ve SP yazmacının yığında saklanma sırası geldiğinde bu geçici değer yığına SP'nin yerine yerleştirilir. Yazmaçların yığına yerleştirilme sırası şöyledir:

Temp=SP
PUSH AX
PUSH CX
PUSH DX
PUSH BX
PUSH Temp
PUSH BP
PUSH SI
PUSH DI

PUSHAD Push all general registers-double

32 bitlik tüm yazmaç değerleri aşağıda verilen sıra dahilinde yığına yerleştirilir. Ancak ESP yazmacını değeri bu aşamada yığına yerleştirilmez.
Yapılan işlem:
Temp=ESP
PUSH EAX
PUSH ECX
PUSH EDX
PUSH EBX
PUSH Temp
PUSH EBP
PUSH ESI
PUSH EDI

PUSHF Push Flag On To Stack

Bu komut ile 16 bit ile ifade edilen bayrak değeri yığına yerleştirilir.
Yapılan işlem:
SP=SP-2
SS:[SP]=FLAG

PUSHFD Push EFlag On To Stack

Bu komut ile 32 bit ile ifade edilen bayrak değeri yığına yerleştirilir.
Yapılan işlem:
ESP=SP-4
SS:[ESP]=EFLAG

Ödev

10 elemanı olan, word olarak tanımlı bir dizi içinde kaç tane tek, kaç tane çift sayı olduğunu bulan program