

ASSEMBLY DİLİ

ASSEMBLY DİLİ NEDİR?

Pek çok kiři assembly ile makine dilini eşdeęer olarak kabul eder. Makine dili temelde assembly ile oldukça benzerlik gösterir, ancak aynı şeyler değildir. Makine dili, tamamen 16'lık (hexadecimal) sistemde oluşturulmuş komut serilerinden oluşmaktadır. Oysa assembly bu komut serilerini bire bir karşılık gelen kelimelerden ve kelimelerin kısaltmalarından oluşmuştur.

Örnek:Makine dili ile B8 01 00 şeklinde belirtilen bir komut serisi assembly dilinde MOV AX,0001 komutuna karşılık gelir.

Şimdi de assembly ve assembler sözcüklerinden bahsedeyim; Yukarıda assembly'nin ne olduğunu makine dili ile arasındaki farktan bahsederek anlatmaya çalışmışım. Yani assembly de basic, pascal yada c dilleri gibi kelimelerden oluşmaktadır. Neticede bu komutları makina diline dönüştürecek yazılımlara ihtiyaç vardır ki bunlara derleyici (compiler) diyoruz. İşte hazırlanmış olan assembly komutlarını makine diline dönüştüren programlara assembler denilmektedir.Dięer bir söyleyişle; assembly derleyicisine assembler denir.

ASSEMBLY DİLİNİN MAHZURLARI

Program yazmak için üzerinde çalışılan bilgiwayarın donanımı hakkında detaylı bilgi sahibi olunmalıdır. (Mikroişlemcide blunan kagydediciler ve sayısı,kullanığı komut kümesi ve adreslem türleri gibi) ayrıca elastiki olmamasıdır, firmalarca üretilen her mikroişlemcinin kendisine has programlama dili vardır. Bundan dolayı, bir için yazılan porgram gięer bir mikroişlemcide çalışmayabilir. Assemblinin dięer yüksek düzeyli dillere göre yazılması zordur, dięer dillerde komutlar konuşma dilinde kullanılan anlaşılabilir kelimelerden oluştuęundan, yazılması ve yorumlanması kolaydır.

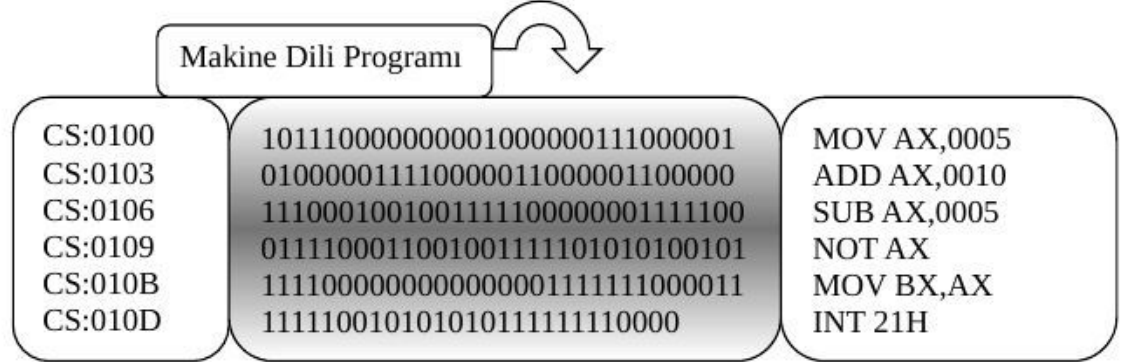
ASSEMBLY DİLİNİN AVANTAJLARI

Programcıya hızlı program çalıştırma, dar bellek alanınada çalışma yüksek düzeyli dillerde olmayan yetenekler. Yüksek düzeyli programlar yazma imkanı sağlar.

- Bilgisayar üzerindeki özel donanım parçaları üzerinde çalışırken, iyi bir denetim imkanı verir.
- Küçük ve oturaklı çalışabilir programlar yazılabilir.
- Diğer yüksek düzeyli dillerde yazılanlara göre daha hızlı çalışır.

MAKİNA DİLİ

Mikroişlemcili sistemlerde mantıksal 0 ve 1 oluşan ve belli bir görevi yerine getirmek üzere yazılmış dile makine dili denir. Makina dilini meydana getiren bu sayıların yan yana gelmesi, programcı için anlaşılmayan bir dizi oluşturur. Bir bilgisayar için yazılan makine dilindeki program tek şartla diğer makineler için geçerli olabilir aksi halde sadece kendi bilgisayarında çalışır. Eğer makinenin kullandığı işlemci aynı komut kümesini kullanıyorsa biri diğerinde çalışabilir.



Şekil-1. Hexsadesimal kodlanmış bir makine dilindeki bir program

Buradaki ikili komutlar makinenin anlayacağı ve çalışacağı gerçek dili gösterir. Her işlemcinin kendine has dili vardır. İkili dizilerden oluşan 0 ve 1 leri bilgisayarın belleğine girmek, hatalar doğurur. Diller, makine ile insan arasında anlaşılabilmesi açısından çeşitli katagorilere ayrılırlar:

- Üst düzey diller
- Orta düzey diller
- Alt düzey diller

Bir bilgisayar için yazılan makine dilindeki program, tek şartla diğer makineler için geçerli olabilir, aksi halde sadece kendi bilgisayarında çalışır. Makinenin kullandığı işlemci aynı komut kümesini kullanıyorsa, bir diğerinde çalışabilir.

ÜST DÜZEYLİ DİLLER

kolay yazılabilir. Bu dillere örnek : Basic, Pascal, C dir.

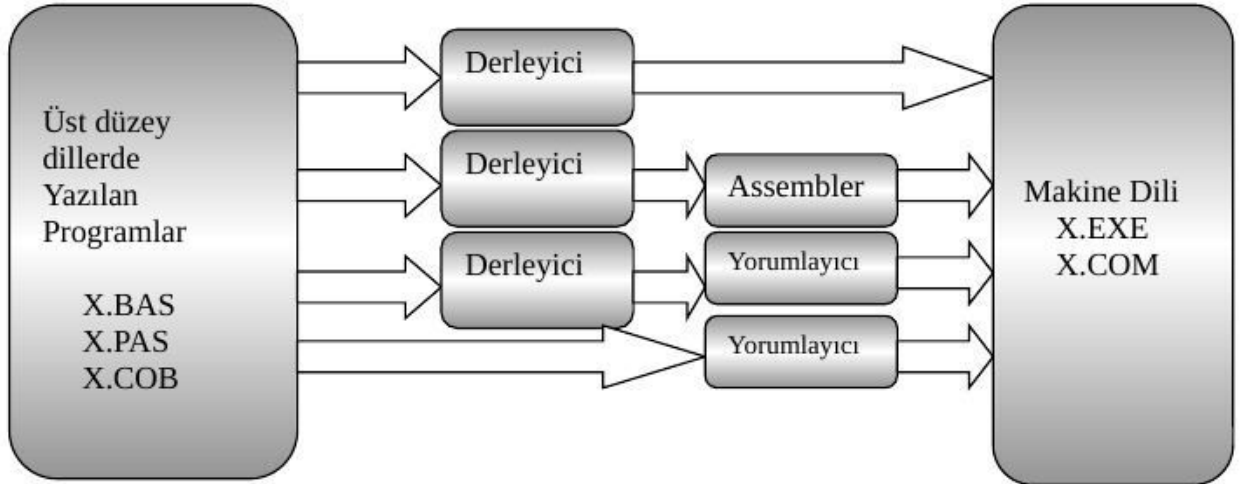
ÇEVİRME İŞLEMLERİ

Kaynak programın, bir çeşit aracı program kullanarak amaç programı tercüme edilmesi işlemine denir. Yazılan programlar makine diline çevrilirken farklı tipte çevirme işlemine tabi tutulur. Bu işlemi yapan programlar yorumlayıcı ve derleyicidir.

DERLEYİCİLER (Compiler)

Üst düzey dillerde yazılan programları makine diline, Assembly diline veya bazı ara dillere çeviren aracı programdır. Eğer kullanılan derleyici ilkin Assembly dilinde bir program üretiyorsa, daha sonra Assembly dili, makine dilinde program üretiminde kullanılmalıdır.

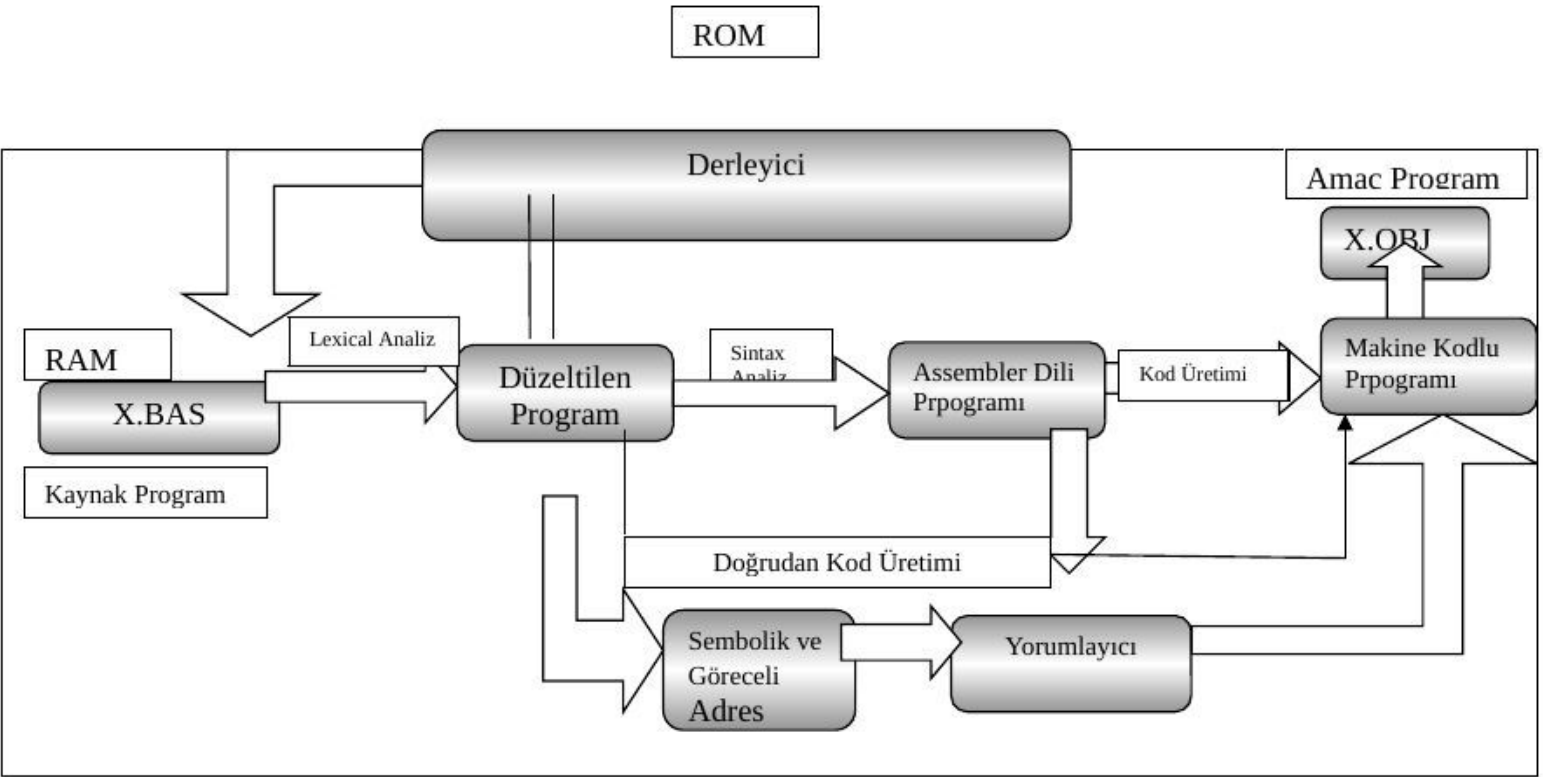
Derleyici, ROM'da veya manyetik ortamda bulunan programlar veya alt yordamlardan oluşur. Derleyici programı tarayarak, ilk görevi olan anahtar kelimeleri ve oparetörleri tespit eder. Konu ile ilgisi olmayan tüm detaylar ve boşluklar yok edilir. Kalan kısım, özel işaretlerle ifadelere bölünmüş özlü bir program haline alır.



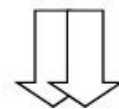
Şekil-2. Derleyici ve Yorumlayıcı ayrı veya Birlikte Çalışabilir

- Anahtar kelime veya operatörleri tespit etmek için kelime analizi (lexical analysis)

- Her ifadenin tipini ve düzetmelerini tespit etmek için söz dizimi (Syntactic analysis)
- Hata kontrolü, kaydedici ataması ve optimizasyona yardımcı olacak ifadeler arasındaki ilişkiyi bulan akış analizi
- Toplam komut sayısını azaltmak için optimizasyon
- Assembly veya makine dili için kod üretilmesi
- Program listesinin üretilmesi

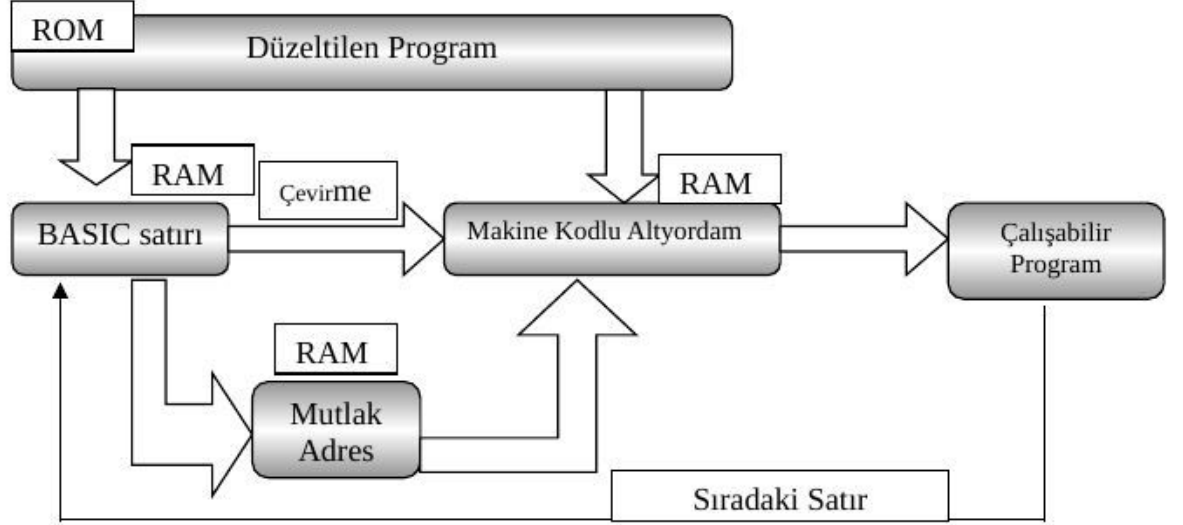


Programdaki her satırın formatı, satır numarası komut ve işlenenden oluşur. Bu satırlar kontrol edilerek, her hangi bir hata varsa hata kodu ekranda görüntülenir. Programdaki değişkenler, Assembly dilinde daha sonra değişken olarak listelenirler. Her değişkenlerin sembolik adresleri ve uzunlukları verilir. Her talimat veriler için ayrılmış RAM alanında adres olarak atanır ve derlenen listenin sembolik adresi, sembol tipi ve mutlak adresi RAM de depolanır. Bu listeye genel olarak Dictionary denir.



YORUMLAYICILAR(Interpreter)

Bir BASIC programındaki satırları, bir anda bir satır işlemiyle yorumlanırlar. Yani, satırlar tek tek yorumlanarak amaç programa erişilir. Bir satır başarılmadan, sonraki satıra geçilmez. Yorumlayıcı, değişkenler kütüphanesini ve onların adreslerini derler ve çevrilmiş BASIC satırının gerekli olduğu yerlere mutlak adresi sıkıştırır.



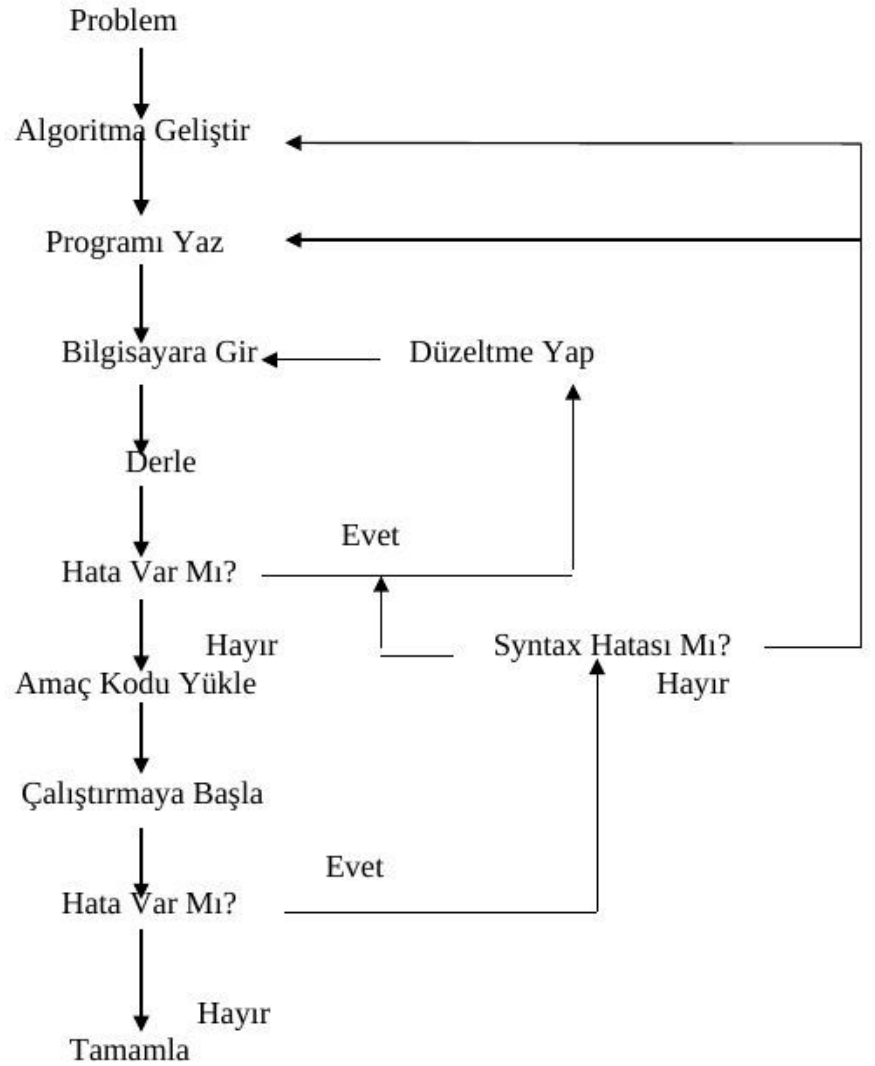
Şekil-4. Basit Bir Yorumlayıcının Yapısı

Hata ayıklamalarında yorumlayıcı derleyiciden daha az başarılıdır. Yorumlayıcı her ifadeyi bellekte yerini alacak makine dili kodlarına çeviremez ve bundan dolayı da amaç program üretmez. BASIC satırının çevrilmesi sırasında, üretilen makine kodu silinirken satır ifadeleri işletilir.

Yorumlayıcıda bir satırdaki hata düzeltilmeden diğer satıra geçilmez.



A) Yorumlanmış Program Akışı



B) Derlenmiş Program Akışı

Şekil-5. Derleyici ve Yorumlayıcı Arasındaki Çalışma Farkını gösteren Program Örneği

ASSEMBLY DİLİ ÇEŞİTLERİ

Assembly dilinde yazılmış kaynak programı, amaç program olan makine diline çeviren aracı programa Assembler. Bu ROM da olabileceği gibi, Disk veya Disketten çalıştırılabilir. ROM dakiler Yerleşik Assemblerlerdir.

1-)CROSS-ASSEMBLY: Herhangi bir bilgisayarda üretilen Assembleri başka bir bilgisayarda çalıştırmak için kullanılır.

2-)SELF-ASSEMBLY: Belek ve çevre cihazı gerektirir. Oldukça yavaş çalışır.

3-)META- ASSEMBLY: Bir çok komut kümesini farklı şekilde ele alıp kullanabilir.

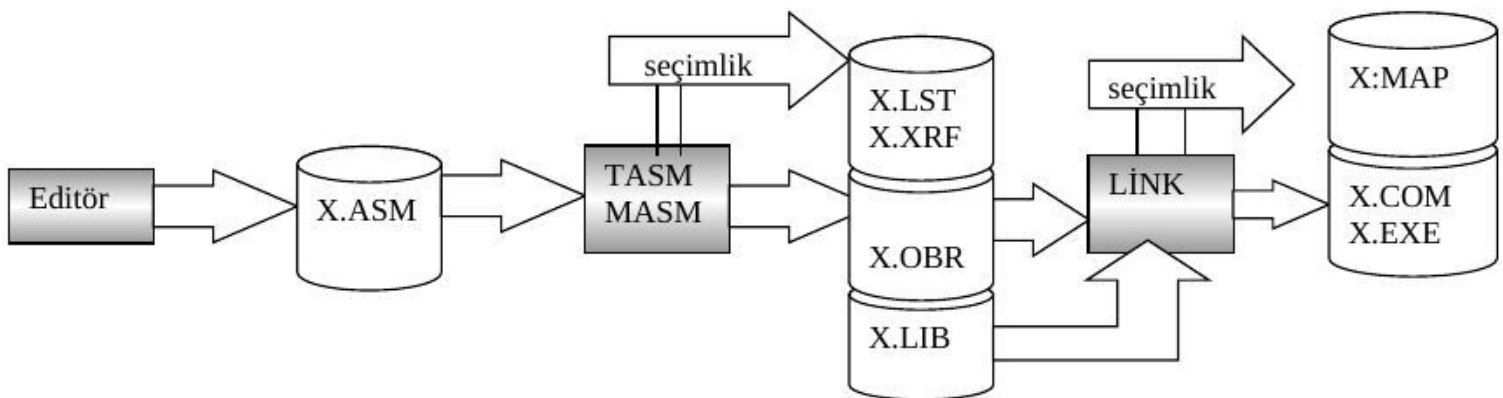
4-)TEK-GEÇİŞLİ ASSEMBLY: Programın bir defa taranarak amaç programa çevrilir.

5-)İKİ-GEÇİŞLİ ASSEMBLY: Kaynak program iki defaele alınır. İlk taramada bütün sembolleri toplar, tanımlar. İkincisinde ise Referansları gerçek tanımlarıyla yerine koyar.Yedekleme yoksa oldukça yavaş olabilir. Çoğu microişlemciye bağlı Assembler, Two-Pass Assemblere gerek duyar.

- Mini-Assembler programı eğer ROM veya benzeri bir yerde yerleşik değilse, bilgisayarın RAM belleğine yüklenir.
- Bilgisayara mini assembler programının çalıştırılması için komut verilir.
- Bilgisayar kaynak kodunu makine kodna çevirerek RAM belleğe yerleştirir.
- Bir defa amaç kodu belleğe yerleştirildikten sonra, daha sonra kullanılmak üzere manyetik ortam olan diske veya teybe saklanır.

ÇEVİRİCİLER

Assembly yazım kollarlarına uyarak program yazılır ve X.ASM olarak kaydedilir ve Assembler vasıtasıyla amaç programa çevrilir. Amaç program, obj program modülünün başına eklenen program başlığını (header) oluşturur. Com tipi programda 256 baytlık psp ve exe tipi programda 512 baytlık program header, tamamlanmış adresler hakkında bilgi sağlar.



Şekil-6.Assemble, Link ve İcra Adımları

Programın çalıştırılabilir hale gelmesinde, MASM veya TASM assembler programları kullanılır.

Komut satırındaki X.ASM den sonraki virgüller, LST ve XRF dosyalarının oluşturulması içindir. XRF (CRF) uzantılı program, büyük programlarda hangi komutun hangi veriye karşılık geldiğini görmek için, LST ise, CRF tarafından programdaki ifade sayılarının üretildiği dosyadır.

PROGRAM YÜKLEYİCİLER

Çalıştırılabilir programın işletilebilmesi için işlemcinin kurulumunu gerçekleştirir. Yükleyici Programın resetlenmesi için donanım hizmeti verilmesinde sağlar. Microişlemci mandallanır veya kilitlenirse, programcı reset düğmesini kullanarak sistemi yeniden başlatabilir. EXE tipinde olan çalıştırılabilir programın, çalıştırılmak üzere belleğe yüklenmesindeki adımlar:

- Diskteki EXE programa erişim
- Bellek üzerinde 512 baytlık file header (COM tipi programlarda 256 baytlık PSP) sınırlarının oluşturulması
- File header için ayrılan yer dışındaki lara programın yerleştirilmesi
- File header adresinin DS ve ES veri segment kaydedicilerine yüklenmesi
- Kod segment adreslerinin CS kaydedicisine yüklenmesi ve IP kaydedicisinin kod segmentindeki ilk komutun ofet adresine (genelde 0000H) konulanması
- Yığın segmentinin SS kaydedicisine yüklenmesi ve SP kaydedicisinin programda belirtilen alana göre kurulması
- Programın çalıştırılması için, denetimin kod segmentteki ilk komuta aktarılması

Yükleyiciler; CS:IP ve SS:SP ikililerini oluşturarak programın ilk komuttan itibaren çalıştırılmasını sağlarlar. File header adresinin yüklenmesinde DS ve ES kaydedicilerinin ikisine birden, string işlemleri dışında gerek yoktur. Normal programda veri segmenti olarak DS nin yüklenmesi yeterlidir.

HATA AYIKLAYICILAR

Debugger bir hata ayıklama programı olup, programın çalışması sırasında, istenen noktadaki kaydedici ve bellek durumlarının incelenmesini sağlar. Eğer programda hata bulursa, bazı Debugger programları, kullanıcıya hataları düzeltme ve programı kaldığı yerden çalıştırmaya izin verir. Bu özellik, kaydedicilerin kapsamalarının incelenmesi ve içeriklerinin değiştirilmesini sağlar.

İşlemler, Debugger tarafından mikroişlemcideki komutların icrasıyla veya bir Simülatör kontrolü veya bir Emülatör altında icra edilmesiyle yapılır.

BAYRAKLAR

FLAGLAR	BIT SET (1)	BIT RESET (0)
Overflow	OV	NV
Direction	DN	UP
Interrupt	EI	DI
Sign	NG	PL
	ZR	NZ
Zero		NA
	AC	PO
Auxiliary Carry		NC
	PE	
Parity		
	CY	
Trap		
Carry		

Overflow (Taşma) = İşaretli sayılarla işlem yapılırken bir hatanın ortaya çıkması durumunda devreye girer.

Direction (Yön) = Harf , dizi (string) işlemlerde indis kaydedicisininileri yada geri hareket etmesini sağlar.

Interrupt (Kesme) = Sisteme bağlı harici cihazlardan gelen kesme işlemlerini değerlendirir.

Sign (İşaret) = İşaretli sayılarda yapılan işlemlerde bu bayrak devreye girer.Sonuç eksi (-) değerliyse bayrak 1 , artı (+) değerliyse bayrak 0 olur.

Zero (Sıfır) = İşlem sonucu 0 ise bayrak 1 değilse 0 olur.

Auxiliary Carry (Yardımcı elde) = Elde bayrağı ile aynı işi yapar fakat 3. bitten bir fazlalık ortaya çıkarsa bayrak 1 aksi takdirde 0 olur.

Parity (Eşlik Biti) = İşlem sonucunda kaydedicideki mantıksal birlerin sayısı çift ise bayrak 1 değilse 0 olur.

Trap (Tuzak) = Hata ayıklama işleminde verilerin adım adım işlenmesini sağlar.

Carry (Elde) = Toplama işlemi sonunda elde veya çıkarma işleminde borç ortaya çıkıyorsa bayrak 1 aksi takdirde 0 olur.

VERİ TIPLERİ

Bir programlama dilinin temel özelliklerinden birisi dilin içinde kullanılacak olan veri tipleridir. Burada assembler dilinde kullanılan en basit veri tiplerinden bahsedilecektir.

Assembler programlama dilinde kullanılacak olan bilgilerin uzunluğu bilgisayarın mikroişlemcisine bağlıdır. 8088 mikroişlemci için en fazla 16 bit uzunluğunda bir veri temsil edilebilir. 80286, 80386 ve 80486 mikroişlemcileri daha g elişmiş oldukları için kullandıkları bilgi uzunluğu daha fazla olabilir.

BYTE

Byte 8 bitten oluşan bir bilgi birimidir. Assembler¹da byte ile hem nümerik hemde alfanümerik bilgileri temsil etmek mümkündür.

Gerek BYTE gerekse diğer veri tiplerinde sayılar işaretli ve işaretsiz olarak iki ana guruba ayrılırlar. İşaretsiz sayılar tüm veri alanını bir bütün olarak değerlendirildiği bir durumu ifade eder. Sayıların işaretli veya işaretsiz olması makine açısından izafi bir durumdur.

1 0 0 0 0 0 1 İşaretsiz 129 sayısını temsil eder.

1 0 0 0 0 0 1 İşaretli -1 sayısını temsil eder.

Negatif sayılarda en duyarlı bit 1 olur.

Byte iki ana kısımdan oluşur. Her kısma NIBBLE adı verilir. Her NIBBLE bir hexadecimal sayıyı temsil eder.

1 0 0 0 0 0 1

2.NIBBLE 1.NIBBLE

WORD

2 Adet Byte ile oluşturulmuş bir yapıdır. 8088 ve diğer mikroişlemciler 16 bit uzunluğundaki bu yapıyı desteklerler. Word yapısını bir bütün olarak işleyebileceğiniz gibi yüksek byte (HB) veya Low byte (LB) olarak da işleyebilirsiniz.

DOUBLEWORD

İki word uzunluğunda yani 32 bit uzunluğunda bir yapıdır. Gerek adreslemede gerekse büyük sayıların saklanması için kullanılır.

QUADWORD

64 Bit uzunluğunda bir yapıdır. 4 word^{Fuz}un birleştirilmesi ile oluşturulur.

8088 Mikroişlemcisi olan bir makinada direkt mikroişlemcinin desteklediği BYTE ve WORD kavramları kullanılabilir. 80286, 80386 ve üzeri mikroişlemcilerde BYTE ve WORD kavramlarının dışında DOUBLEWORD ve QUADWORD kavramları da kullanılabilir. 8088 mikroişlemcilerde WORD^{Fuz}den daha uzun sabit ve değişkenler de kullanılabilir. Ancak bu kullanım direkt mikroişlemci desteği dışında yazılım desteği ile olur.

YAZMAÇLAR

8 bitlik yazmaçlar: Ah, Al, Bh, Bl, Ch, Cl, Dh, Dl

16 bitlik yazmaçlar: Ax, Bx, Cx, Dx, Si, Di, Bp, Cs, Ds, Es, Ss, Sp ve Flags (İşaretler)

Bu yazmaçları şu biçimde gruplandırabiliriz:

Ax : Aritmetik amaçlı yazmaç Ah ve Al yazmaçlarının birleşmesinden oluşur.

Bx : Taban yazmacı, Bh ve Bl yazmaçlarının birleşmesinden oluşur.

Cx : Sayaç yazmacı, Ch ve Cl yazmaçlarının birleşmesinden oluşur.

Dx : Veri yazmacı, Dh ve Dl yazmaçlarının birleşmesinden oluşur.

Si : Kaynak indeks yazmacı, bir kaynak dizinin elemanlarını gösterir.

Di : Hedef indeks yazmacı, bir hedef dizinin elemanlarını gösterir.

Bp : Taban göstergesi, genelde bir dizinin taban adresini göstermek için kullanılır.

Cs : Kod segmentini gösteren yazmaçtır.

Ds : Veri segmentini gösteren yazmaçtır.

Es : Ekstra segmenti gösteren yazmaçtır.

Ss : Yığın segmentini gösteren yazmaçtır.

Sp : Yığın göstergesi, yığının aktif noktasını gösterir.

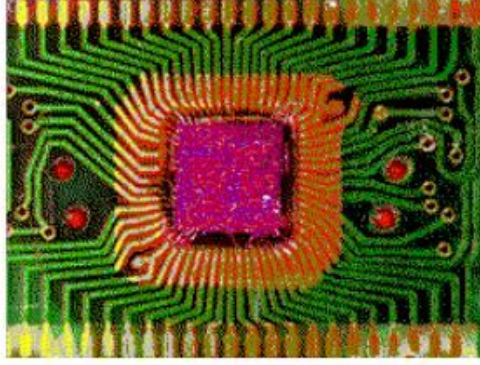
BASİT MİKROİŞLEMCİLER

2. BÖLÜM

BASİT MİKROİŞLEMÇİ YAPISI

Mikro işlemci, bellek ve I/O elemanlarının birleşiminden meydana gelen, mikro işlemciye dayalı bilgisayar sistemlerine mikrobilgisayar denir. Bilgisayar mimarisi, bilgisayar sistemine uygulanmış mantıksal yapı ve sistemi oluşturan donanım elemanlarının organizasyonunu ifade eder.

Mikroişlemci, bir mikro bilgisayarın merkezi işlem birimi gibi çalışır. İşlemcinin fonksiyonu, program komut kodlarının bellekten alınıp getirilmesi, kodunun çözülmesi ve çalıştırılması ve giriş-çıkış işlemlerinden kullanılan kontrol sinyallerinin üretilmesi ve senkronizasyon sağlanmasıyla, somunların gözlenmesi işlemleridir. Bu mikro işlemcinin fonksiyonel konfigürasyonuna kaydediciler, işlemlerde ardışıklığı sağlamak için frekans üretici, bölücü ve sayıcı gibi zamanlama ve kontrol elemanları dahildir.



Bilgisayarın kalbi veya beyni olarak adlandırılan mikro işlemciler, aynı zamanda Merkezi İşlem Birimi(CPU) olarak bilinirler. CPU genel olarak aşağıdaki işlemleri yapar:

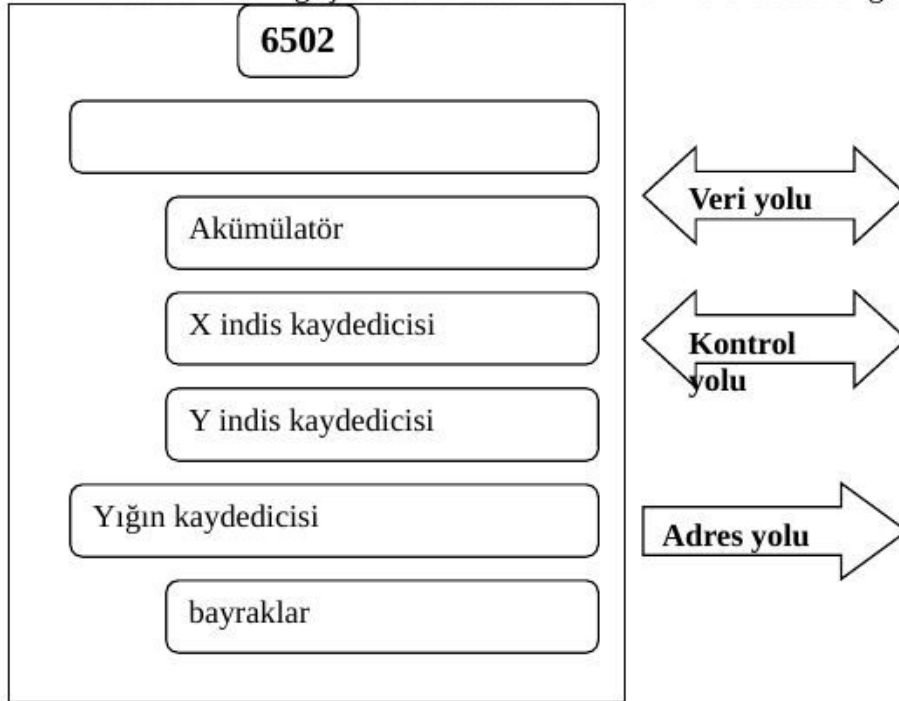
- Sistemdeki tüm elemanlar ve birimlere zamanlama ve kontrol sinyali sağlar.
- Bellekten komut alıp getirir.
- Komutun kodunu çözer.
- Komutun operandına göre, veriyi kendisine veya G/Ç birimine aktarır.
- Aritmetiksel ve mantıksal işlemleri yürütür.
- Program işlenirken, diğer donanım birimlerinden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

Bir mikro işlemcinin mimari yapısı en basit şekilde ifade edilmek istenirse, bir grup kaydedici, Aritmetik ve Mantık birimi ve birde sistemin ne zaman hangi işi yapacağını denetleyen zamanlama ve kontrol biriminden meydana gelmektedir.

MİKROİŞMLEMCİNİN TEMEL YAPITAŞINI OLUŞTURAN BİRİMLER

1-)KAYDEDİCİLER

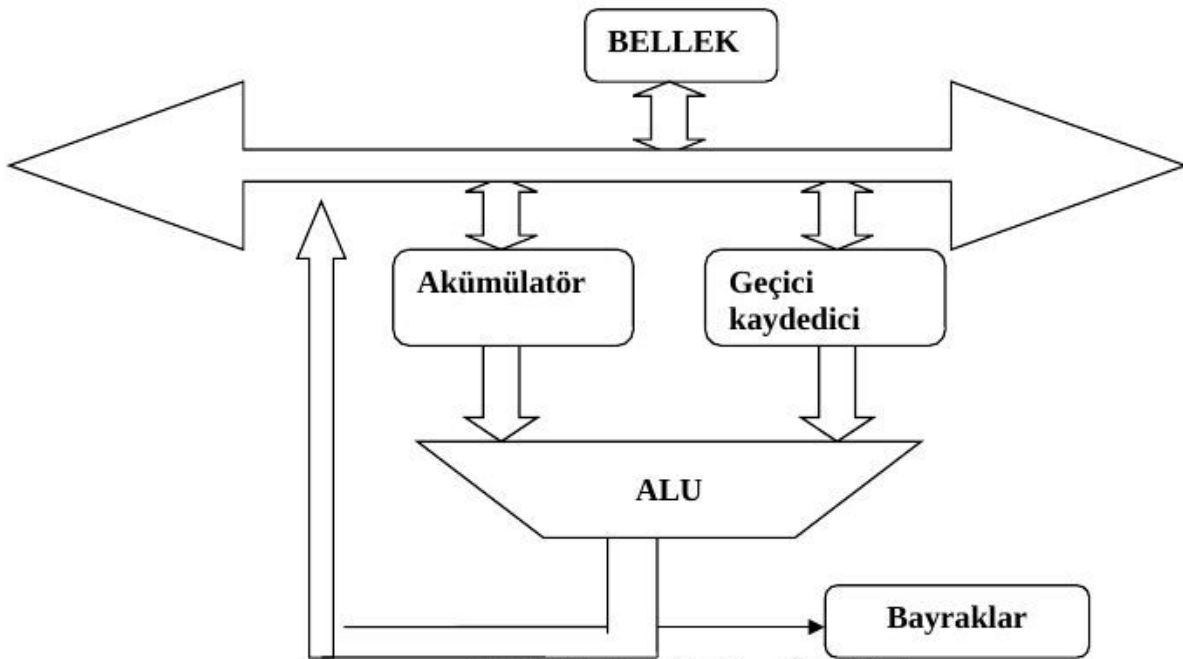
Mikroişlemcinin mimarisini belirleyen elemanlarından birisidir. Verinin manevrasında ve geçici olarak tutulmasında birinci dereceden görevlidir.



Şekil-1. Basit bir 8 bitlik işlemci kaydedici grubudur.

2-) ARİTMETİK VE MANTIK BİRİMİ(ALU)

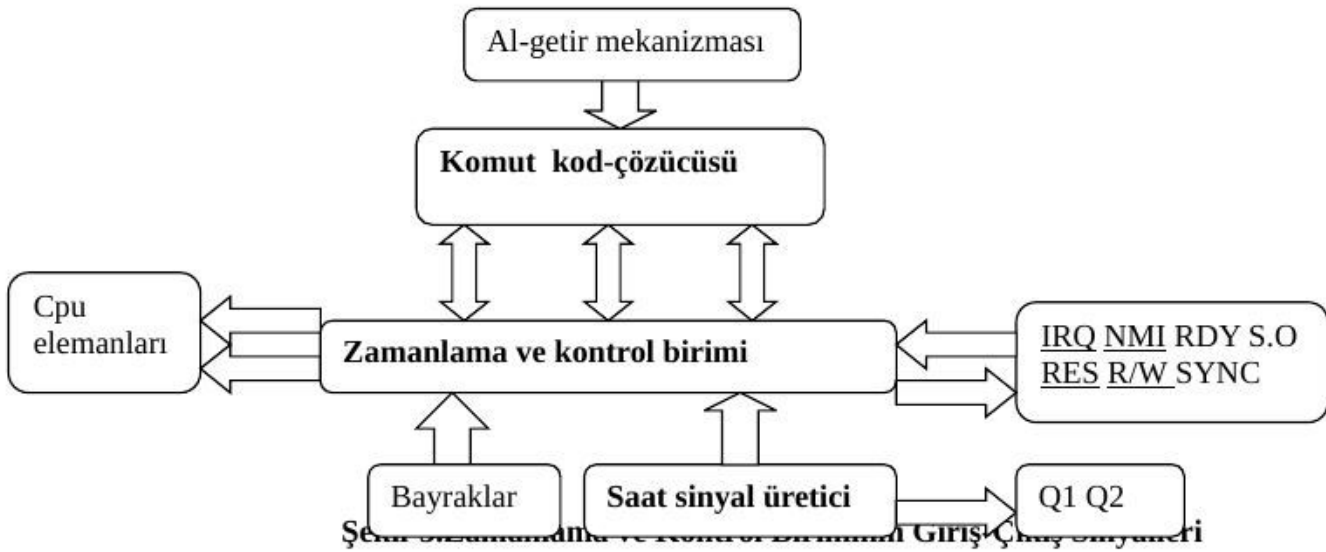
Alu 'da gerçekleşen bütün işlemler kontrol sinyalleri vasıtasıyla ve zamanlama ve kontrol biriminin gözetiminde eş anlamlıdır.



Şekil-2.Aritmetik-Mantık Birimi

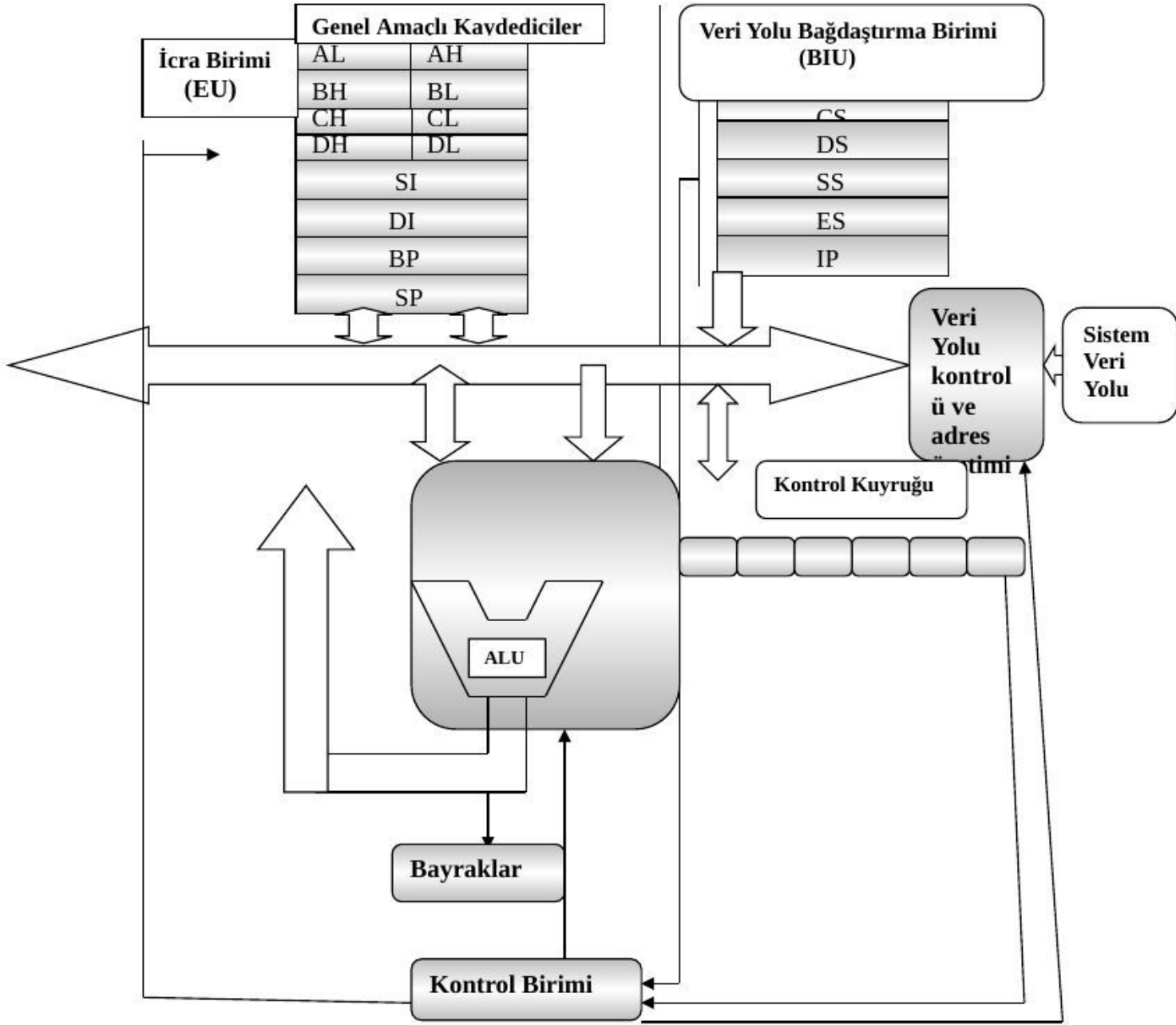
3-)ZAMANLAMA VE KONTROL BİRİMİ

Sistemin tüm işleyişinden ve işlemin zamanında yapılmasından sorumludur. Bellekte program bölümünde bulunan komut kodunun alınıp getirilmesi, kodunun çözülmesi, ALU tarafından işlenmesi ve sonucun alınıp belleğe geri konulması için gerekli olan kontrol sinyallerini üretir. Bilgisayar sisteminde bulunan dahili ve harici bütün elemanlar, bu kontrol sinyalleri ile denetlenir.



16 BİTLİK GELİŞMİŞ İŞLEMCİLER

Mimari yapısı çoklu görev ortamına uygun hale getirilmiştir. 16 bitlik x86 tabanlı işlemciler Veri Yolu bağdaştırma Birimi (BIU) ve İcra Birimi (EU) olmak üzere iki ana bölümde toplanır. BIU bölümüne segment kaydedicileriyle birlikte IP ve komut kuyukları ve veri alıp getirme birimleri dahildir.



Şekil-4. Basit 16-Bit Mikroişlemci Mimarisi

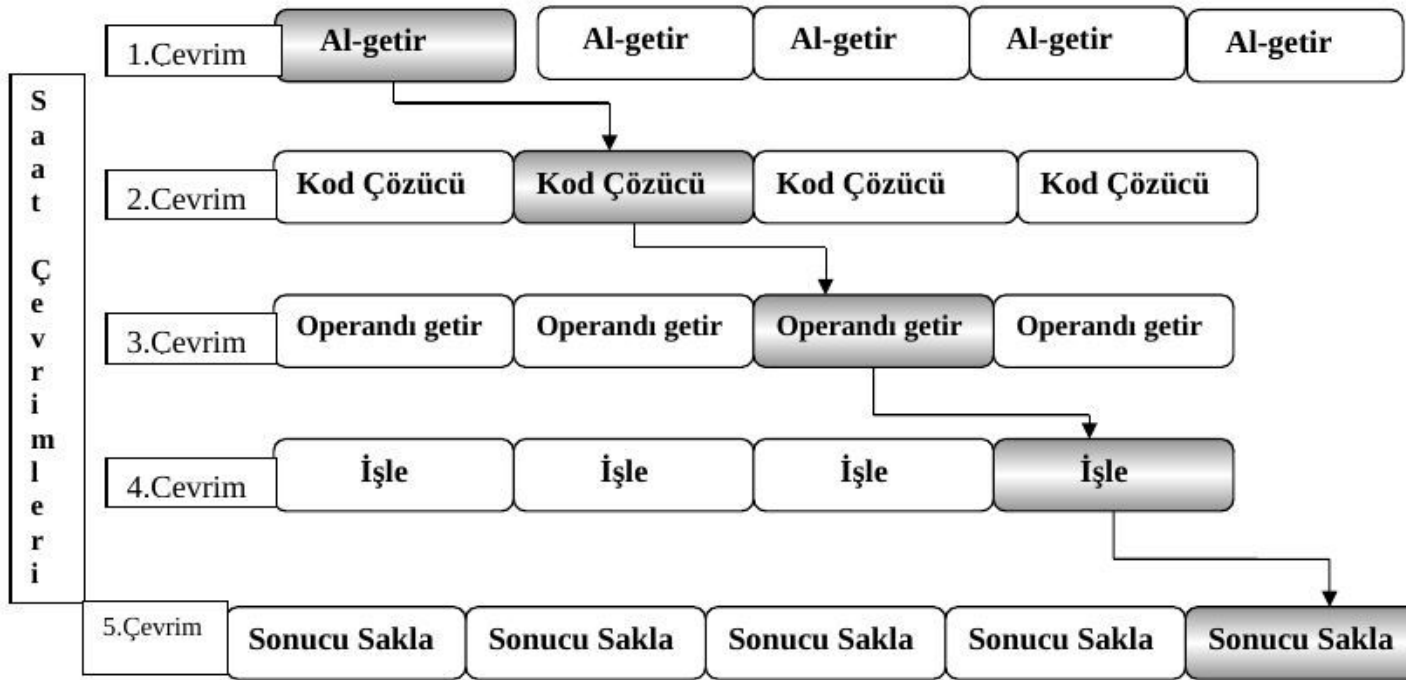
VERİ YOLU BAĞDAŞTIRMA BİRİMİ

Bellekten veya I/O birimlerinden, komut kuyruğuna getirilen komut kodları ve operand bilgileri, icra birimi tarafından işlenmek için hazır durumdadır.

İCRA BİRİMİ (EU)

BIU ile birlikte çalışır. Komutların işlenmesi sırasında her hangi bir veriye gerek duyulursa ve veri genel amaçlı kaydedicilerden birindeyse alınıp getirilmesini,

gerek duyulan veri harici ortamdaysa (bellek veya çevresel cihazlarda), BIU' den verinin talep edilmesi gibi işlemleri gerçekleştirir. EU bölümüne genel amaçlı kaydediciler, kontrol birimi, aritmetik ve mantıksal komutların işlendiği biri dahildir.



Şekil-5. 16-Bitlik İşlemcilerde Paralel İşlem

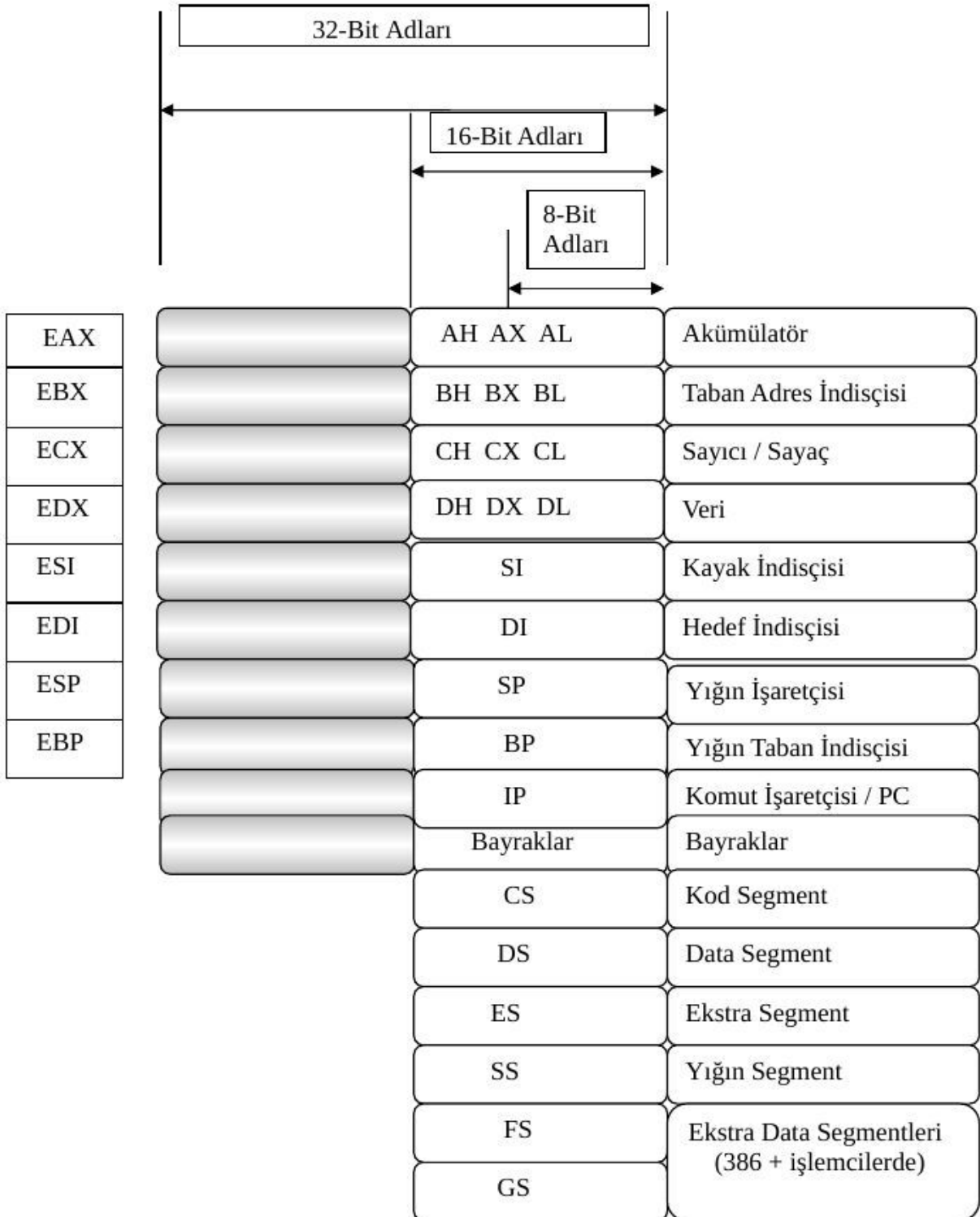
ARİTMATİK VE MANTIK BİRİMİ

Komut kodu çözüldükten sonra, matematik veya mantık işlemlerini ALU (Aritmetik ve Logic Unit) birimi yapmaktadır.

KAYDEDİCİLER

1-)SEGMENT KAYDEDİCİLER

Bilginin yönetimi Bellekte başlangıç adresi segment kaydedicileri tarafından tutulur.



Şekil-7.80*86 işlemcilerin kaydedici modeli

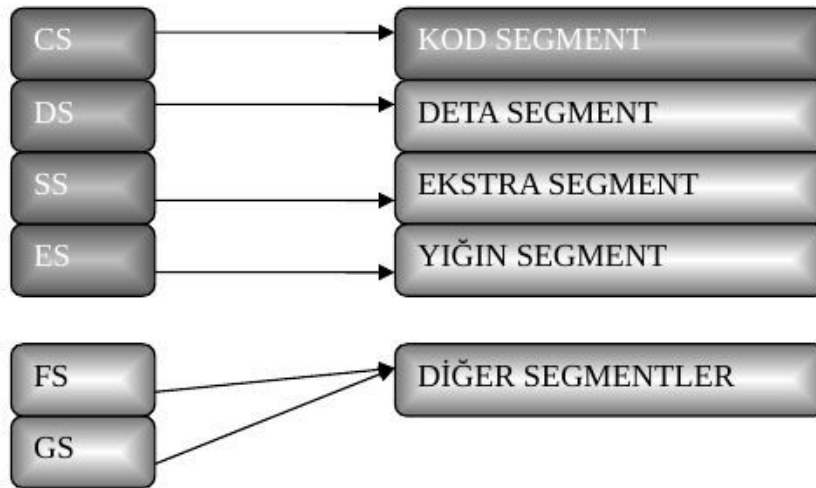
C) SEGMENT ADRESİ: Bblok adı yanındaki ifade ise o bloktaki sınıf numaralarının (05) ofset adresi olarak tanımlanabilir.

KOD SEGMENT KAYDEDİCİSİ: CS olarak adlandırılır. Kod segment, bellekte çalıştırılacak komutların sıralı bir şekilde bulunduğu bölümdür.

DATA SEGMENT KAYDEDİCİSİ: DS olarak adlandırılır Tam adresin segment tarafını gösterir. Segmentteki veriler genelde ilk veri segmenti olan data segmentinde depolanır.

EKSTRA SEGMENT KAYDEDİCİSİ: ES olarak adlandırılır. Bu kaydedici kullanıcı tarafından tanımlanmadıkça kullanılamaz. String işlemlerinde hedef adresi olarak adlandırılır.Kaynak veriler data segmentte bulunurken sonuçlar ekstra segmente aktarılır.

YIĞIN SEGMENT KAYDEDİCİSİ: SS olarak adlandırılır. Bir takım veri işlenirken yer yokluğundan veya kaydedici yetersizliğinden dolayı verinin geçici olarak yerleştirildiği yerdir.



Şekil -8. Segment kaydediciler ve bellekteki gösterdikleri alanlar

GENEL AMAÇLI KAYDEDİCİLER

Mikroişlemcide program komutlarının icrası sırasında verilerin manevrasında kullanılan ve yaoısal olarak en küçük bölümü 8-bit bellek hücresine benzeyen elektronik elemanlardır. Yaptıkları işlere göre 2 ye ayrılırlar.

1- GRUP:EAX,EBX,ECX,EDX,ESI,EDI,EBP

2-GRUP:ESP,EIP ve BAYRAK kaydedicisi

386 işlemciye kadar bu kaydediciler 16-bitlik AX,BX,CX,DX olarak işlem görülür. Daha küçük 8-bitlik veriler (bayt) işlenmesinde kullanılmak üzere daha da ufak

parçalarla anımlanlanmaktadır. AH,AL,BH,BL,CH,CL,DH,DL gibi.

EAX		AH	AX	AL
EBX		BH	BX	BL
ECX		CH	CX	CL
EDX		DH	DX	DL
ESI		SI		
EDI		DI		
ESP		SP		
EBP		BP		
EIP		IP		
EFLAGS				

Şekil-9. X86 tabanlı işlemcilerdeki genel amaçlı kaydedici modeli

AX KAYDEDİCİSİ: Akümülatör AX koduyla tanımlanır ve verilen ilk ele alınmasında kullanılır ve baş kaydedici olarak adlandırılır. 8,16,32 bitlik verilerle çarpma bölme, bazı I/O işlemlerinde ve bazı harf dizi işlemlerinde kullanılır.

BX KAYDEDİCİSİ: Taban adres kaydedicisidir. Bellekteki veri gruplarının ofsetinin tutulmasında indisçi gibi davranır. Hesaplamalarda ve 32-Bitlik işlemcilerde, bellekteki verinin adreslenmesinde kullanılır.

CX KAYDEDİCİSİ: Sayaç kaydedicisidir. Harf dizi işlemlerinde tekrarlama sayıcısı gibi çok değerli görevleri yerine getirir. CL kaydedici parçası, özellikle birden fazla kaydırma ve yönlendirme işlemlerinde, kaydırma bilgisini üzerinde tutar. String işlemlerinde, REPn komut ön-takısı ile, LOOP komutuyla birlikte, üzerindeki değer sıfır olana kadar işlem sürdürülür.

DX KAYDEDİCİSİ: Data kaydedicisidir. Akümülatöre yardımcı olan ve bütün işlemlerde tampon gibi davranır. Giriş-Çıkış (I/O) işlemlerde port numarasını üzerinde tutarken, büyük değerlikli sayıların (Word ve Doubleword) çarpılması ve bölünmesinde AX ile birlikte kullanılır.

İŞARETÇİ Ve İNDİS KAYDEDİCİSİ: Bellekteki ara adresleri gösteren kaydedicidir. Kod segmente bulunan bir komutun yerinin belirlenmesinde, CS kaydedicisine ofset değerini bulmada komut kaydedicisi (IP) yardımcı olur. Yığın bölgesindeki bir verinin yerinin belirlenmesinde, Yığın kaydedicisiyle birlikte, ofset değeri olarak SP yada BP kaydedicisi (Stack Pointer) yardımcı olur.

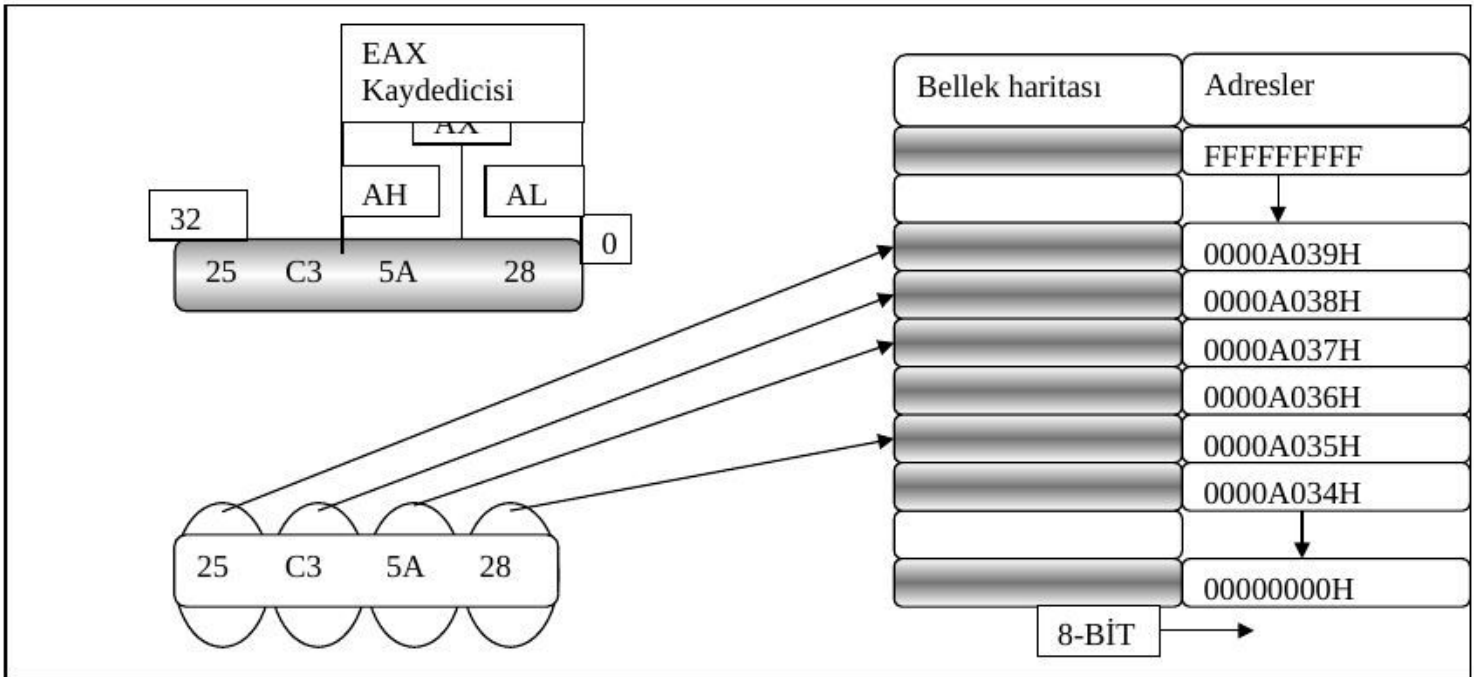
Source Index (SI,kaynak indisçi) ve Destination Index (DI,Hedef indisçi) kaydedicileri adres indislemesi işlemlerinde kullanılır.

BAYRAK KAYDEDİCİSİ: Tipine bağlı olarak 8-Bit,16-Bit ve 32-Bit olmak üzere, bir işlemin sonunda sonucun ne olduğunu kaydedici bitlerine yansıtan bir bellek hücresidir. Kaydedici bitlerin mantıksal 1 olması bayrak kalktı, mantıksal 0 olması bayrak indi anlamındadır.

BELLEK ADRESLEMESİ

MANTIKSAL BELLEK: Bütün adresler Bayt olarak numaralandırılır. 16 adres hattı işlemcilerin adres numaraları 0000H ile başlar, FFFFH ile biter. Adres uzayları 1MB' tır.

32 hattı işlemcilerin numaraları 00000000H ile başlar, FFFFFFFFH ile biter. İstem adresleme kapasiteleri 4GB olur.

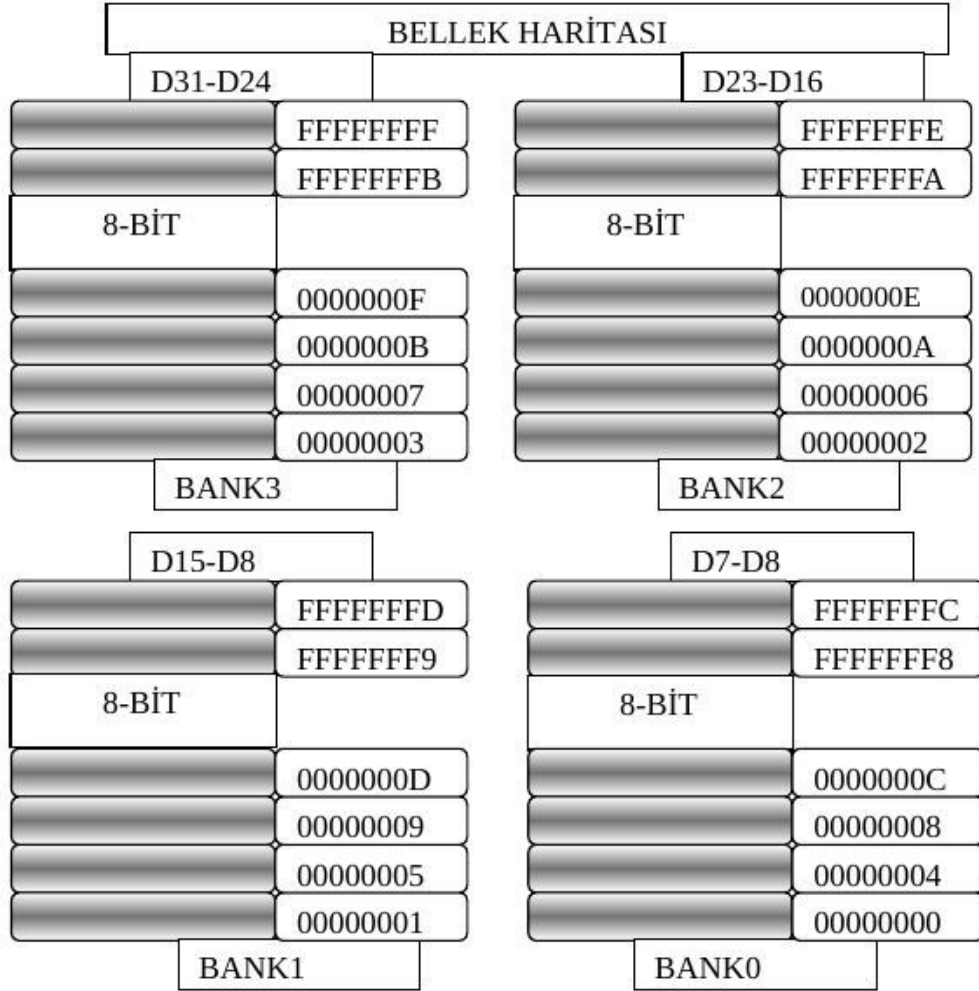


Şekil-11. 32 Bitlik verinin belleğe yerleşmesi

Bellek model tanımına bağlı olarak, bir mikroişlemci bir anda bellekteki bir veya daha fazla baytlık veriye erişebilir.

FİZİKSEL BELLEK TANIMLAMASI

Belleklerin fiziksel tanımlaması donanımsal bir yaklaşım olduğundan işlemci tipleri ve mimarisine bağlıdır. Programcı için bütün bellek tipleri veya grupları, daima mantıksal olarak 8-bit.



Şekil-12. 32-Bitlik işlemcilerde fiziksel bellek yapısı

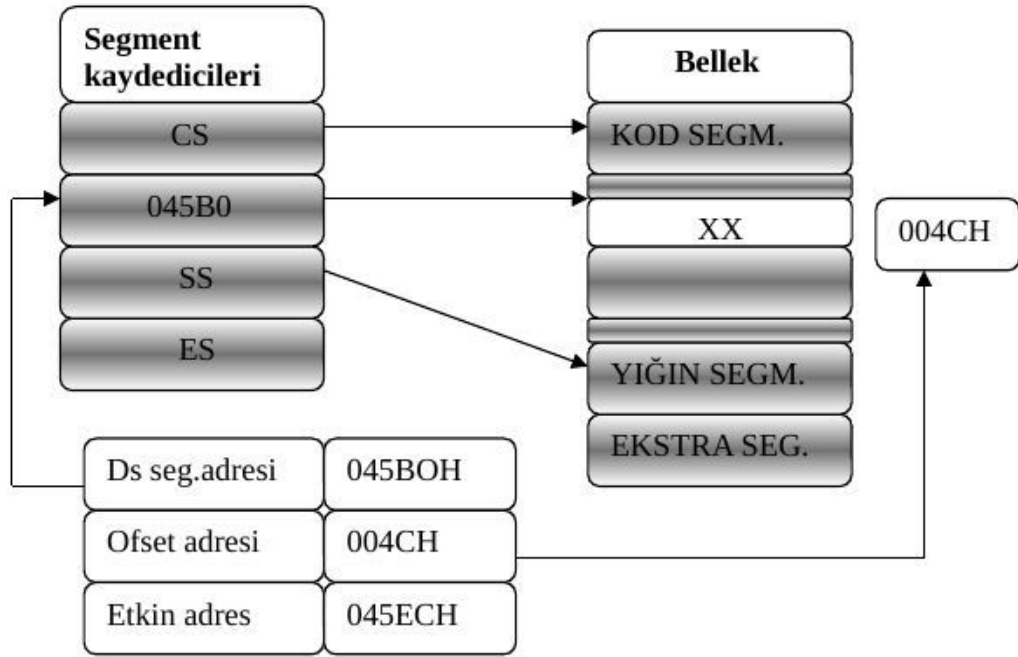
SEGMENT SINIRLARI

Segmentler, adreslerin eşit bir şekilde ondalık olarak 16 ile (ondalık 10) bölünebilen ve adına paragraf sınırı denilen adresle başlar. Verilerin belleğe belli bir orjin noktasından başalamasında veri kaybı en aza indirilmek istenir ve bundan dolayı da segment adres başlangıçları daima 0H ile başlar.

OFSET ADRESİ

Verinin bulunduğu adres, segment kaydedicisinin gösterdiği başlangıç adresinden uzaklığı kadardır.

64KB'lık bir bellek uzayında, adres 0000H 'dan başlayarak FFFFH 'e kadar gider. 4GB'lık bir bellek uzayında 64KB'lık segment tanımı, ofset haneleri artsa bile yine benzer tanımlar yani 00000000H'dan başlayarak 0000FFFFH kadar uzanır. Segment adresleri işlemci tarafından otomatik atanır.



Şekil-13. Ofset adres ile segment adresinin birleşimi

PIC16F84

3.

BÖLÜM

PİC NEDİR?

MİKROKONTROLCÜ

PIC ve CPU, bellek, osilatör, watchdog ve I/O' nun tek bir yonga üzerinde bulunduğu bir mikrokontrolcüdür. Bu, tasarımda zaman ve yer kazandırmakta, aynı zamanda çevre birimlerin zamanlama ve uyumluluk problemlerini de azaltmaktadır. Ancak bazı durumlarda bellek boyutlarını ve I/O kapasitesini kısıtlayabilir. PIC mikrokontrolcüler ailesindeki ürünler, I/O, bellek ve özel fonksiyonlar geliştirme mühendislerinin çoğunun ihtiyaçlarını karşılayabilecek ölçüde geniş bir yelpaze sunar.

PIC Serisi mikroişlemciler MICROCHIP firması tarafından geliştirilmiş ve üretim amacı çok fonksiyonlu logic uygulamalarının hızlı ve ucuz bir Mikroişlemci ile yazılım yoluyla karşılanmasıdır. PIC'in kelime anlamı -PERIPHERAL INTERFACE CONTROLLER- Giriş Çıkış işlemcisidir. İlk olarak 1994 yılında 16 bitlik ve 32 bitlik büyük işlemcilerin giriş ve çıkışlarındaki yükü azaltmak ve denetlemek amacıyla çok hızlı ve ucuz bir çözüme ihtiyaç duyulduğu için geliştirilmiştir. Çok geniş bir ürün ailesinin ilk üyesi olan PIC16C54 bu ihtiyacın ilk meyvesidir. PIC işlemcileri RISC -benzeri işlemciler olarak anılır. PIC16C54 12 Bit komut hafıza genişliği olan 8 bitlik CMOS bir işlemcidir. 18 bacaklı dip kilifta 13 I/O bacağına sahiptir ve 20 Mhz osilatör hızına kadar kullanılabilir. 33 adet komut içermektedir. 512 byte program epromu ve 25 byte RAM'i bulunmaktadır. Bu hafıza kapasitesi birçok insani güldürmüştür sanırım ama bir risc işlemci olması birçok işin bu kapasitede uygulanmasına olanak vermektedir. Örneğin ANTRAK R94 rölesinde kullanılan röle kontrol devresi bir adet PIC16C54 içermektedir. Bu devre sayesinde R94 Time Out Timer, DTMF kodlu Kontrol ve kapatma, Konuşma sonu zamanlaması gibi işlemler program belleğinde 324 byte yer kaplamakta 14 byte RAM kullanılmaktadır. PIC serisi tüm işlemciler herhangi bir ek bellek veya giriş/çıkış elemanı gerektirmeden sadece 2 adet kondansatör, 1 adet direnç ve bir kristal ile çalıştırılabilir. Tek bacaklı 40 mA akım çekilebilmekte ve entegre toplamı olarak 150 mA akım akıtma kapasitesine sahiptir. Entegrenin 4 Mhz osilatör frekansında çektiği akım çalışırken 2 mA stand-by durumunda ise 20uA kadardır. PIC 16C54'ün fiyatının 2.0 US\$ civarında olduğu düşünülürse bu işlemcinin avantajı kolayca anlaşılar.

PIC 16C54 'un mensup olduğu işlemci ailesi 12Bit core 16C5X olarak anılır. Bu gruba temel grup adı verilir. Bu ailenin üyesi diğer işlemciler PIC16C57, PIC16C58 ve dünyanın en küçük işlemcisi olarak anılan 8 bacaklı PIC12C508 ve PIC 12C509'dur. Interrupt kapasitesi ilk işlemci ailesi olan 12Bit Core 16C5X ailesinde bulunmamaktadır. Daha sonra üretilen ve Orta sınıf olarak tanınan 14Bit Core- 16CXX ailesi birçok açıdan daha yetenekli bir grup işlemcidir. Bu ailenin temel özelliği interrupt kapasitesi ve 14 bitlik komut işleme hafızasıdır. Bu özellikler

Pic'i gerçek bir işlemci olmaya ve karmaşık işlemlerde kullanılmaya yatkın hale getirmiştir. PIC16CXX ailesi en geniş ürün yelpazesine sahip ailedir. 16CXX ailesinin en önemli özellikleri seri olarak devre üstünde dahi programlanması -ki bu özellik PIC16C5x de epey karmaşıktı , paralel programlanabiliyordu- interrupt kabul edebilmesi, 33 I/O,AD Converter, USART, I2C, SPI gibi endüstri standardi giriş çıkışları kabul edecek işlemcilere ürün yelpazesinde yer vermesi. PIC 16CXX ailesinin amatör elektronikçiler arasında en çok tanınan ve dünyada üzerinde en çok proje üretilmiş, internetin gözdesi olan bireyi PIC16C84 veya yeni adıyla PIC16F84 dur.

PIC 16F84 un bu kadar popüler olması onun çok iyi bir işlemci olmasından ziyade program belleğinin Eeprom - Elektrikle silinip yazılabilen bellek - olmasından kaynaklanmaktadır. Seri olarak dört adet kabloyla programlanması da diğer önemli avantajıdır. Bugüne kadar amatörce bir işlemciyle uğraşmış herkesin en büyük sıkıntısı eeprom veya eeprom tabanlı işlemcileri programladıktan sonra UltraViolet ışık kaynağı ile silip tekrar programlamaktır. Bu çok zahmetli ve bir amatör için ekipman gerektiren yöntem olmuştur. Evde üretilmesi zor olan özel bir programlayıcı da madalyonun diğer yüzüdür. PIC16F84 amatörler tarafından internette en bol programlayıcısı bulunan işlemcidir herhalde. Ben su ana kadar 50 den fazla PC ve MAC tabanlı evde yapabileceğiniz programlayıcıya rastladım. Eeprom silmek diye birşey zaten söz konusu değil zira eeprom belleği programlayan programlayıcı devre 1 saniye içinde aynı belleği silebilmektedir.

Bu özellik size çok hızlı ve defalarca deneyerek program geliştirme avantajını getirmektedir ki bu amatör elektronikçi için bulunmaz bir nimettir. Bu denemeleri yaparken işlemciyi devrenizden sökmeniz dahi gerekmez. Bu tip programlamaya ISP -In System Programming- denmektedir. Amatör bir elektronikçi PIC16F84 ile Program geliştirmek için aşağıdaki ekipmana ihtiyaç duyacaktır: PIC16F84 MS-DOS tabanlı PC ASCII Editör (Örneğin Dos'un EDIT'I) MPASM Assembler Programlayıcı/silici Programlayıcı için yazılım Bu öğelerin tümünü bu yazı dizisinde sizlere tanıtmaya çalışacağım. Gelecek yazımda PIC 16F84 un özelliklerine daha kapsamlı değinip, RS-232 COM portu kullanan PC tabanlı oldukça karmaşık !!! (Üç adet direnç) bir programlayıcı/silici tasarımı anlatacağım. PIC Serisi işlemciler ile ilgili daha geniş bilgi için adresine başvurabilirsiniz.

NEDEN PIC?

PIC, Harvard mimarisi temelli 8 bit' lik bir mikrokontrolcüdür. Bu, bellek ve veri için ayrı yerleşik bus' ların bulunduğu anlamına gelir. Böylelikle akış miktarı veriye ve program belleğine simultane erişim sayesinde artırılmış olur. Geleneksel mikrokontrolcülerde veri ve programı taşıyan bir tek yerleşik bus bulunur. Bu, PIC' le karşılaştırıldığında işlem hızını en az iki kat yavaşlatır. Tüm komutlar 12 veya 14 bit'lik bir program bellek sözcüğüne sığar. Yazılımın, programın VERİ kısmına atlamaya ve VERİ'yi komut gibi çalıştırmasına gerek yoktur. Bu 8 bit'lik bus kullanan ve Harvard mimarisi temelli olmayan mikrokontrolcülerde gerçekleşmektedir. PIC, Harvard mimarisi temelli 8 bit' lik bir mikrokontrolcüdür.

Bu, bellek ve veri için ayrı yerleşik bus'ların bulunduğu anlamına gelir. Böylelikle akış miktarı veriye ve program belleğine simultane erişim sayesinde artırılmış olur. Geleneksel mikrokontrolcülerde veri ve programı taşıyan bir tek yerleşik bus bulunur. Bu, PIC'le karşılaştırıldığında işlem hızını en az iki kat yavaşlatır. Tüm komutlar 12 veya 14 bit'lik bir program bellek sözcüğüne sığar.

Yazılımın, programın VERİ kısmına atlamaya ve VERİ'yi komut gibi çalıştırmasına gerek yoktur. Bu 8 bit'lik bus kullanan ve Harvard mimarisi temelli olmayan mikrokontrolcülerde gerçekleşmektedir.

PIC tamamıyla statik bir mikroişlemcidir. Başka bir deyişle saati kurdurduğunuzda, tüm yazmaç içeriği korunur. Pratikte bunu tam olarak gerçekleştirmeniz mümkün değildir. PIC' i uyutma moduna getirdiğinizde, saat durur ve PIC' e uyutma işleminden önce hangi durumda olduğunu size hatırlatacak çeşitli bayraklar kurar. PIC uyuma modunda yalnızca 1 mA'dan küçük bir değere sahip bekleme (standby) akımı çeker.

PICBASIC

PICmikro assembly yazılımıyla sayfalar dolusu kod yazmaktan bıktıysanız ya da sırf assembly'den nefret ettiğiniz için mikrokontrolcülerle ilgilenmiyorsanız, PICBasic, size önemli fırsatlar sunacak.

PICmikro assembly'siyle sayfalar dolusu yazdığınız ve hata arındırma sürecinde sanki pösteği üstünde kıl sayar gibi takip etmek zorunda kaldığınız program parçalarını toplam 8-10 satırda PICBasic'le yazabilir, düzeltebilir, derleyebilir ve PIC mikro'ya yükleyebilirsiniz. Ne PIC mikronun ayrıntılı yazmaç yapısını bilmenize ne de hangi yazmaçta hangi işlemin sonunda ne olacağını bilmenize gerek var. Kullanmayı düşündüğünüz PIC mikronun hangi portlarını ve özelliklerini kullanacaksanız, algoritmanızı kurun, herhangi bir metin editörü (NOTEPAD, WORD, EDIT vb) ile bildiğimiz BASIC komutlarına çok benzeyen PICBasic komutları ile program örneği;

PICBasic Standart:

- Desteklediği PIC mikrolar: 12C67x, 16C55x, 12C6x, 14000, 16C62x, 16C71x, 16F8x, 16F87x, 16C9x serisi...
- Derleme kapasitesi: 2K...
- MPLAB ortamına entegre olma olanağı...
- Assembly kodlarını ekleme olanağı...

PICBasic Profesyonel:

- Desteklediği PIC mikrolar: 12C67x, 16C55x, 12C6x, 14000, 16C62x, 16C71x, 16F8x, 16F87x, 16C9x serisi...
- Derleme kapasitesi: PIC mikronun program belleği oranında sınırsız...
- MPLAB ortamına entegre olma olanağı...
- Harici/Dahili seri EEPROMlara yazma-okuma,LCD göstergeler, dahili-harici AD çeviriciler vb ve matematiksel işlemler için özel hazırlanmış fonksiyonlar...
- Assembly kodlarını ekleme olanağı...

Her iki sürümünde İngilizce olarak hazırlanmış bol örnekli kullanma kılavuzu bulunmakta...

PICBASIC PRO Uygulama Örneği: Harici AD Çeviriciler...

Her ne kadar PICmicro ailesine ait bir grup mikrodenetleyici 8-10 bit A/D çeviriciyi yonga üzerinde kullanıma sunsa da bazen çözünürlüğü daha yüksek A/D çeviricilere de gereksinim olmakta.

Ya da kullanılabilir port sayısında sıkıntımız varsa harici A/D çeviriciler kullanmak daha anlamlı olabilmekte. Bu ayki uygulamamız 40 bacaklı 16C6x veya 16C7x serisinden herhangi bir PICmicro'yla kullanılmak üzere geliştirilmiş , 2 ayrı kanaldan okunan analog bilginin LCD göstergede gösterilmesi üzerine. Prototip devresinde kullanılan PIC16F877 üzerinde 8 adet 10 bitlik A/D çevirici portu varsa da bu

PICmicro'nun seçiminin temel nedeni uygulama yazılımını geliştirirken sağladığı kolaylıktan başka bir şey değil. A/D çevirici olarak iki kanallı, 12 bit çözünürlüğe sahip seri bir A/D çevirici yongası olan LTC1298 kullanılmıştır. Uygulama devresi Yeni Başlayanlar için PIC PROGRAMLAMA EL KİTABINDA bulabilirsiniz. Buna ilişkin uygulama yazılımıysa, daha önceki sayılarda yayınlanan assembly ile sınırlanmış PICmicro uygulamalarından biraz farklı olarak, geçen sayımızda tanıttığımız PicBASIC Pro'yla hazırlanmıştır.

LTC1298 ADC'yi okumak için hazırlanan Picbasic Pro programı

'HD44780 uyumlu LCD bağlantıları

```
Define LCD_DREG PORTD
Define LCD_DBIT 4
Define LCD_RSREG PORTE
Define LCD_RSBIT 0
Define LCD_EREG PORTE
Define LCD_EBIT 1
    include "modedefs.bas"
'Bacak tanımları
CS var PORTC.5 ' Chip select
CK var PORTC.3 ' Clock var
DI var PORTA.2 ' Data in var
DO var PORTC.1 ' Data out
```

```
'Değişken tanımları
addr    var    byte ' Kanal adresi / mod
result  var    word
x        var    word
y        var    word
z        var    word
```

High CS

'Chip select etkin değil,yani LTC1298 devrede değil.

ADCON1 = 7

' PORTA, PORTE sayısal port. PIC16F877'de reset anında
' PORTA ve PORTD analog giriştir.Bu uygulamada analog
' girişleri sayısal giriş olarak kullanacağımızdan bu
' tanımlamayı yapmak gerekiyor.

Low PORTE. 2

' LCD R/W hattı düşük (W).

Pause 100

' LCD'nin başlama rutini için bekle.

Goto mainloop

' alt rutinleri atla.

' A/D çeviriciyi okumak için alt rutin

getad

CS = 0

'Chip select etkin,yani LTC1298 devrede.

'[adres / mod - başlangıç biti, 3 bit addr, 0 biti] 'gönder.

ShiftOut DI, CK, MSBFIRST, [DI, addr\3, 0\1]

'12bitlik sonucu al

ShiftIn DO, CK, MSBPRE, [result\12]

CS = 1

' Chip select etkin değil,yani LTC1298 devrede değil.

Return

'x değerini okumak için (0. kanal) alt rutin

getx :

addr = \$05

'tek sonlu,0.kanal,MSBF yüksek

Gosub getad

x = result

Return

'y değerini okumak için (1. kanal) alt rutin

gety :

addr = \$07

tek sonlu, 1. kanal, MSBF yüksek

Gosub getad

y = result

Return

'z değerini okumak için (0. ve 1. kanal farkı) alt rutin

addr = \$01

Farksal (ch0 = + , ch1 = -), MSBF yüksek

Gosub getad

z = result

Return

mainloop:

Gosub getx 'x değerini al

Gosub gety 'y değerini al

Gosub getz 'z değerini al

'Değerleri LCD'ye gönder

Lcdout \$fe, 1, "x=", "#x, "y=", #y, "z=", #z

Pause 100 'saniyede 10 kere

Goto mainloop 've sürekli yap

End

Yukarıdaki programı assembly ile yazmış olsaydık 10 katından fazla yer işgal etmek durumunda kalırdık herhalde...

DONANIM

Herhangi bir sistem tasarlamak için, donanım ve yazılımın tamamen anlaşılması şarttır. Donanım, dış dünyayla iletişimini sağlayan arabirim olduğundan, yazılımdan önce incelenmelidir. Bu bölümde şunların üstünde durulacak:

Mikrokontrolün Yapısal Analizi:

Ana hatlarıyla bir mikroislemci kendini var eden temel etmenlerden oluşur. Bunlar kısaca MPU, RAM, ROM, IO dur. Simdi kısaca bu yapılara deginelim;

MPU : Merkezi Islem Birimi olarakda anılan birim mikroislemcimizin ana yapısal islemleri olan mantiksal islemler, aritmetik islemler ve yürütme islemlerini gerçekleştirir yani kısaca diger birimlerden aldigi verileri isleyerek yeniden baska bir birime aktaran kismidir. Belirli bir saat frekansinda çalışır yani saniyede belli sayıda islem gerçekleştirir. Kendi içerisinde ROM,RAM ve IO'suda vardır.

ROM : Yanlizca okunabilen bellek olarak adlandırılan ROM üretim esnasında bir kerelik olarak doldurulan hafıza parçasıdır.

RAM : Asil bellek olarak bilinen kısımdır aktif olarak merkezi işlem birimi tarafından kullanılır azlığı veya çokluğu sistem kapasitesini etkiler. Elektrik kesilince silinir.

IO : Mikrokontrolcünün dissal bagidir. Disaridan alınan bilgiyi MPU ve yine MPU 'dan aldığı veriyi dissal bir porta aktaran korunmuş kısımdır. Sistemdeki en yavas aygitlardır.

PIC' in iç yapısı YAZILIM TASARIMI

Donanımı ve PIC' in gerçek dünyaya nasıl bağlanacağını anladıktan sonra, PIC' in birşeyler yapmaya başlamak için yazılım konusunu öğrenmek gerekiyor.

Önceden yazılımla ilgili çalışmalarınız varsa, aşağıdaki bilgilerden bir kısmını atlayabilirsiniz.

- mantıksal fonksiyonların anlaşılması
- onaltılık-heksadesimal gösterim
- kaynak kod yazımı
- assembler direktiflerinin kullanımı
- kod yazımında zaman kazandıran makrolar
- program geliştirme
- komut seti
- dahili yazmaçlar
- alt rutinler,
- lookup tabloları
- I/O port kurulumu
- Reset vektörleri
- RTCC
- WDT
- kesmeler
- stack
- A/D dönüştürme

- E2 temelli PIC' ler
- hangi PIC' i seçmeli?
- gereksinilen asgari donanım
- reset devreleri, zamanlayıcılar
- watchdog
- osilatörler
- girdiler ve çıktılar
- I/O genişlemesi
- şehir şebekesi ile bağlantı
- analog sinyal işleme
- PIC' ler de numaralandırma

PIC16F84 YAPISAL GÖRÜNÜM

PIC16CXX ailesinin yüksek performansı, RISC mikro işlemcilerinde sıkça bulunan bir dizi yapısal niteliğe atfedilebilir. Başlangıç olarak, PIC16CXX bir Harvard Yapısı kullanmaktadır. Bu yapı, ayrı hafızalardan kabul edilen program ve verilere sahiptir. Bu yüzden gereç bir program hafıza taşıyıcısı ve bir veri hafıza taşıyıcısına sahiptir. Bu, program ve verilerin aynı hafızadan edinildiği (aynı taşıyıcı üzerinden erişimler) geleneksel von Neumann yapısı üzerinde bant genişliğini geliştirmektedir.

Program ve veri hafızasını ayırmak dahası 8 bit genişliğindeki veri dünyasından farklı olarak boyutlandırılması için talimatlara izin vermektedir. PIC16CXX opkodları, tek kelime talimatlarını mümkün kılan 14 bit genişliğindedir. Tam 14 bit genişliğinde program hafıza taşıyıcısı tek bir döngü dahilinde bir 14 bitlik talimat sağlamaktadır. İki kademeli bir boru hattı talimatların alınması ve gerçekleştirilmesini kısmen kaplamaktadır (Örnek 3 – 1). Sonuç olarak, program dalları için hariç tüm talimatlar tek bir döngüde gerçekleştirilmektedir (400 ns @ 10 MHz). PIC16C84, 1 K x 14 program hafızasına hitap etmektedir. Tüm program hafızası içeridedir. PIC16CXX gereçleri, kayıt dosyaları veya veri hafızasına doğrudan veya dolaylı olarak hitap etmektedir. Program sayacı dahil tüm özel işlev kayıtları veri hafızası dahilinde haritalandırılmıştır.

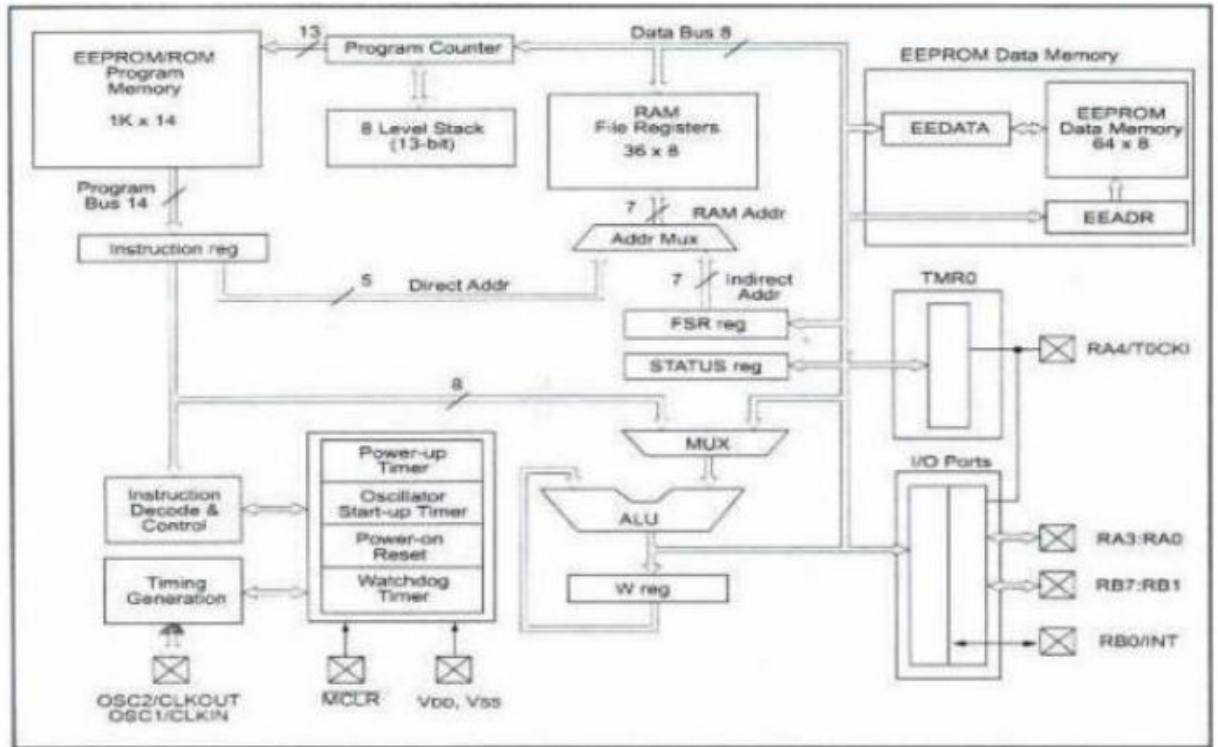
Bir ortogonal (simetrik) talimat, herhangi hitap etme modunu kullanan herhangi bir kayıt hakkında herhangi bir işlemi gerçekleştirmesini mümkün kılacak şekilde ayarlamaktadır. Bu simetrik yapı ve “özel optimal durumlar”ın eksikliği,

PIC16CXX ile programlamayı basit bir şekilde etkin hale getirmektedir. Ek olarak, öğrenim eğrisi belirgin bir şekilde azaltılmıştır.

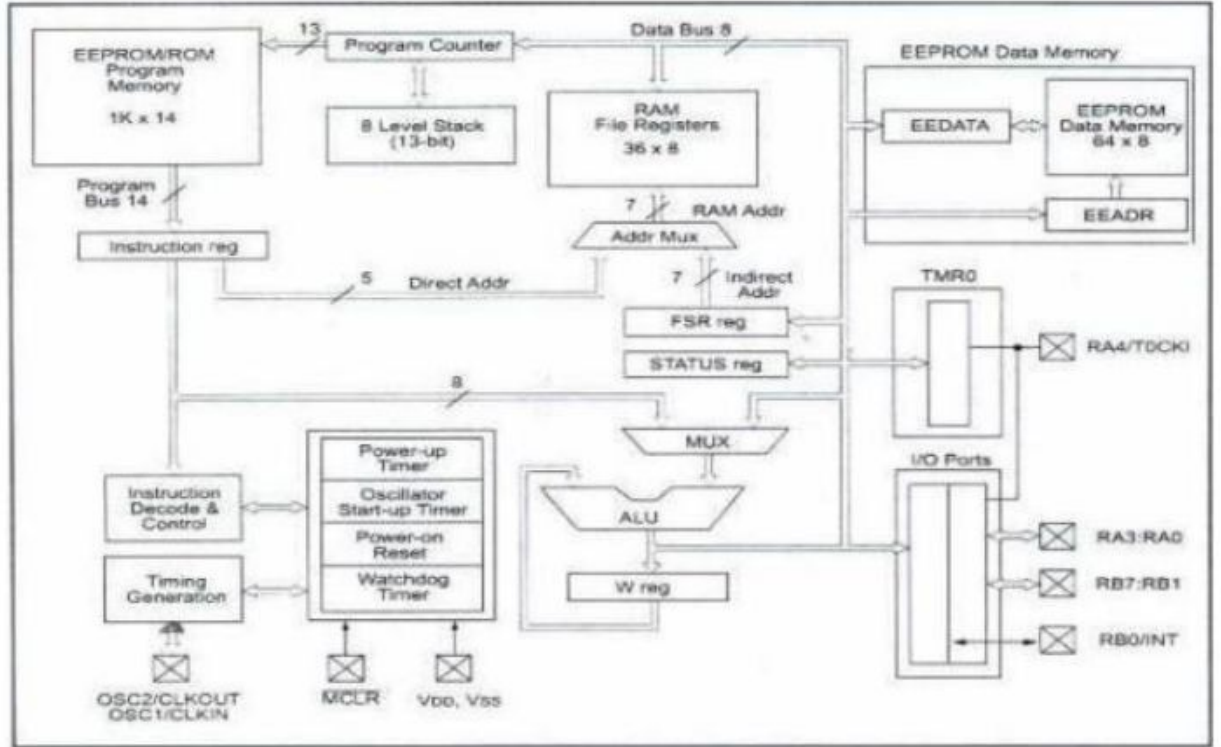
PIC16C84, 36 x 8 SRAM ve 64 x 8 EEPROM veri hafızasına sahiptir. PIC16C XX gereçleri, bir 8 bitlik ALU ve çalışan bir kaydedici içermektedir. ALU genel amaçlı bir aritmetik birimdir. Çalışan bir kaydedici ve herhangi bir kayıt dosyasındaki veriler arasındaki aritmetik ve Boolean işlevlerini gerçekleştirmektedir.

ALU, 8 bit genişliğinde ve ekleme, çıkarma değişim ve mantıksal işlem yeteneğidir. Aksi belirtilmediği sürece, aritmetik işlemler doğal olarak ikinin bileşenleridir. İki işlenen talimatları dahilinde, tipik olarak bir işlenen çalışan kaydedicisidir (W kaydedicisi), ve diğer işlenen bir dosya kaydedicisi veya bir anlık sabitidir. Tek işlenen talimatlarında, işlenen ya W kaydedicisi veya bir dosya kaydedicisidir. W kaydedicisi, ALU işlemleri için kullanılan 8 bitlik bir çalışan kaydedicisidir. Yönlendirilmiş bir kayıt değildir.

Gerçekleştirilen talimata bağlı olarak ALU, STATUS (Statü) kaydında Taşıma (C), Basamak Taşıma (DC) ve Zero (Sıfır) (Z) bitlerinin değerlerini etkileyebilir. C ve DC bitleri, çıkarımda sırasıyla alıcı ve basamak alıcı dış biti olarak çalışmaktadır. Örnekler için SUBLW ve SUBWF talimatlarına bakınız. PIC16C84 için basitleştirilmiş bir blok diyagramı aşağıda gösterilmiştir.



ŞEKİL 3-1 : PIC16C84 BLOCK DIAGRAM



ŞEKİL 3-1 :PIC16C84 BLOCK DIAGRAM

3.1 Zamanlayıcı Şeması / Talimat Döngüsü

(OSC1'den) Saat girdisi, Q1, Q2, Q3 ve Q4 olarak adlandırılan dört kısmen kaplamayan dördüleme saatlerini oluşturmak amacıyla içten dörde bölünmüştür. Dahili olarak, program sayacı (PC) her bir Q1'i artırmakta olup talimat program hafızasından alınıp Q4 dahilinde talimat kaydı içine kilitlemektedir. Q1'den Q4'e doğru takip ederken talimat çözülüp gerçekleştirilmektedir. Saatler ve talimat gerçekleştirme akışı Şekil 3 – 2 dahilinde gösterilmiştir.

3.2 Talimat Akışı / Boru Hattı

Bir "Talimat Döngüsü" dört Q döngüsünden (Q1, Q2, Q3 ve Q4) oluşmaktadır. Talimat alımı ve gerçekleştirilmesi, çözülüp gerçekleştirme diğer bir talimat döngüsünü oluştururken, bir talimat döngüsünü oluşturmaktadır. Yine de, boru hattı dolayısı ile, her bir talimat etkin bir şekilde tek döngüde gerçekleştirilmektedir. Eğer bir talimat program sayacının değişmesine neden olursa, (ÖR: GOTO) bu durumda talimatı tamamlamak için iki döngü gereklidir. Bir alım döngüsü, Q1 dahilinde artan Program Sayacı (PC) ile başlamaktadır. Gerçekleştirme döngüsünde, alınan talimat, Q1 döngüsündeki "Talimat Kaydı" içine kapatılmaktadır. Bu talimat daha sonra Q2, Q3 ve Q4 döngüleri esnasında çözülür ve gerçekleştirilir. Veri hafızası Q2 esnasında okunmakta olup (işlenen okuması) ve Q4 esnasında yazılır (amaç yazılımı).

3.0 YAPISAL GÖRÜNÜM

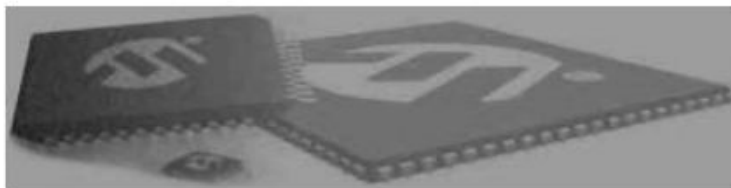
PIC16CXX ailesinin yüksek performansı, RISC mikroişlemcilerinde sıkça bulunan bir dizi yapısal niteliğe atfedilebilir. Başlangıç olarak, PIC16CXX bir Harvard Yapısı kullanmaktadır. Bu yapı, ayrı hafızalardan kabul edilen program ve verilere sahiptir. Bu yüzden gereç bir program hafıza taşıyıcısı ve bir veri hafıza taşıyıcısına sahiptir. Bu, program ve verilerin aynı hafızadan edinildiği (aynı taşıyıcı üzerinden erişimler) geleneksel von Neumann yapısı üzerinde bant genişliğini geliştirmektedir. Program ve veri hafızasını ayırmak dahası 8 bit genişliğindeki veri dünyasından farklı olarak boyutlandırılması için talimatlara izin vermektedir. PIC16CXX opkodları, tek kelime talimatlarını mümkün kılan 14 bit genişliğindedir. Tam 14 bit genişliğinde program hafıza taşıyıcısı tek bir döngü dahilinde bir 14 bitlik talimat sağlamaktadır. İki kademeli bir boru hattı talimatların alınması ve gerçekleştirilmesini kısmen kaplamaktadır. Sonuç olarak, program dalları için hariç tüm talimatlar tek bir döngüde gerçekleştirilmektedir (400 ns @ 10 MHz). PIC16C84, 1 K x 14 program hafızasına hitap etmektedir. Tüm program hafızası içeridedir. PIC16CXX gereçleri, kayıt dosyaları veya veri hafızasına doğrudan veya dolaylı olarak hitap etmektedir. Program sayacı dahil tüm özel işlev kayıtları veri hafızası dahilinde haritalandırılmıştır.

Bir ortogonal (simetrik) talimat, herhangi hitap etme modunu kullanan herhangi bir kayıt hakkında herhangi bir işlemi gerçekleştirmesini mümkün kılacak şekilde ayarlanmaktadır.

Bu simetrik yapı ve "özel optimal durumlar"ın eksikliği, PIC16CXX ile programlamayı basit bir şekilde etkin hale getirmektedir. Ek olarak, öğrenim eğrisi belirgin bir şekilde azaltılmıştır. PIC16C84, 36 x 8 SRAM ve 64 x 8 EEPROM veri hafızasına sahiptir.

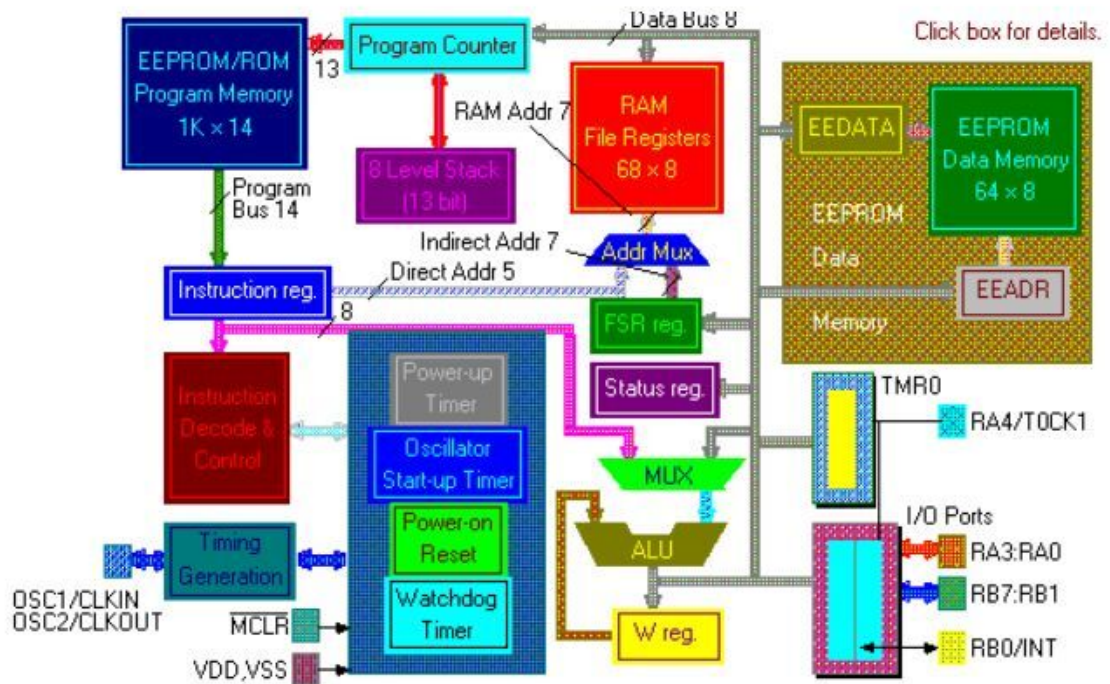
PIC16CXX gereçleri, bir 8 bitlik ALU ve çalışan bir kaydedici içermektedir. ALU genel amaçlı bir aritmetik birimdir. Çalışan bir kaydedici ve herhangi bir kayıt dosyasındaki veriler arasındaki aritmetik ve Boolean işlevlerini gerçekleştirmektedir. ALU, 8 bit genişliğinde ve ekleme, çıkarma değişim ve mantıksal işlem yeteneğidir. Aksi belirtilmediği sürece, aritmetik işlemler doğal olarak ikinin bileşenleridir. İki işlenen talimatları dahilinde, tipik olarak bir işlenen çalışan kaydedicisidir (W kaydedicisi), ve diğer işlenen bir dosya kaydedicisi veya bir anlık sabittir. Tek işlenen talimatlarında, işlenen ya W kaydedicisi veya bir dosya kaydedicisidir. W kaydedicisi, ALU işlemleri için kullanılan 8 bitlik bir çalışan kaydedicisidir. Yönlendirilmiş bir kayıt değildir. Gerçekleştirilen talimata bağlı olarak ALU, STATUS (Statü) kaydında Taşıma (C), Basamak Taşıma (DC) ve Zero (Sıfır) (Z) bitlerinin değerlerini etkileyebilir. C ve DC bitleri, çıkarmada sırasıyla alıcı ve basamak alıcı dış biti olarak çalışmaktadır. Örnekler için SUBLW ve SUBWF talimatlarına bakınız.

Programlanabilir Kontrol Arayüzü(PIC)



Pic üretilmiş bulunan mikroislemcilerden farklı olarak basite indirgenmiş fakat hızı ve etkisi düşürülmemiş olan bir mikrochiptir. MICROCHIP firması tarafından uygulama formları için üretilmiştir. Basit yonga seti, uygun portları ve opsiyonel işlevleri bakımından hedef kontrolü için bulunmaz bir mikroislemcidir PIC. Yapısal olarak risc işlemcileri çağrıştırmakla beraber geliştirilmiş komut yapısı sayesinde emsallerinden çok daha çabuk ve hızlı programlanabilir. 8-16 bitlik işlemcilerdir. 20 MHz kadar çıkabilirler. Bazı tiplerinde üzerindeki portlar ADC ve DAC görevini tek word üzerinden görebilmektedirler. Rakiplerine göre hızları ve çalışma ortamları daha esnek ve kararlidir. 2.5V-6V arasında çalışabilirler ve son olarak cazip fiyatları bu tip ihtiyaçları tam olarak karşılamaktadır. (Ayrıca enson teknoloji her zaman için bir öncekinden daha avantaj ve yenilik içerir.)

Aşağıdaki model PIC 16C serisidir. Bu serideki işlemcilerde EEPROM chip içerisinde mevcut olduğundan ektradan eeprom bağlamak ve işletmek için port ayırmak gerekmemektedir. Aşağıda temel olarak bir pic 16c serisi mikrochip'in blok şemasını görmekteyiz;



Şemada görüldüğü gibi dissal ve içsel interruptları(kesmeleri) mevcuttur. Bu devrede motorun encoderi real time çalışmasının sağlanması için A portunun 5 numaralı bacağı ile uart sinyalizasyonuna uydurulmuştur.

'#' Yerine '\$' Bastırma

```
.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,09
LEA ES:DI,DIZI
REPNE SCASB
JNE CIK
MOV BYTE PTR[DI - L],BH

```

```

CIK: MOV AH,4CH
INT 21H
ANA ENDP
END ANA

```

Ascii Karakter Setinin Ekranda Görüntülenmesi

```

Page 60 , 132
TITLE ALLASC (COM) ASCII Karakterlerini Görüntüle(00FF)
CODESG SEGMENT PARA 'Code'
ASSUME CS:CODESG , DS:CODESG , SS:CODESG , ES:NOTHING
ORG 100H
BEGIN JMP SHORT MAIN
CTR DB 00 , '$'

```

Ana Prosedür

```

MAIN PROC NEAR
CALL B10CLR (Ekranı Sil)
CALL C10SET (İmleci Ayarla)
CALL C10DISP (Karakterleri Görüntüle)
RET
MAIN ENDP

```

Ekranı Sil

```

B10CLR PROC
MOV AX , 0600H
MOV BH , 07
MOV CX , 0000 (Üst Sağ Konum)

```

```
MOV DX , 184FH (Alt Sag Konum)
INT 10H
RET
B10CLR ENDP
```

İmleci 0000'a Aayarla

```
C10SET PROC
MOV AH , 02
MOV BH , 00
MOV DX , 0000
INT 10H
RET
C10SET ENDP
```

Ascii Karakterleri Görüntüle

```
D10DISP PROC
MOV CX , 256 (256 Tekrar İçin İlk Değeri Verir)
LEA CX , CTR (Sayacın Adresine İlk Değeri Verir)
D20:
MOV AH , 09 (Ascii Karakterini Görüntüle)
INT 21H
INC CTR (Sayacı Artır)
LOOP D20 (CX'i Azalt 0'dan Farklı İse Devam)
RET
D10DISP ENDP
CODESG ENDS
END BEGIN
```

İki Byte'i Toplayan Program

```
PAGE 60,80
TITLE MODUL1
PUBLIC ALTPRO1
EXTERN SAY1:WORD
EXTERN SAY2:WORD
EXTERN TOPSON:WORD
.MODEL SMALL
.CODE
ALTPRO1 PROC FAR
MOV AX,SAY1
ADD AX,SAY2
MOV TOPSON,AX
RET
ALTPRO1 ENDP
END
```

Assembly ile Bilgisayarınızdaki Dosyalara Çabuk Ulaşın

```

model tiny
NFILES EQU 200 ;Max # files in display
FILESPEC EQU 82h ;Address of filespec in command tail
PATH EQU 81h ;Count of chars in path part of filespec here
NORMAL_FILE EQU 11100001b ;Attribute for a normal file
TAGGED EQU 80h ;Bit to set in attrib to indicate file tagged
DTA_ATTRIB EQU DTA+21 ;Offsets of elements in DTA
DTA_TIME EQU DTA+22
DTA_DATE EQU DTA+24
DTA_SIZE EQU DTA+26
DTA_NAME EQU DTA+30
FILE_TIME EQU bp ;Addrs of elements in 'fileRecords' after name
FILE_DATE EQU bp+2
FILE_SIZE EQU bp+4
INVERSE_VIDEO EQU 70h ;Black on white
NORMAL_VIDEO EQU 17h ;White on blue
COPYVAL EQU 1 ;Keys for Copy/Move/Delete/Rename operations
DELETEVAL EQU 2
MOVEVAL EQU 3
RENAMEVAL EQU 4
codeseg
BOF:
org 100h
Start:
mov sp, OFFSET stackEnd ;Initialize stack
sub ax, ax
push ax
mov [programSeg], cs ;Save current value of CS -> program segment
mov ah, 0Fh ;Find display memory
int 10h
mov di, 0B000h ;If mode 7 (= MDA or Herc), video seg=B000h,
cmp al, 7
je @@L1
mov di, 0B800h ;Otherwise video seg=B800h,
@@L1:
mov es, di ;Unless running under DesqView
mov cx, 'DE'
mov dx, 'SQ'
mov ax, 2B01h
int 21h
cmp al, 0FFh
je @@L2
mov ah, 0FEh
int 10h
mov di, es
@@L2:
mov [displaySegment], di

```

```

mov es, [2Ch] ;Find COMSPEC
sub di, di
@@L3:
mov si, OFFSET comspec
mov cx, 8
repe cmpsb
jne @@L3
mov [comspecOff], di
mov [comspecSeg], es
mov es, [programSeg]
NewSpec:
mov si, FILESPEC ;Examine command line filespec
mov di, si
cmp [byte FILESPEC-2], 1 ;If none, use "*.*)"
jbe @@L3
@@L1:
lodsb
cmp al, '*' ;Look for wildcard chars
je GetPathLength
cmp al, '?'
je GetPathLength
cmp al, 0Dh
jne @@L1
cmp [byte si-2], '\' ;If none, assume spec is a directory
jne @@L2
dec si ;If last char was not '\', add it
jmp @@L3
@@L2:
mov [byte si-1], '\'
@@L3:
mov ax, si ;Compute length of path minus filename
sub ax, di
mov [PATH], al
mov di, si
NewDir:
mov [keepSorted], 0 ;Directory is initially unsorted
mov si, OFFSET starDotStar ;Append '*.*',0
call CopyString
jmp SHORT Restart
NoRoomError:
mov dx, OFFSET noRoomMsg ;Not enough room to run S
Abort:
call BlankScreen
mov ah, 9
int 21h
mov ax, 4C00h ;Bye
int 21h
GetPathLength:
mov al, [byte FILESPEC-2] ;Null-terminate other filespecs

```

```

sub ah, ah
mov si, ax
add si, FILESPEC-1
mov [si], ah
mov cx, ax ;Count chars in path excluding file name:
std ;Search backward for last '\' or ':'
@@L1:
lods b
cmp al, '\'
je @@L2
cmp al, ':'
loopne @@L1
@@L2:
cld
mov [PATH], cx
Restart:
call HideCursor
mov ax, OFFSET files ;Cursor -> first file
mov [cursor], ax
mov [top], ax
Restart0:
mov bx, (EOF-BOF)/16+1 ;Resize memory to amount needed
mov ah, 4Ah
int 21h
jc NoRoomError
mov ah, 19h ;Get default drive for path/spec display
int 21h
add al, 'A'
mov di, OFFSET pathNSpec ;Store letter, ':'
stos b
mov ax, '\:'
stos w
mov si, di ;Get path and name of current directory
sub di, di
mov ah, 47h
int 21h
sub al, al ;Find end
mov cx, -1
repne scas b
dec di
mov ax, ' '+100h*' ' ;Add two spaces
stos w
mov si, FILESPEC ;Append filespec
call CopyString
;Get all file names matching filespec and set up tables
GetFileRecords:
mov dx, OFFSET DTA ;Set up DTA
mov ah, 1Ah
int 21h

```

```

mov dx, FILESPEC ;Get first file name
mov cl, 37h
mov ah, 4Eh
int 21h
jnc FileFound ;No files. Try a different filespec.
mov si, OFFSET NoFilesMsg
call Error
jmp NewFilespec
FileFound:
mov di, OFFSET fileRecords ;DI -> storage for file names
mov bx, OFFSET files ;BX -> array of files
sub bx, 2
StoreFileName:
add bx, 2 ;For all files that will fit,
cmp bx, (OFFSET files) + NFILES*2
jb @@L1
sub bx, 2
mov [last], bx
mov si, OFFSET tooManyMsg
jmp DoError
@@L1:
mov [bx], di ;Store pointer to status/filename in files[]
mov al, [DTA_ATTRIB] ;Store status byte
and al, 3Fh ;Top bit is used to indicate file is Marken
stosb
mov si, OFFSET DTA_NAME ;Copy file name form DTA to filename storage
call CopyString
inc di
mov si, OFFSET DTA_TIME ;Copy time, data and size
mov cx, 4
rep movsb
mov ah, 4Fh ;Neft filename
int 21h
jnc StoreFileName
mov [last], bx ;Save pointer to last file entry
mov al, [keepSorted] ;If returning form EXEC, need to resort files?
or al, al
jz DisplayFiles
jmp Sort0
;Main loop. Display files and wait for command.
DisplayFiles:
call BlankStatus ;Clear status line
mov si, OFFSET helpF1 ;Display help key
call DisplayString
mov bx, [cursor]
mov si, [bx]
lodsb ;Get attributes of file at cursor
mov [attrib], al ;Save attribute byte
call DisplayString ;Display name of highlighted file

```

```

mov bp, si ;Save pointer to time, data, size
test [attrib], 10h ;If a directory,
jz @@L7
mov di, 44 ; show '
' instead of file size
mov si, OFFSET dirMsg
call DisplayString
jmp SHORT @@L9
@@L7:
std ;File size, right justified
mov di, 56
mov ax, [FILE_SIZE]
mov dx, [FILE_SIZE + 2]
mov cx, 10000 ;Divide by 10000, show quotient|remainder
div cx
xchg ax, dx
or dx, dx
jz @@L8
mov cl, 4
call WriteDecimal
mov ax, dx
@@L8:
sub cl, cl
call WriteDecimal
@@L9:
std
mov di, 74 ;File data:
sub dx, dx
mov bx, [FILE_DATE] ;Year
mov al, bh
shr al, 1
add al, 80
sub ah, ah
mov cl, 2
call WriteDecimal
mov al, '/'
stosw
mov ax, bx ;Day
and ax, 1Fh
mov cx, 2
call WriteDecimal
mov al, '/'
stosw
mov ax, bx ;Month
mov cl, 5
shr ax, cl
and ax, 0Fh
mov cl, 2
call WriteDecimal

```

```

mov di, 86
mov bx, [FILE_TIME] ;File time:
mov ax, bx ;Minutes
mov cl, 5
shr ax, cl
and ax, 3Fh
mov cx, 2
call WriteDecimal
mov al, ':'
stosw
mov al, bh ;Hours
mov cl, 3
shr al, cl
sub ah, ah
sub cl, cl
call WriteDecimal
cld
mov di, 90
mov dl, [attrib] ;Display attribute letters
test dl, 1 ;Read-only
jz @@L3
mov al, 'R'
stosw
@@L3:
test dl, 2 ;Hidden
jz @@L4
mov al, 'H'
stosw
@@L4:
test dl, 4 ;System
jz @@L5
mov al, 'S'
stosw
@@L5:
test dl, 20h ;Archive
jz @@L6
mov al, 'A'
stosw
@@L6:
mov di, 100 ;Display path and filespec
mov si, OFFSET pathNSpec

call DisplayString
mov bx, [top]
mov di, 160
DisplayNext:
mov ah, NORMAL_VIDEO ;Set to inverse video if cursor line
cmp bx, [cursor]
jne @@L0

```

```

mov ah, INVERSE_VIDEO
@@L0:
cmp bx, [last] ;If done with files,
jle @@L0a
mov cx, 16 ;Blank out name area
jmp @@L7
@@L0a:
mov si, [bx] ;Get table entry for a file
lodsb ;Get status byte
test al, TAGGED ;If file has been tagged, display '>'
mov al, '>'
jnz @@L1
mov al, ''
@@L1:
stosw
mov cx, 9 ;In field of 9,
@@L2:
lodsb ;Display filename up to extension
cmp al, '.'
je @@L3
or al, al
jz @@L6
@@L2a:
stosw
loop @@L2
@@L3:
cmp cx, 9 ;Check for special cases of '.', '..'
je @@L2a
cmp [byte si-2], '.'
je @@L2a
mov al, '' ;Else pad with spaces out to 9 chars
rap stosw@@L4:
mov cx, 6 ;Display extension in field of 6
@@L5:
lodsb
or al, al
jz @@L7
stosw
loop @@L5
jmp SHORT @@L7
@@L6:
add cx, 6 ;Just pad with blanks if no extension
@@L7:
mov al, ''
rap stosw
cmp di, 4000 ;Stop at screenful
je GetCommand
cmp di, 3872
jb @@L8

```

```

sub di, 3808 ;Neft column
@@L8:
add di, 128 ;Neft row
add bx, 2
jmp DisplayNext
;Get command
GetCommand:
mov es, [programSeg]
mov ah, 8 ;Get keypress
sub ch, ch
@@L1:
inç ch
int 21h
or al, al
jz @@L1
mov ah, ch ;AH = 2 if aux code, 1 if plain ASCII
call ToUpper
mov di, OFFSET CommandKeys ;Look it up, get pointer to routine
mov cx, NCOMMANDS
repne scasw
jne InvalidCommand
add di, CommandAddrs-CommandKeys-2
mov bx, [cursor] ;SI -> file record for highlighted file
mov si, [bx]
jmp [word di] ;Jump to routine
InvalidCommand:
call Beep
jmp GetCommand
; Commands
Up:
sub bx, 2
jmp SHORT NewLine
Down:
add bx, 2
jmp SHORT NewLine
Left:
sub bx, 48
jmp SHORT NewLine
Right:
add bx, 48
jmp SHORT NewLine
PageUp:
sub bx, 238
jmp SHORT NewLine
PageDown:
add bx, 238
NewLine:
cmp bx, [last] ;Make sure cursor is still within bounds
jbe @@L1

```

```

mov bx, [last]
@@L1:
cmp bx, OFFSET files
jae @@L2
mov bx, OFFSET files
@@L2:
cmp bx, [top] ;Slide window if off screen
jae @@L3
mov [top], bx
jmp SHORT @@L4
@@L3:
mov ax, bx
sub ax, 238
cmp ax, [top]
jb @@L4
mov [top], ax
@@L4:
mov [cursor], bx
jmp DisplayFiles
Tag:
test [byte si], NOT NORMAL_FILE ;Only allow marking of normal files
jnz @@L1
xor [byte si], TAGGED
@@L1:
jmp DisplayFiles
Go:
mov [byte inputString], 0 ;Initialize to no user-entered command tail
Go0:
lodsb ;Get attribute byte
mov dx, si ;Join path and file name
mov si, PATH
mov di, OFFSET buffer
call Join
test al, 10h ;If a directory,
jz DoExec
ChangeDir:
mov dx, OFFSET buffer
mov ah, 3Bh ;Change to it
int 21h
mov [byte PATH], 0 ;No path now
mov di, FILESPEC ;Read in contents of new directory
jmp NewDir
DoExec:
mov si, dx ;Find extension
@@L1:
lodsb
cmp al, '.'
je @@L2
or al, al

```

```

jne @@L1
jmp SHORT @@L4
@@L2:
mov di, OFFSET extensions ;If .EXE, .COM or .BAT, execute it
mov dx, si
@@L3:
mov si, dx
mov cx, 3
repe cmpsb
je @@L5
sub al, al
repne scasb
cmp [byte di], 0
jne @@L3
@@L4:
mov si, OFFSET notExecMsg ;Else error, not an executable file
jmp DoError
@@L5:
call BlankScreen
mov dx, OFFSET buffer ;If .BAT, need COMMAND.COM, otherwise don't
cmp [byte di], 0
jne @@L6
mov si, OFFSET CC
label DoEdit near ;Edit function enters here
mov di, OFFSET EXECCmdLine+1
call CopyString ;Store '/c ' or '/c ' to command tail
mov dl, cl
mov si, OFFSET buffer ;Append path\file
call CopyString
add dl, cl
mov [EXECCmdLine], dl ;Store length, append CR
mov [byte di], 13
mov dx, [comspecOff]
mov ds, [comspecSeg]
@@L6:
cmp [byte cs:inputString], 0 ;If a command tail was entered,
jz DoDOS
push ds
mov ds, [cs:programSeg]
mov di, OFFSET EXECCmdLine+1
mov al, ' ' ;Add a space
stosb
mov si, OFFSET inputString ;Add command tail
call CopyString
inc cl
add [EXECCmdLine], cl
mov [byte di], 13
pop ds
DoDOS:

```

```

mov cx, bx
mov bx, (inputString-BOF)/16+1 ;Release unneeded memory
mov ah, 4Ah
int 21h
jc SysErr
call BlankScreen
push cx
mov [cs:temp], sp ;Do EXEC
mov bx, OFFSET EXECParams
mov ax, 4B00h
int 21h
mov bx, cs ;Restore critical registers
mov ds, bx
mov es, bx
mov ss, bx
mov sp, [temp]
pop bx
jnc @@L2
mov si, OFFSET sysErrMsg ;EXEC error
cmp al, 8 ;If return code = 8, no room
jne @@L1
mov si, OFFSET noExecRoomMsg
@@L1:
jmp DoError
@@L2:
mov [byte EXECCmdLine], 0 ;Tidy up
call HideCursor
jmp Restart0
SysErr:
mov si, OFFSET sysErrMsg
DoError:
call Error
jmp DisplayFiles
GoCL:
mov si, OFFSET commandTailMsg ;Prompt for command tail
call Query
mov si, [bx]
jmp Go0
DOS:
call BlankScreen
mov dx, [comspecOff]
mov ds, [comspecSeg]
jmp DoDOS
Edit:
lea dx, [si+1] ;Join path and file name
mov si, PATH
mov di, OFFSET buffer
call Join
mov si, OFFSET editor ;Invoke editor

```

```

mov [byte inputString],0
jmp DoEdit
Copy:
mov [byte CMDR], COPYVAL ;Set Copy/Move/Delete/Remove key to Copy
jmp SHORT DoCopy
Delete:
mov [byte CMDR], DELETEVAL ;Set Copy/Move/Delete/Remove key to Delete
jmp SHORT DoCMDR
Move:
mov [byte CMDR], MOVEVAL ;Set Copy/Move/Delete/Rename key to Move
DoCopy:
mov si, OFFSET DestMsg ;If Copy or Move, prompt for destination
call Query
mov di, si
mov [byte si], '\ ' ;Append '\ '
inc si
mov [temp], si
cmp [byte CMDR], MOVEVAL
jne DoCMDR
mov ax, [word FILESPEC] ;If Move to same drive,
mov dx, [word inputString]
cmp ah, ':'
je @@L1
cmp dh, ':'
jne @@L2
@@L1:
cmp ax, dx
jne DoCMDR
@@L2:
mov [byte CMDR], RENAMEVAL ; do Rename instead (much faster)
DoCMDR:
mov bp, OFFSET files - 2 ;For each file
CMDRNext:
add bp, 2
cmp bp, [last]
jbe @@L0
jmp GetFileRecords
@@L0:
mov si, [bp] ;Skip if not tagged
lodsb
test al, TAGGED
jz CMDRNext
xor [byte si-1], TAGGED ;Else untag
mov dx, si
mov si, PATH ;Source path\filename -> sourceFileSpec
mov di, OFFSET sourceFileSpec
call Join
cmp [byte CMDR], DELETEVAL ;If not Deleting
je @@L4

```

```

mov si, dx ;Append current file's name to destination path
mov di, [temp]
call CopyString
cmp [byte CMDR], RENAMEVAL ;If Rename, do it
jne @@L2
label DoRename near
mov dx, OFFSET sourceFileSpec
mov di, OFFSET inputString
mov ah, 56h
int 21h
jc CantOpen
@@L1:
jmp CMDRNext
@@L2:
mov dx, OFFSET sourceFileSpec ;Copy or Move: open source, dest files
mov ax, 3D00h
int 21h
jc CantOpen
mov [sourceHandle], ax
sub cx, cx
mov dx, OFFSET inputString
mov ax, 3C00h
int 21h
jc CantRename
mov [destHandle], ax
@@L3:
mov bx, [sourceHandle] ;Read a bufferful
mov cx, 1024
mov dx, OFFSET buffer
mov ah, 3Fh
int 21h
jc ReadError
mov bx, [destHandle] ;Write it
mov cx, ax
mov ah, 40h
int 21h
jc WriteError
cmp cx, 1024 ;Loop until done
je @@L3
mov ah, 3Eh ;Close files
mov bx, [sourceHandle]
int 21h
mov bx, [destHandle]
int 21h
cmp [CMDR], MOVEVAL ;If Move, now do Delete
jne @@L1
@@L4:
mov bx, OFFSET sourceFileSpec ;Delete file
mov ah, 41h

```

```

int 21h
jnc @@L1
CantOpen:
mov si, OFFSET cantOpenMsg
jmp DoError
CantRename:
mov si, OFFSET cantRenameMsg
jmp DoError
ReadError:
mov si, OFFSET readMsg
jmp DoError
WriteError:
mov si, OFFSET writeMsg
jmp DoError
NewFilespec:
mov si, OFFSET newSpecMsg ;Prompt for new filespec
call Query
inc al
mov [FILESPEC-2], al ;Store count of chars
mov si, OFFSET inputString
mov di, FILESPEC
call CopyString ;Copy new filespec to command tail area
mov [byte di], 0Dh ;Append CR
jmp NewSpec ;Process new filespec
Rename:
lea dx, [si + 1] ;Join path and current name
mov si, PATH
mov di, OFFSET sourceFileSpec
call Join
mov si, OFFSET newNameMsg ;Prompt for new name of file or directory
call Query
jmp DoRename
Drive:
mov si, OFFSET newDriveMsg ;Prompt for letter of drive to change to
call QueryChar
sub al, 'A'
mov dl, al
mov ah, 0Eh
int 21h
cmp dl, al ;Try again if that drive doesn't exist
jb @@L1
mov si, OFFSET badDriveMsg
jmp DoError
@@L1:
jmp Restart
Sort:
;Sort algorithm:
; 1) make up array of records {pointer to field to sort | tag},
;one for each

```

```

; file, in 'buffer'.
; 2) bubble-sort these records
; 3) copy file record pointers in 'files' to 'buffer' in
;order of sorted tags
; 4) copy file record pointers back to 'files' in new order
mov si, OFFSET sortMsg ;Prompt for sort field
call QueryChar
cmp al, 'N' ;Check for legal sort field option
je Sort0
cmp al, 'E'
je Sort0
cmp al, 'D'
je Sort0
mov si, OFFSET genErrorMsg
jmp DoError
Sort0:
mov [keepSorted], al ;Remember for later resorting after EXEC
mov dl, al
sub dh, dh < ;DH = tag (position of file in current order)
mov di, OFFSET buffer
mov bx, OFFSET files
@@L1:
mov si, [bx] ;Find field to sort: get pointer to record
@@L1a:
inc si
mov cx, si
cmp dl, 'N' ;If Name, already pointing at it
je @@L4
sub ah, ah ;If data, find null at end of name
cmp dl, 'E' ;If Ext find '.' or end of name
jne @@L2
cmp [byte si], '.' ;'.' and '..' are special cases
je @@L1a
mov ah, '.'
@@L2:
lodsb
or al, al
je @@L3
cmp al, ah
jne @@L2
@@L3:
dec si ;If Ext, back up to '.' or null
cmp dl, 'E'
je @@L4
add si, 3 ;If data, advance to data field
@@L4:
mov ax, si ;Store pointer to field to sort
stosw
mov al, dh ;Store tag

```

```

stosb
inç dh ;Bump tag
add bx, 2 ;Loop until no more files
cmp bx, [last]
jbe @@L1
DoSort:
lea bp, [di-3] ;BP -> last
push bp
@@L0:
mov bx, OFFSET buffer ;Do bubble sort
@@L1:
mov si, [bx]
cmp [word si-1], '.' ;Leave '.' and '..' alone
je @@L4
mov di, [bx+3]
cmp dl, 'D' ;If sorting Dates, compare one word
jne @@L2
cmpsw
jmp SHORT @@L3
@@L2:
mov cx, -1 ;Else compare bytes until not equal
repe cmpsb
@@L3:
jbe @@L4 ;If first field > second
mov ax, [bx] ;Exchange field pointers and tags
xchg ax, [bx+3]
mov [bx], ax
mov al, [bx+2]
xchg al, [bx+5]
mov [bx+2], al
@@L4:
add bx, 3 ;Loop until no more files this pass
cmp bx, bp
jb @@L1
sub bp, 3
cmp bp, OFFSET buffer ;Loop until no more passes
jne @@L0
OrderByTags:
mov di, OFFSET buffer ;Arrange file pointers in order of tags
mov si, OFFSET files
pop bp
@@L1:
mov bl, [di+2] ;Get tag
sub bh, bh
add bx, bx
mov ax, [bx+si] ;Get file ptr associated with that tag
stosw ;Store in place of field pointer in sort buffer
inç di
cmp di, bp

```

```

jbe @@L1
mov si, OFFSET buffer
mov di, OFFSET files
@@L2:
movsa ;Copy file pointers back in new order
inc si
cmp di, [last]
jbe @@L2
jmp DisplayFiles
Help:
call BlankStatus ;Display help on status line
mov si, OFFSET helpMsg
call DisplayString
mov ah, 8
int 21h
jmp DisplayFiles
Exit:
call BlankScreen
mov ax, 4C00h ;Bye
int 21h
; Subroutines
WriteDecimal:
;Display a decimal number in inverse video, writing digits
;backwards form right.
; IN: AX = number, CL = field width with leading 0s (no leading
;0s if CL = 0)
; ES:DI -> Video RAM where rightmost digit will go
; OUT: AH = INVERSE_VIDEO
;USED: AL, CX
push bx
push dx
mov bx, 10
sub ch, ch
@@L1:
sub dx, dx ;Get a digit
div bx
xchg ax, dx ;Write it to display, right to left
add ax, '0' + INVERSE_VIDEO*100h
stosw
xchg ax, dx
cmp cx, 0 ;If CX > 0, loop even if AX = 0
jg @@L2
or ax, ax ;Else loop only if AX > 0 (more digits left)
jz @@L3
@@L2:
loop @@L1
@@L3:
pop dx
pop bx

```

```

mov ah, INVERSE_VIDEO
Ret1:
ret
DisplayString:
;Display string on status line.
; IN: SI -> string (null-terminated),
;ES:DI -> status line, AH = attribute
; OUT: DI is advanced past end of string
;USES: AL SI
@@L1:
lodsb
or al, al
jz Ret1
stosw
jmp @@L1
Error:
;Beep, display string on status line and wait for keypress (any key)
; IN: SI -> string
; OUT: ES -> program segment
;USED: AX
call Beep
call BlankStatus
call DisplayString ;Display error string
mov si, OFFSET ErrorMsg ;Display 'Press any key' message
call DisplayString
mov ah, 8
int 21h
mov es, [programSeg]
ret
Query:
;
;Prompt for string input on status line
; IN: SI -> message
; OUT: SI -> null at end of ASCIIZ string input,
;AX = length (excluding null)
;USED: none
push bx
push cx
push dx
push di
push es
call BlankStatus
call DisplayString
mov dx, di ;Set cursor to end of printed string
add dx, 2
shr dl, 1
sub dh, dh
mov ah, 2
sub bx, bx

```

```

int 10h
mov cx, 80 ;Get input
mov dx, OFFSET inputString
mov si, dx
mov ah, 3Fh
int 21h
sub ax, 2
add si, ax
mov [byte si], 0 ;Null-terminate it
call HideCursor ;Hide cursor
pop es
pop di
pop dx
pop cx
pop bx
ret
QueryChar:
;
;Prompt for single character input
; IN: SI -> message
; OUT: AL = character (lower case converted to upper)
;USED: AH
push es
call BlankStatus
call DisplayString ;Display string
mov ah, 8 ;Get char
int 21h
call ToUpper
pop es
ret
BlankStatus:
;
;Clear top line of display to inverse video
; IN: none
; OUT: ES = video segment, DI = 0
;USED: AX CX
mov ax, [displaySegment]
mov es, ax
sub di, di
mov ax, '' + INVERSE_VIDEO*100h
mov cx, 80
rep stosw
sub di, di
ret
Join:
;
;Copy counted path, then ASCIIZ file name to buffer and null-terminate
; IN: SI -> path, DX -> file, DI -> destination
; OUT: DI -> null at end of copied string,

```

```

;CX = total chars excluding null
;USED: none
push ax
push si
lodsb ;Get count of chars in path (a counted string)
sub ah, ah
mov cx, ax
rap movsb ;Copy it to destination
mov cx, ax
mov si, dx ;Now copy file name (null-terminated)
call CopyString
add cx, ax ;Sum string counts -> CX
mov [byte di], 0 ;Null-terminate the result
pop si
pop ax
ret
CopyString:
;
;Copy null-terminated string.
; IN: SI -> string, DI -> destination
; OUT: CX = length (excluding null), DI -> terminating
;null of copied string
;USED: AL
mov cx, -1
@@L1:
lodsb ;Copy string
stosb
or al, al ;Until null at end is encountered
loopnz @@L1 ;Accumulate count of chars
neg cx ;Adjust count
sub cx, 2
dec di ;DI -> terminating null
ret
HideCursor:
;
;Move cursor off bottom of screen
; IN: none
; OUT: none
;USED: AH BH DX
mov dx, 1900h
sub bh, bh
mov ah, 2
int 10h
ret
Beep:
;
;Output a bell char
; IN: none
; OUT: none

```

```

;USED: AH DL
mov dl, 7
mov ah, 2
int 21h
ret
BlankScreen:
;
;Clear screen and home cursor
; IN: none
; OUT: none
;USED: AX
push bx
push cx
push dx
push es
push di
mov es, [cs:displaySegment] ;Blank screen
sub di, di
mov ax, '+' + NORMAL_VIDEO * 100h
mov cx, 25 * 80
rep stosw
sub dx, dx ;Put cursor at upper L hand corner
sub bh, bh
mov ah, 2
int 10h
pop di
pop es
pop dx
pop cx
pop bx
ret
ToUpper:
;
;Convert lower to upper case
; IN: AL = char
; OUT: AL = char
;USED: none
cmp al, 'a'
jb @@L1
cmp al, 'z'
ja @@L1
add al, 'A' - 'a'
@@L1:
ret
;Data
;Command dispatch table: aux,2 or ASCII,1 paired to
;command routine addresses
commandKeys db 72,2, 80,2, 75,2, 77,2, 73,2, 81,2, 'T',1, 13,1, 'C',1
db 'D',1, 'M',1, 'R',1, 'E',1, 'F',1, 'V',1, 'S',1

```

```

db 60,2, 10,1, 59,2, 27,1
commandAddr dw Up, Down, Left, Right, PageUp, PageDown, Tag, Go, Copy
dw Delete, Move, Rename, Edit, NewFilespec, Drive, Sort
dw DOS, GoCL, Help, Exit
NCOMMANDS EQU (commandAddr-commandKeys)/2
;Strings
helpMsg db 'Copy Delete Edit Filespec Move Ren Sort '
db 'Tag driVe Enter=cd/run F2=DOS Esc=exit',0
helpF1 db 'Help F1',186,0
noFilesMsg db 'No matching files or invalid path',0
noRoomMsg db 'Out of room$'
cantRenameMsg db "Can't rename to that",0
badPathMsg db 'Bad path',0
sysErrMsg db 'System error',0
cantOpenMsg db "Can't open file",0
writeMsg db 'Write error',0
readMsg db 'Read error',0
notExecMsg db 'Not a directory or executable file',0
badDriveMsg db "Drive doesn't exist",0
genErrMsg db 'Error',0
tooManyMsg db 'Too many files',0
noExecRoomMsg db 'Not enough memory',0
ErrorMsg db '. Press any key.',0
destMsg db 'Where to?',0
newSpecMsg db 'New filespec:',0
newDriveMsg db 'New drive:',0
sortMsg db 'Sort on: Name Ext data',0
newNameMsg db 'New name:',0
commandTailMsg db 'Command tail:',0
dirMsg db '
',0
extensions db 'EXECOMBAT',0
starDotStar db '*.*',0
editor db '/C E ', 12 dup (0)
cc db '/C ',0
comspec db 'COMSPEC='
;EXEC function parameter block
EXECParams dw 0
EXECCmdLineOff dw OFFSET EXECCmdLine
programSeg dw 0
dw -1, -1, -1, -1
EXECCmdLine db 0, 80 dup (?)
sourceFileSpec EQU EXECCmdLine ;Second use for this space during Copy etc.
;Variables, buffers
dw 128 dup (?) ;Stack (here for protection during EXEC)
stackEnd:
keepSorted db ? ;Holds sort subcommand char if this dir sorted
CMDR db ? ;Key for Copy/Move/Delete/Rename actions
attrib db ? ;File attribute byte

```

sourceHandle dw ? ;Source handle for Copy, etc.
 destHandle dw ? ;Destination handle for Copy etc.
 cursor dw ? ;Position in 'files' of highlighted file
 last dw ? ;Position in 'files' of last file in directory
 top dw ? ;Position in 'files' of file at top of screen
 temp dw ? ;Holds SP during EXEC, other uses
 comspecSeg dw ? ;Segment of environment
 comspecOff dw ? ;Offset of 'C:\COMMAND.COM' in environment
 displaySegment dw ? ;Segment of display RAM
 pathNSpec db 80 dup (?) ;Default path and current filespec for display
 DTA db 64 dup (?) ;Disk Transfer Area
 inputString db 80 dup (?) ;String returned by 'Query', other uses
 files dw NFILES dup (?) ;Array of pointers to file records
 fileRecords db 80*NFILES dup (?) ;File records: attrib/name/time/data
 buffer db 1024 dup (?) ;Buffer for Copy, etc., other uses
 EOF:
 end Start

Sırasız Bir Listede En Büyük ve Küçüğün Bulunması

```

PAGE 60 , 80
TITLE MAX MIN BUL
.MODEL SMALL
.CODE
ORG 100H
BAS: JMP SHORT ANA
LISTE DW 2245,4425,40008,22012,34209,36377
ANA PROC NEAR
CLD (SOLDAN SAGA BAK)
PUSH CX
PUSH DI
MOV CX , 06
MOV DI , OFFSET LISTE
MOV BX , ES : [DI]
MOV AX , BX
ARA ADD DI , 2
CMP ES : [DI] , BX
JAE MAKSI
MOV BX,ES:[DI]
JMP TARA
MAKSI: CMP ES,[DI],AX
JBE TARA  MOV AX,ES:[DI]
TARA: LOOP ARA
POP DI
POP CX
MOV AH,4CH
INT 21H
ANA ENDP
END BAS
  
```

KAYNAKLAR

- **Nurettin Topaloğlu, Mikroişlemci mimarisi ve Assembly Dili**
Seçkin yayıncılık 2001
- **Abel Peter, IBM PC Assembly Language and programming, international 5th edition, prentice hall, 2000**
- **GümüşKaya Haluk, Mikroişlemciler ve bilgisayar intel ailesi ve IBM PC, alfa yayınları 2000**
- **Scanlo leo, 8086-8088-80286 Assembly Language, Brady book, 1988**
- **Dreiver George, introduction to programming in Assembly language (IBM PC) west puclishing co.1993**
- **Hyde randal, the art of Assembly language and HLA, 1996**
- **www.sdu.com.tr**
- **www.selcuk.com.tr**
- **www.afyon.com.tr**