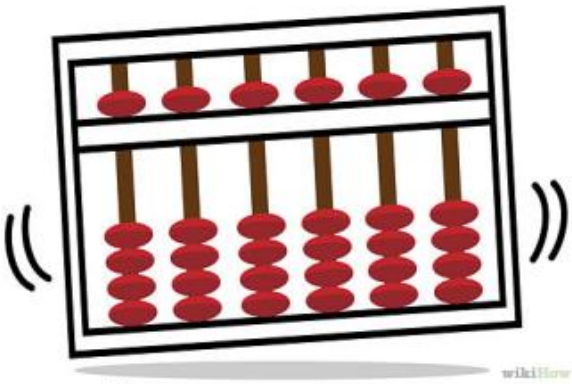
Bilgisayar Organizasyonu ve Mimarisi

Bir zamanlar

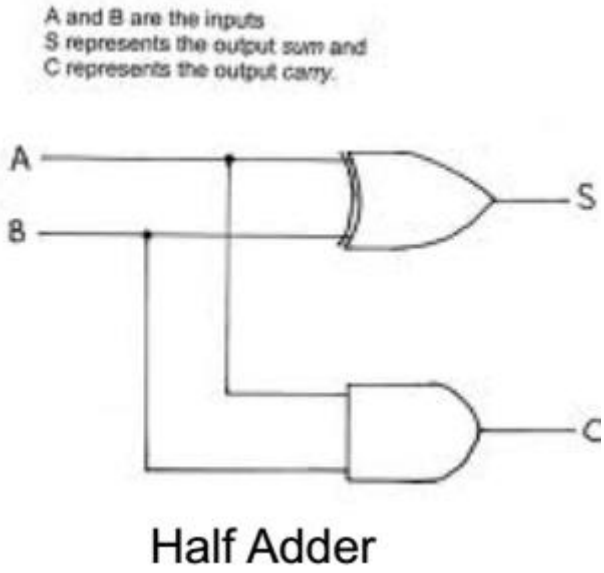


Abacus

17. Yüzyıl (Dişli Çarklar / Makineler)

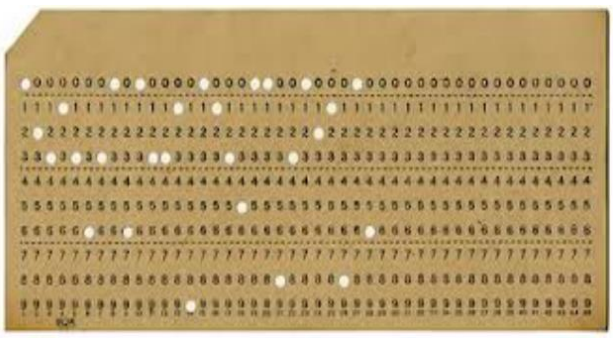
Pascaline

20. Yüzyıl (Elektronik)



Half Adder

Peki Hafıza?



Delikli Kart

Şimdi nasıl oluyor?

Intel Machine Language		Assembly Language

A1 00000000
F7 25 00000004
03 05 00000008
E8 00500000
	=	mov eax, A
		mul B
		add eax, C
		call WriteInt

C++ language

cout<<(A*B+C)

Neden Assembly?

- Bilgisayar 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.
- 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.

Katmanlı Mimari (Layered Architecture)

Bilgisayarlar katmanlardan oluşmuş karmaşık yapılardır. Bu katmanların karmaşıklıklarını soyutlanır(abstraction).

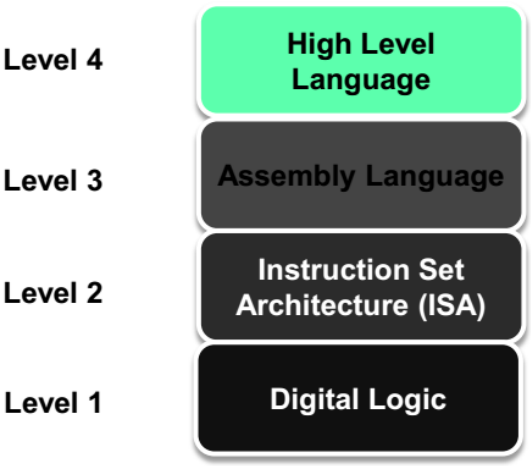
Bunun için katmanlı programlama dilleri kullanırız.

Programın yürütülmesi

Yorumlama, Derleme (Çeviri)

Her CPU kendi "komut kümesi" için (ISA, Komut Seti Mimarisi, programlanmış ikili dil) yerleşik bir çeviriciye sahiptir.

Makine Seviyeleri



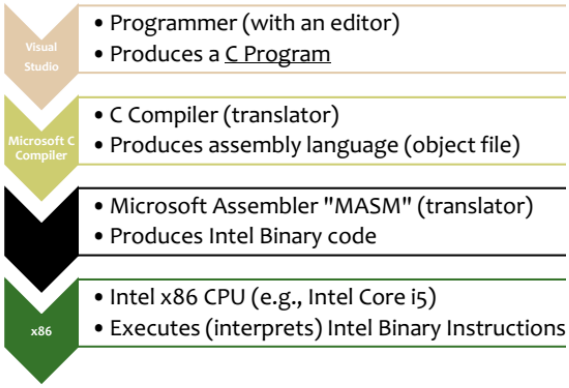
1: Yüksek Düzeyli Bir Dil (C, C ++, Fortran, Cobol) Assembly diline derlenir (tercüme edilir).

2: Assembly dili (belirli bir CPU için) ikili makine dili olarak birleştirilir.

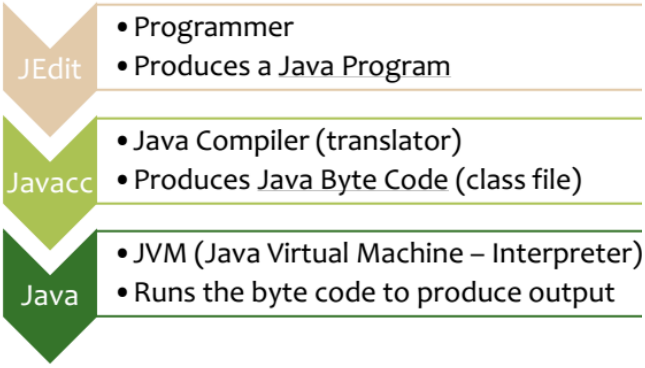
3: İkili makine dili, bilgisayardaki CPU'lardan biri tarafından yorumlanır.

4: İşlemci (Intel, AMD vb.), Yorumlamayı yapmak ve sonuçları üretmek için dijital mantık devrelerini kullanır.

C++ Derlenmesi



Java Derlenmesi



Bağlama ve Yükleme(Linking and Loading)

Birleştirme (Assembling) (MASM çalıştıran) gerçekte uygulanabilir bir program oluşturmaz, Gerçekleştirilmesi gereken en az 4 temel adım vardır:

Assembling – kodu ikili haline çevir
Linking – birlikte tüm parçaları birleştir ve isimlerini çöz
Loading – programı hafızaya taşı
Execution – Programı çalıştır

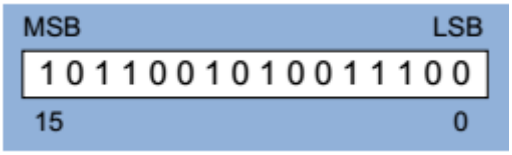
Assembly Dili

Belirli bir CPU ailesi için tasarlanmıştır (yani, Intel x86).
Genellikle her talimat(mnemonic), tek bir ikili CPU talimatına eşdeğerdir.

Mantık, bilgi işlemin temel dilidir.

Verilerin Temsil Edilmesi

Bilgisayarlar ikili veriler ile çalışır(bazen 8'li veya 16'lı sayı sistemleri ile de temsil edilirler).
İkili Sayılar: 1/0 'dan oluşan sayılardır.

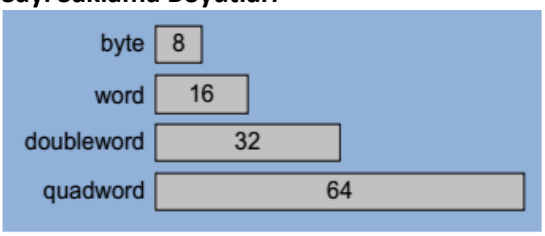


MSB – most significant bit (Çok Değerlikli Bit)
LSB – least significant bit (Az Değerlikli Bit)

SAYI SİSTEMLERİ VE DÖNÜŞÜMLERİ

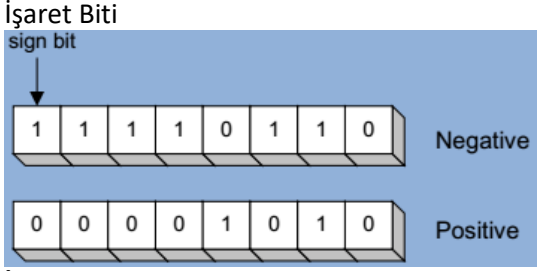
.....

Sayı Saklama Boyutları



Storage Type	Max Value	Power of 2
Unsigned byte	255	2^8-1
Unsigned word	65,535	2^{16}-1
Unsigned doubleword	4,294,967,295	?

İşaret Biti



İkiye Tamamlayıcı

Starting value:	00000001
Step1: reverse the bits	11111110
Step 2: add 1 to value from step 1	11111110 + 00000001
Sum: two's complement representation	11111111

İşaretli Binary değeri Decimal Yapmak

Starting value:	11110000
Step1: reverse the bits	00001111
Step 2: add 1 to value from step 1	00010000
Convert to decimal and add (-) sign	-16

Maksimum ve Minimum Değerler

Storage Type	Range(Min-Max)	Power of 2
Unsigned byte	0 to 255	0 to (2^8-1)
Singed byte	-128 to +127	-2^7 to (2^7-1)
Unsigned word	0 to 65,535	0 to (2^{16}-1)
Signed word	-32,768 to +32,767	-2^{15} to (2^{15}-1)

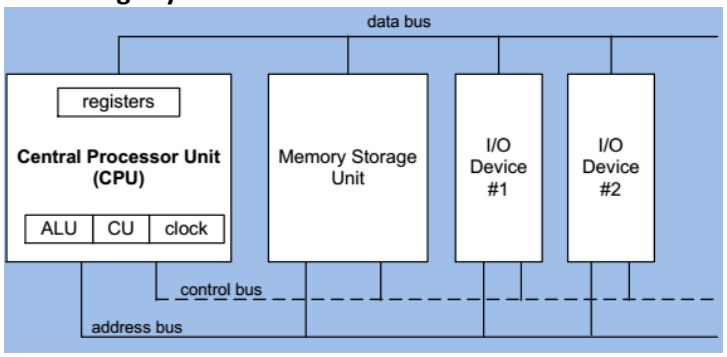
ASCII Tablo

	0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1 XON	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3 XOFF	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	del

BOOLEAN ARİTMETİĞİ

...

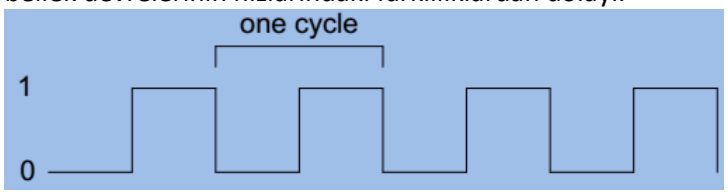
Temel Bilgisayar Tasarımı



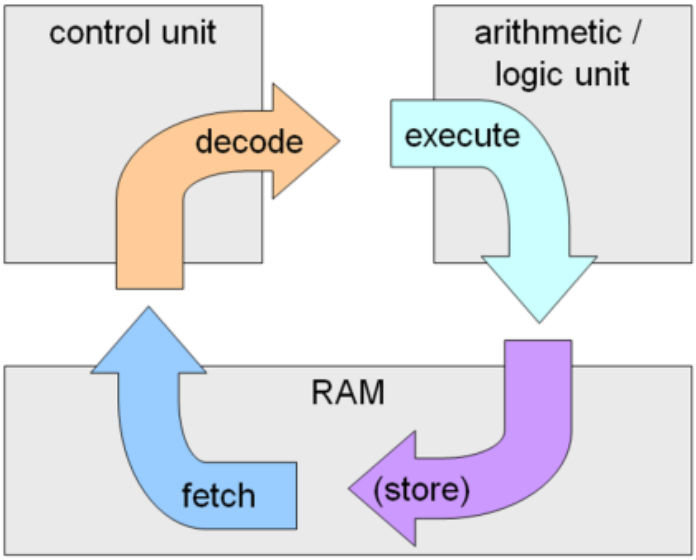
Clock

Tüm CPU ve bus işlemlerini senkronize eder.
Clock cycle tek bir işlemin zamanını belirler.

Bir makine komutu çalıştırmak için en az bir çevrim gerektirir. Bellek erişimi gerektiren komutlar genellikle boş çevrimlere ihtiyaç duyarlar. **(Bekleme Durumu):** CPU, sistem veri yolu ve bellek devrelerinin hızlarındaki farklılıklardan dolayı.



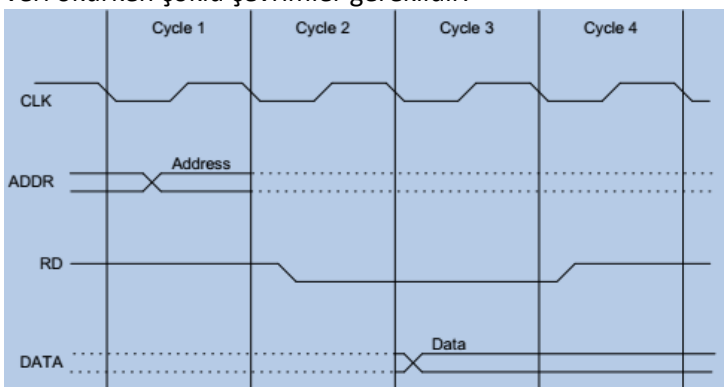
Komut Çalıştırma Döngüsü



- **Fetch instruction:** Komutlar alınır ve CPU 'ya getirilir.
- **Decode: Komutlar çözülür ve anlamlandırılır.**
- **Fetch operands:** İşlenecek veriler alınır. Memory /registers (dahili)
- **Execute:** Komut çalıştırılır
- **Store output:** Sonuç kayıt edilir.

Bellekten Okuma

Bellek CPU'dan çok daha yavaş tepki verdiği için bellekten veri okurken çoklu çevrimler gereklidir.



- Adres, adres yoluna yerleşir
- Read Line(RD) 0 yapılır
- CPU, hafızanın yanıtlanması için bir çevrim bekler.
- Okunan Hat (RD), verinin veri yolunda olduğunu belirten 1'e gider.

Ön Bellek

CPU nun hem içinde hemde dışında bulunan pahalı static RAM çeşididir. Bir program bir veri bloğu ilk okurken, önbellekte bir kopya bırakır. Programın aynı verileri aynı anda okuması gerekiyorsa, önbellekteki verileri arar.

Cache hit: Okunacak veri cache bellekte halihazırda var ise.

Cache miss: Okunacak veri cache bellekte yok ise.

Multitasking

OS aynı anda birden fazla programı çalıştırabilir. Aynı program içinde birden çok iş parçacığı(thread) yürütülebilir. Zamanlayıcı(Scheduler) yardımcı programı, çalışan her program için belli bir CPU zamanı atar. Görevlerin hızlı değiştirilmesi sağlanır. Bu sayede kullanıcı tüm programların aynı anda çalıştığı hissine girer. Bunun için CPU görev değiştirmeyi desteklemelidir (context switching).

Yürütme: OS Rolü

Bir programı çalıştırdığınızda OS sizin için birçok şey yapar.

- İkili yürütülebilir dosyayı diskte bulur
- Dosyayı diskten belleğe taşımak için "yükleyiciyi" çağırır ve gerekirse tüm adresleri çözümler
- Herhangi bir çalışma zamanı DLL'ini bulur ve bağlar.

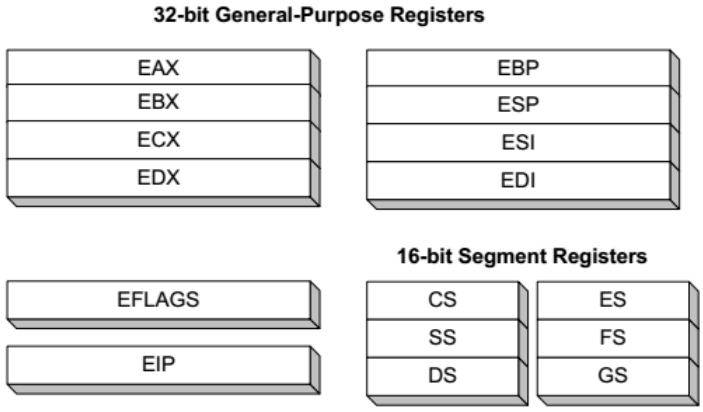
Yürütme Esnasında OS

- Tüm adresleri çözer
- Çalışma süresini tahsis eder (time-slice for multitaskin-çoklu görev için zaman dilimi)
- Aygıtlarla IO isteklerini yönetir

Uygulama bittikten sonra temizlenir

Temel Program Yürütme Registers

Kayıt birimleri **Registers**, doğrudan CPU'nun içinde yüksek hızlı depolama yerleridir. Geleneksel bellekten çok daha hızlı çalışması için tasarlanmıştır.

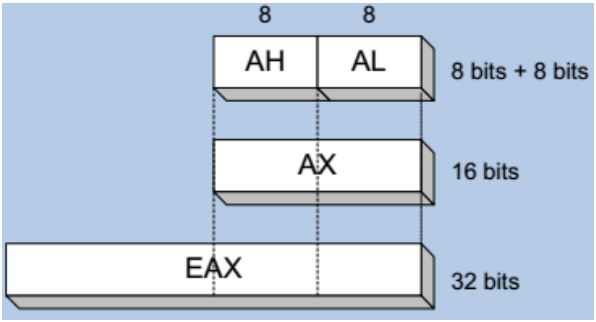


Register Tipleri

- General purpose registers, Segment registers, Processor status flags register, Instruction pointer

General Purpose Registers

Bu kaydediciler öncelikli olarak aritmetik ve veri hareketi için kullanılır. 8, 16 ve 32 bitlik isimleri vardır.



EAX, EBX, ECX ve EDX şeklinde adlandırılır.

32-bit	16-bit	8-bit (high)	8-bit (low)
EAX	AX	AH	AL
EBX	BX	BH	BL
ECX	CX	CH	CL
EDX	DX	DH	DL

Index & Base Registers

Geriye kalan genel amaçlı kayıtçılar düşük yarıları için sadece 16 bitlik bir ada sahiptir.

32-bit	16-bit	
ESI	SI	Source Index
EDI	DI	Destination Index
EBP	BP	Stack Base Pointer
ESP	SP	Stack Pointer

İstenilen Kayıtçının Kullanımı

Genel Amaçlı

▪ EAX – accumulator	▪ ECX – loop counter
▪ ESP – stack pointer (TOS)	▪ ESI, EDI – index registers
▪ EBP – extended (stack) frame (base) pointer	

Segment

▪ CS – code segment	▪ DS – data segment
▪ SS – stack segment	

Özel Registers

EIP: instruction pointer(Komut işaretçisi) bir sonraki komutun adresini gösterir.

EFLAGS: Durum ve kontrol bayrakları olarak ayrılır. Her bayrak tek bir ikili bitten oluşur.

Durum Bayrakları:

- **Carry:** İşaretsiz aralık dışı
- **Overflow:** İşaretli aralık dışı
- **Sign:** Sonuç negatif
- **Zero:** Sonuç sıfır

```
                    ; initially, assume ZF = 0
mov   AL,55H        ; ZF is still zero
sub   AL,55H        ; result is 0
                    ; ZF is set (ZF = 1)
mov   CX,0          ; ZF remains 1
inc   CX            ; result is 1
                    ; ZF is cleared (ZF = 0)
```

32-Bit registers	Name	16- and 8-bit sub-registers	Brief description and/or primary use
eax	Accumulator	ax,ah,al	Arithmetic and logic
ebx	Base	bx,bh,bl	Arrays
ecx	Counter	cx,ch,cl	Loops
edx	Data	dx,dh,dl	Arithmetic
esi	Source index	si	Strings and arrays
edi	Destination index	di	Strings and arrays
esp	Stack pointer	sp	Top of stack
ebp	Base pointer	bp	Stack base
eip	Instruction pointer	ip	Points to next instruction
eflags	Flag	flags	Status and control flags

Intel CPU Tarihi

Intel 8086 (**1978**)/8088 (**1979**)

İlk defa IBM-PC de kullanıldı 8088 (**1980**)

1 MB adreslenebilir RAM, 16-bit kaydediciler(registers)

16-bit veri yolu (8088 için 8-bit), $3000 - $6000 USD (8088)

Intel IA-32 Ailesi Tarihi

Intel 80386 (**1985**)

4 GB adreslenebilir RAM, 32-bit kaydediciler(registers), sayfalama(paging) (sanal bellek), Intel 80486 (**1989**), pipelining komut çalıştırma, Pentium – P5 (**1993**),superscalar (Çoklu ALU)

64-Bit İşlemciler

Intel64 Mode

64-bit doğrusal adres alanı

Intel: Pentium Extreme, Xeon, Celeron D, Pentium D, Core 2, and Core i3, i5, i7..

İki alt modda çalışır:

Eski 16 ve 32 bit uygulamalar için uyumluluk modu, 64 bitlik Mod ile 64 bitlik adresler ve komutlar kullanır.

İşlemci Teknolojileri:

Pipelined: İsmini boru hatlarının işleyişinden alan yaklaşım, kısaca bir işlem borunun sonundayken, borunun başından yeni bir işin konulabileceğini anlatmaktadır. Buna göre örneğin bir işin (process) çalışması için 4 farklı safhadan geçmesi gerekiyor olsun: Fetch (almak), Decode (algılamak), Execute (çalıştırmak), Store (saklamak).

Bu dört safhanın her birisi her işlem için tekrarlanacaktır. Yani örneğin P1 isimli işlem sırasıyla Fetch, Decode, Execute ve store aşamalarından geçecektir.

Zamanlama	Fetch (almak)	Decode (algılamak)	Execute (çalıştırmak)	Store (saklamak)
T1	P1			
T2	P2	P1		
T3	P3	P2	P1	
T4	P4	P3	P2	P1
T5	P5	P4	P3	P2

Superscalar bir işlemcide her birisinin kendisine ait pipeline'ı olan iki ya da daha fazla fonksiyon ünitesi yer alabilir. Böyle bir işlemci birkaç komutu birden paralel olarak işletebilir.

Mesela iki aritmetik birimi olan superscalar makine iki çift sayıyı aynı anda toplayabilir. Bu makine iki çıkışlı makine olarak adlandırılır. RISC komut kümesi supersckalar mimariye tam uyumludur.

HyperThreading: İki görev aynı anda tek bir işlemci üzerinde yürütülür. Hyper-Threading teknolojisi, tek bir fiziksel işlemcinin çok sayıda komut zincirini eş zamanlı olarak işlemesi ile performans artışı sağlamasıdır. Hyper-Threading teknolojisine sahip olan bir işlemci, mantıksal olarak iki adet işlemciden oluşmaktadır. Her bir işlemci fiziksel olarak aynı chip üzerinde bulunmasına rağmen farklı komut zincirlerini işleyebilir. Geleneksel iki farklı fiziksel işlemci kullanan sistemlerin aksine Hyper-Threading teknolojisinde, mantıksal işlemciler tek bir işlemci kaynağını (sistem veri yolu, bellek) paylaşırlar. Bu yüzden Hyper-Threading mimarisine sahip bir işlemci, işletim sistemine iki işlemcili bir sistem gibi görünmesine rağmen iki gerçek fiziksel işlemcinin sağladığı performansı vermeyecektir.

Dual Core processing: bir işlemci üzerinde 2 adet çekirdek (core)'in yerleştirilerek bu çekirdeklerin birbirinden bağımsız çalışabildiği x86 mimarisi mikroişlemciler için kullanılan bir teknolojidir. Her işlemci kendi kaynaklarına sahiptir.

CISC ve RISC:

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.** 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.

- Geniş komut seti
- Üst düzey karmaşık işlemler (daha az komut)
- Mikrokod çeviricisi gerektirir

CISC mimarisinin karakteristik iki özelliğinden birisi, **değişken uzunluktaki komutlar**, diğeri ise **karmaşık komutlardır**.

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.

RISC (Reduced Instruction Set Computer) 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. 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.

- Bütün komutlar tek bir çevrimle çalıştırılmalıdır.
- Belleğe sadece "load" ve "store" komutlarıyla erişilmelidir.
- Bütün icra birimleri mikrokod kullanmadan donanımdan çalıştırılmalıdır.
- Basit, atomik komutlar (doğrudan donanım tarafından yürütülür), Küçük talimat seti, Örnekler: ARM (Gelişmiş RISC Makineleri)

x86 Hafıza Yönetimi

Protected mode (Korumalı Mod)

- Windows, Linux gibi işletim sistemleri için yerel moddur.
- Programlara segment adı verilen bellek alanları verilir.
- İşlemci programların atanmış bölümleri olan segmentleri dışında bellek erişimini engeller.

Real-address mode (Gerçek Adres Mod)

- MS-DOS işletim sistemi için yerel moddur
- Sistem belleğine ve donanım aygıtlarına doğrudan erişim: OS çökmesine neden olabilir

System management mode (Sistem Yönetim Modu)

- power management, system security, diagnostics

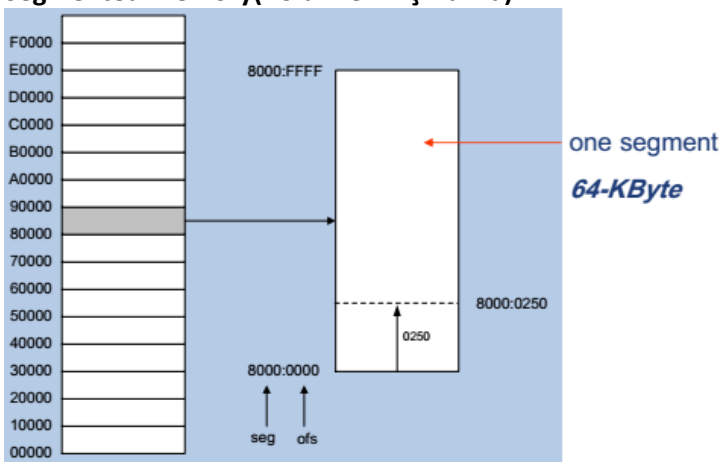
Virtual-8086 mode

Korunumlu özel durum (her program kendi 8086 bilgisayarına sahiptir), Kısacası, sanal 8086 modu, CPU'nun (korumalı modda) "Emulated" (gerçek mod) makine, ör. cmd komutu.

Adreslenebilir Hafıza

- 32 bit Protected mode (Korumalı Mod)'da maksimum 4GB bellek adreslenebilir.
- Real-adres ve Virtual-8086 modda 20 bit ile 1MB alan kullanılabilir.

Segmented Memory(Bölümlenmiş Hafıza)



Doğrusal Adresin Hesaplanması

08F1:0100 adresini doğrusal adrese çevirmeye çalışırsak!

```
Adjusted Segment value: 0 8 F 1 0
Add the offset:           0 1 0 0
Linear address:         0 9 0 1 0
```

Program Segment'leri:

Bir program üç bölümden oluşur code, data ve stack (kod, veri, yığıt). CS, DS, and SS register'ları segmentlerin taban adreslerini içerir. ES, FS ve GS, alternatif veri bölümlerine işaret edebilir, varsayılan veri bölümünü tamamlar.

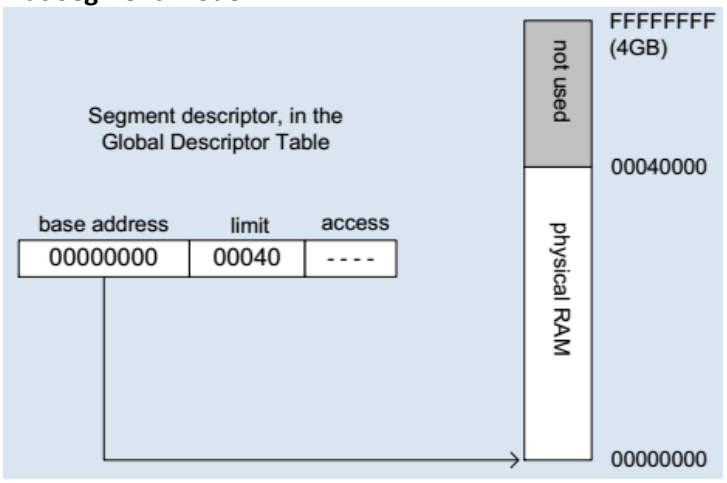
Korumalı Mod

4 GB adreslenebilir RAM (00000000h to FFFFFFFFh) vardır. Her programa korumalı bir hafıza alanı atanır. Multitasking için dizayn edilmiştir. Linux & MS-Windows tarafından desteklenir. Memory modelleri:

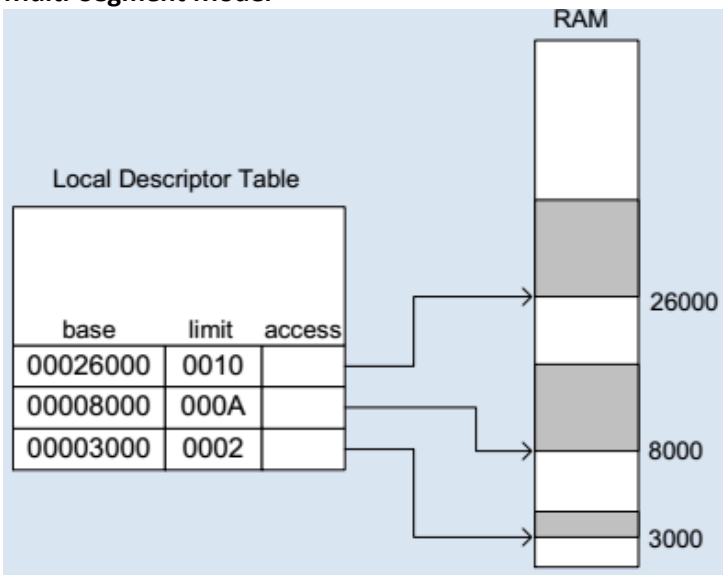
- Flat Segment Model
- Multi Segment Model
- Paging

MASM Programlar Microsoft flat memory mode kullanır.

Flat Segment Model

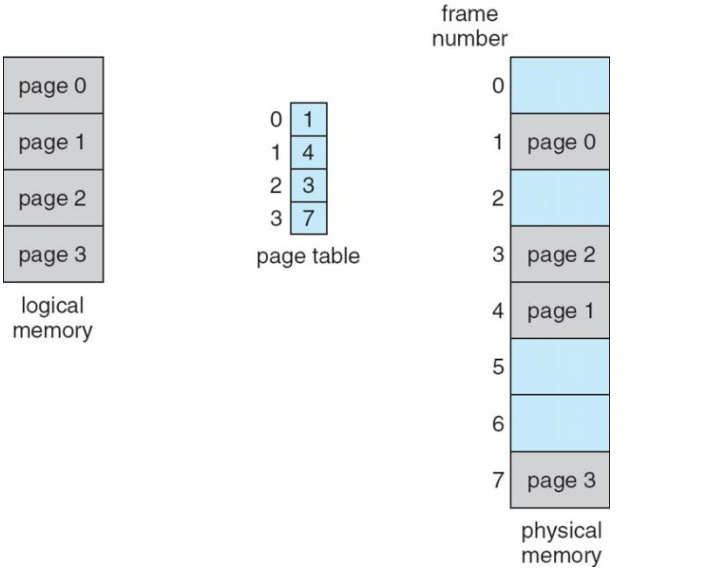


Multi-Segment Model



Paging (Sayfalama)

- Doğrudan CPU tarafından desteklenmektedir
- Her bir bölümü, sayfalar adı verilen 4096 bayt (4 Kb) bloklara böler
- Tüm programların toplamı fiziksel hafızadan daha büyük olabilir
- Çalışan programın bir kısmı bellekte, bir kısmı diskte
- Sanal bellek yöneticisi (VMM) - sayfaların yüklenmesini ve boşaltılmasını yöneten OS yardımcı programı
- Page fault - bir sayfanın diskten yüklenmesi gerektiğinde CPU tarafından verilir



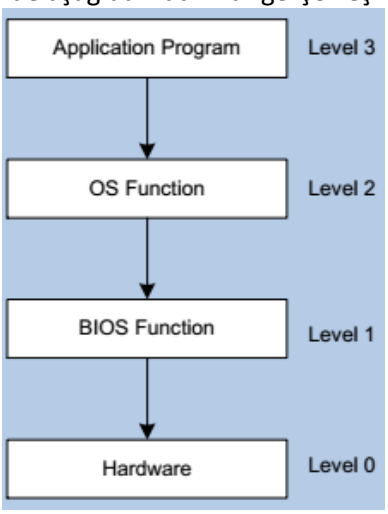
Levels of Input-Output (Giriş-Çıkış Aşamaları)

Seviye 3: Yüksek Seviye Diller (C++,Java vb.)

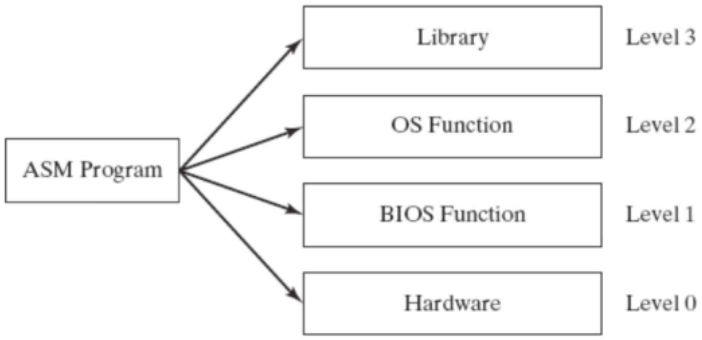
Seviye 2: İşletim Sistemi (API, Geniş yetenekler..)

Seviye 1: BIOS (doğrudan aygıt ile iletişim kuran sürücüler, işletim sistemi uygulamaların bu düzeyde çalışmasını engelleyebilir).

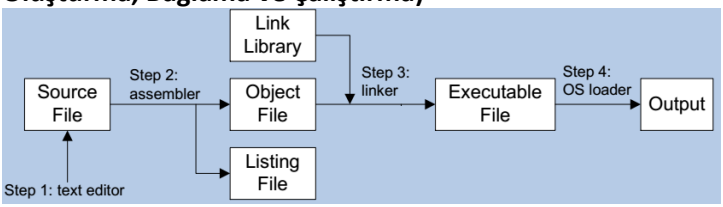
Bir Yksek Seviye Programlama Dili karakter dizini görüntülediğinde aşağıdaki adımlar gerçekleşir.



Assembly dili programları, bu seviyelerin her birinde giriş-çıkış yapabilir.



Assembling, Linking, and Running Programs (Programları Oluşturma, Bağlama ve Çalıştırma)



Şekilde kaynak kodun çalışabilir programa çevrilmesi aşaması gösterilmektedir. Eğer kaynak kodu değişirse 2. Adımdan 4. Adıma kadar işlemler tekrarlanmalıdır.

Listing File

Programınızın nasıl derlendiğini görmek için kullanılır. Kaynak kodu, adresler, makine kodu, seğmen isimleri, değişkenler ve sabitleri içerir. Yazdırmaya uygundur.

Assembly Dilinin Temel Elemanları

- Integer constants and expressions
- Character and string constants
- Reserved words and identifiers
- Directives and instructions
- Labels
- Mnemonics and Operands
- Comments

Integer constants
[{+|-} digits [radix] işaret opsiyoneldir.
h – hexadecimal En fazla kullanılan
d – decimal hex mantıklı gelmediğinde
b – binary bitsel anlaşılırlık için
r – encoded real Gerçek değer
Örneğin: 30d, 6Ah, 42, 1101b
Hexadecimal harf ile başlamaz!: 0A5h
Integer Expressions

Operator	Name	Precedence Level
( )	parentheses	1
+, -	unary plus, minus	2
*, /	multiply, divide	3
MOD	modulus	3
+, -	add, subtract	4

Characters and Strings
Karaker tek string çift tırnak içinde gösterilir.
'A', "x", ASCII karakter olarak 1 byte'tır.
"ABC", 'xyz' Her karakter 1 byte yer işgal eder.
İç içe işlemlere izin verilir: 'Say "Goodnight," Gracie'
Reserved Words and Identifiers
Ayrılmış kelimeler tanımlayıcı olarak kullanılamaz. (Talimat komutlar, direktifler, tip nitelikleri, operatörler, önceden tanımlanmış sembolleri)
Identifiers
1-247 arasındaki rakamları da içine alan karakterlerdir. Büyük küçük harf duyarlılığı yoktur. İlk karakter harf veya _, @, ?, $ olabilir. Sabitleri veya değişkenleri etiketlemek için kullanılır.
Directives (Talimatlar)
Assembler tarafından tanınan ve üzerinde işlem yapılan komutların nasıl kullanılacağını belirtir.

.data

.code

.stack 100h

Intel talimat setinin parçası değildir. Kod beyan etmek, veri alanlarını seçmek, bellek modelini seçmek, beyan etmek, prosedürler, değişkenler vb. için kullanılır. Büyük / küçük harfe duyarlı değildir (.data, .DATA ve .Data). Farklı assembly'ler farklı talimatlara sahiptirler. Assembler yönergelerinin önemli bir işlevi, program bölümlerini veya bölümleri tanımlamaktır.
Intel Komutları
Komutlar aşağıdakileri içerebilir:

- **Label** (opsiyonel), **Mnemonic** (gerekli), **Operand(s)** (komuta bağlı), **Comment** (opsiyonel) –';' ile başlar.

```
[label:] mnemonic [operands] [;comment]
```

```
loop1:    mov    eax,32    ;count of array elements
```

Labels
Kodun ve adresi belirtmek için kullanılır, tanımlayıcı kuralları takip eder, Değişken isimleri için kullanılır, benzersiz olmalıdır, atlama ve döngülerin etiketleri için de kullanılır.
Komut Formatları
stc ; set Carry flag
inc eax ; register
inc myByte ; memory
add ebx,ecx ; register, register
sub myByte,25 ; memory, constant
add eax,36 * 25 ; reg, const-expr

NOP Komutu
No operation(işlem yok) komutudur. Genellikle kodu eşit adres sınırlarına hizalamak için kullanılır. Hafızada 1 byte yer kaplar.

```
00000000 66 8B C3 mov ax,bx

00000003 90 nop ; align next instruction

00000004 8B D1 mov edx,ecx
```

x86 işlemcileri, doublword adreslerinden bile daha hızlı kod ve verileri yüklemek üzere tasarlanmıştır.
Sayısal Değerlerin Toplanması ve Çıkartılması

```
TITLE Program Template                (Template.asm)

.data

; (insert variables here)
.code
main PROC
    ; (insert executable instructions here)
    exit
main ENDP
    ; (insert additional procedures here)
END main
```

```
TITLE Add and Subtract                (AddSub.asm)

; This program adds and subtracts 32-bit integers.

INCLUDE Irvine32.inc

.code

main PROC
    mov eax,10000h          ; EAX = 10000h
    add eax,40000h          ; EAX = 50000h
    sub eax,20000h          ; EAX = 30000h
    call DumpRegs           ; display registers
    exit
main ENDP

END main
```

```
EAX=00030000  EBX=7FFDF000  ECX=00000101  EDX=FFFFFFFF
ESI=00000000  EDI=00000000  EBP=0012FFF0  ESP=0012FFC4
EIP=00401024  EFL=00000206  CF=0  SF=0  ZF=0  OF=0
```

```
TITLE Add and Subtract                (AddSub2.asm)

; This program adds and subtracts 32-bit integers.
; Without include
.386
.model flat, stdcall
.stack 4096
ExitProcess PROTO, dwExitCode:DWORD
DumpRegs PROTO

.code

main PROC
    mov eax,10000h              ; EAX = 10000h
    add eax,40000h              ; EAX = 50000h
    sub eax,20000h              ; EAX = 30000h
    call DumpRegs               ; display registers
    INVOKE ExitProcess, 0
main ENDP

END main
```

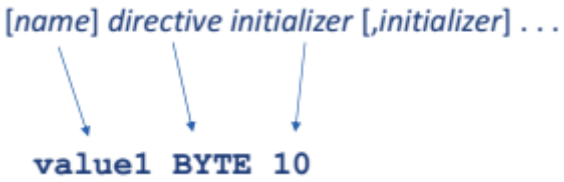
Temel Veri Tipleri

BYTE, SBYTE: 8-bit işaretsiz & işaretli sayılar
WORD, SWORD: 16-bit işaretsiz & işaretli sayılar
DWORD, SDWORD: 32-bit işaretsiz & işaretli sayılar
QWORD: 64-bit sayılar(işaretli/işaretsiz)
TBYTE: 80-bit (ten byte) sayılar
REAL4, REAL8: 4-byte short & 8-byte long reals
REAL10: 10-byte IEEE extended real

Doğru Veri Yönergeleri

Directive	Usage
DB	8-bit integer
DW	16-bit integer
DD	32-bit integer or real
DQ	64-bit integer or real
DT	80-bit (10 bytes) integer

Veri Tanımlama Bildirimi



BYTE ve SBYTE Tanımlama

```
value1 BYTE 'A'          ; character constant
value2 BYTE 0            ; smallest unsigned byte
value3 BYTE 255          ; largest unsigned byte
value4 SBYTE -128        ; smallest signed byte
value5 SBYTE +127        ; largest signed byte
value6 BYTE ?            ; uninitialized byte
```

value1 veri segment 0000 adresinde yer alır ve 1 byte depolama alanı kullanırsa value2 0001 adresinde yerleştirilir. MASM, negatif bir değere sahip BYTE'yi başlatmanıza izin verir (poor style). Bir SBYTE değişkeni bildirirseniz, Microsoft hata ayıklayıcı otomatik olarak değerini önde gelen bir işaretle ondalık olarak görüntüler

Byte Dizileri Tanımlama

```
list1 BYTE 10,20,30,40
list2 BYTE 10,20,30,40
      BYTE 50,60,70,80
      BYTE 81,82,83,84
list3 BYTE ?,32,41h,00100010b
list4 BYTE 0Ah,20h,'A',22h
```

Offset	Value
0000:	10
0001:	20
0002:	30
0003:	40

Bir dizi sadece ardışık bellek konumlarına yerleşir. Directive (BYTE) bir sonraki dizi öğesine ulaşmak için gereken ofseti belirtir. Uzunluk yok, sonlandırma bayrağı yok, özel özellikler yok.

String Tanımlama

```
str1    BYTE "Enter your name: ",0
str2    BYTE 'ERROR!',0Dh,0Ah,'Halting program',0Dh,0Ah,0
str3    BYTE 'A','E','I','O','U'
newLine BYTE 0Dh,0Ah,0
greet   BYTE "A string in"
        BYTE " two parts.",0
menu    BYTE "1. Create a new account",0dh,0ah,
             "2. Open an existing account",0dh,0ah,
             "3. Exit",0ah,0ah,
             "Choice> ",0
```

String bir karakter dizisi olarak uygulanır.
Kolaylık sağlamak için genellikle tırnak işaretleri içine alınır. Genellikle null ile sonlandırılmıştır. Karakterler Byte değere sahip olduğu için, 0Dh (CR) and 0Ah (LF) Hex karakterleri bitirme için daha kullanışlıdır.

DUP Operator

DUP veri için alan ayırmak adına kullanılır. **repetitions DUP(değişken)** şeklinde kullanılır. **Repetitions** 'lar sabit ifadeler olmalıdır.

```
var1 BYTE 20 DUP(0)        ; 20 bytes, all equal to zero
var2 BYTE 20 DUP(?)        ; 20 bytes, uninitialized
var3 BYTE 4 DUP("STACK")   ; 20 bytes: "STACKSTACKSTACKSTACK"
var4 BYTE 10,3 DUP(0),20   ; 5 bytes
```

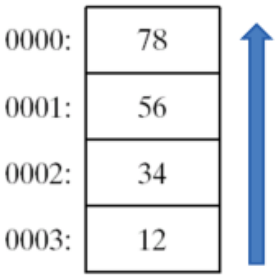
Diğer Tiplerin Tanımlanması

```
val1    WORD   65535             ; largest unsigned value
val2    SWORD  -32768            ; smallest signed value
word3   WORD   ?                 ; uninitialized, unsigned
word4   DWORD   "ABCD"           ; four characters
myList  WORD   1,2,3,4,5         ; array of words
array   WORD   5 DUP(?)          ; uninitialized array
val5    DWORD  0FFFF0000h        ; unsigned
val6    SDWORD -2147483648       ; signed
dwd7    SDWORD -2,-1,0,1,2       ; signed array
qwd8    QWORD  1234567812345678h
rVal1   REAL4  -2.1
rVal2   REAL8  3.2E-260
```

Little Endian Order (Küçük Sonda Sıralaması)

Bir bayttan daha büyük tüm veri türleri tek tek baytlarını ters sırayla depolar. En önemsiz bayt ilk (en düşük) bellek adresinde saklanır.

```
val1 DWORD 12345678h
```




```
TITLE Add and Subtract, Version 3            (AddSub3.asm)
; This program adds and subtracts 32-bit unsigned
; integers and stores the sum in a variable.
INCLUDE Irvine32.inc

.data

val1 DWORD 10000h
val2 DWORD 40000h
val3 DWORD 20000h
finalVal DWORD ?

.code

main PROC
    mov eax,val1              ; start with 10000h
    add eax,val2              ; add 40000h
    sub eax,val3              ; subtract 20000h
    mov finalVal,eax          ; store the result (30000h)
    call DumpRegs             ; display the registers
    exit
main ENDP
END main
```

Segment Kontrolü

.code

Bu alandan sonraki komutlar code segment'ine gider

.data

Bu alandan sonraki komutlar data segment'ine gider

.data?

- Başlatılmamış veri bölümü
- Çalışma zamanında ayrılır Veriyi depolamak için kullanılır
- Depolanmış .exe dosyasında boş alan gerekmiyor (depolanacak değer olmadığından)

Declaring Uninitialized Data-Başlatılmamış Veri Tanımlama

.data? şeklinde tanımlanır ve derlenmiş bir programın boyutunu azaltır

```
.data?

array1 DWORD 5000 DUP (?)
```

Programın çalıştırılması için yüklenene kadar array1'e yer ayrılmaz (.exe 20KB daha küçük)

```
.data

array2 DWORD 5000 DUP (?)
```

array2, boş olsa da var olmayan değerlerini depolamak için .exe dosyasına 20 KB kaydedilir.

Symbolic Constants(Sembolik Sabitler)

Sayısal sembolik sabit: İfade 32 bitlik tamsayıdır. Yeniden tanımlanabilir fakat bu iyi bir program olmaz. Çalışma zamanına etkisi yoktur.

```
COUNT = 500

...

mov ax,COUNT
```

EQU Directive: Bir sembol, bir tamsayı veya metin ifadesi olarak tanımlanır. Yeniden tanımlanamazlar. = komutu sadece sayısal ifadeler için kullanılır.

```
PI EQU <3.1416>
pressKey EQU <"Press any key to continue...",0>
.data
prompt BYTE pressKey
```

TEXTEQU Directive: Metinsel sembolü tamsayı veya metin ifadesi olarak tanımlamak için kullanılır. **text macro** olarak adlandırılır. Yeniden tanımlanabilir. **%** sayılsa ifadeyi text e çevirir.

```
;macros
msg TEXTEQU <"Do you wish to continue (Y/N)?">
rowSize = 5
count TEXTEQU %(rowSize * 2)      ; eval & store as text
setupAL TEXTEQU <mov al,count>    ; macro for a mov instr
.data
prompt1 BYTE msg
.code
setupAL                           ; creates "mov al,10"
```

Array Size (Dizi Boyutu)

```
list BYTE 10,20,30,40
listSize = ($ - list)
```

Eğer eleman boyutu byte den büyükse bölünür. (örneğin, WORD için 2, DWORD için 4, QWORD için 8'e bölünür)

```
list DWORD 1,2,3,4
listSize = ($ - list) / 4
```

Operand Types(işlem Tipleri)

Immediate: Sayısal Sabit Değer (8, 16, veya 32 bit)

Değeri komutlar ile kodlanır.

Registers: kaydedicilerin isimleri.

Memory: Hafızadaki bir konumu gösterir.

Bellek adresi komutlar içinde kodlanır veya bir kayıt bir bellek konumuna ait adresi tutar.

```
.data
var1 BYTE 10h
;Suppose var1 were located at offset 10400h
mov AL,var1  → A0 00010400
```

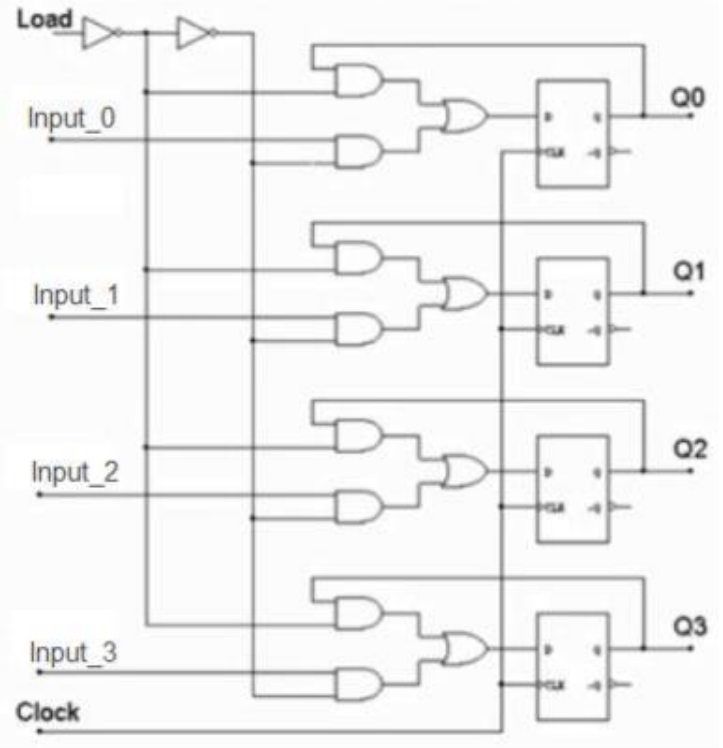
Operand	Description
reg8	8-bit general-purpose register: AH, AL, BH, BL, CH, CL, DH, DL
reg16	16-bit general-purpose register: AX, BX, CX, DX, SI, DI, SP, BP
reg32	32-bit general-purpose register: EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP
reg	Any general-purpose register
sreg	16-bit segment register: CS, DS, SS, ES, FS, GS
imm	8-, 16-, or 32-bit immediate value
imm8	8-bit immediate byte value
imm16	16-bit immediate word value
imm32	32-bit immediate doubleword value
reg/mem8	8-bit operand, which can be an 8-bit general register or memory byte
reg/mem16	16-bit operand, which can be a 16-bit general register or memory word
reg/mem32	32-bit operand, which can be a 32-bit general register or memory doubleword
mem	An 8-, 16-, or 32-bit memory operand

MOV Komutu: kaynaktan hedefe veri taşır. İki işlenen de aynı boyutta olmalıdır. Birden fazla bellek işlenenine izin vermez. CS, EIP ve IP hedef olamazlar.

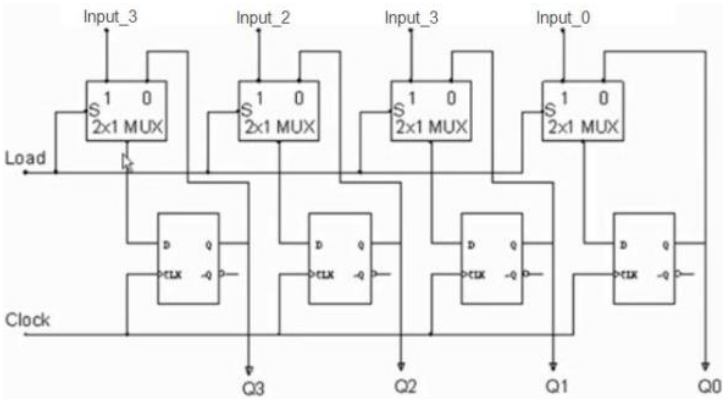
```
MOV destination, source

.code
mov ax,var1
mov var2,ax
```

LOAD = 0, keep the previous state.

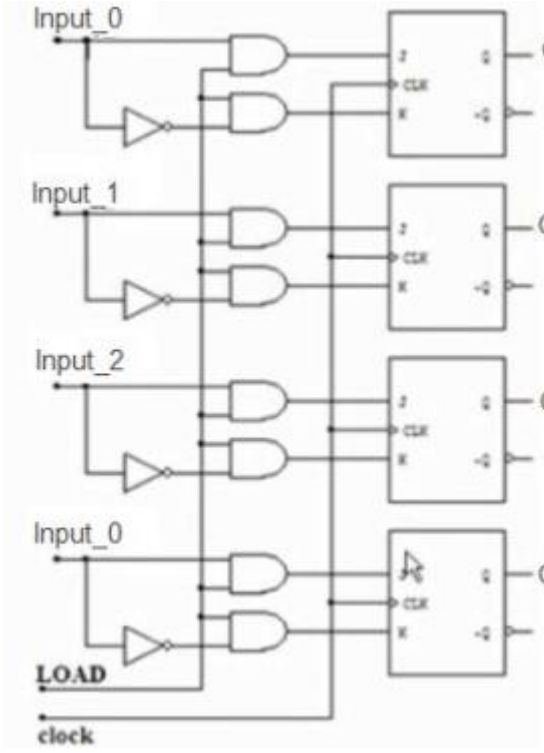
LOAD = 1, Load the inputs to the register



Mux ile Yükleme



J-K Flip Flop ile Yükleme



Direct Memory Operands (Doğrudan Hafıza İşlemleri)

```
.data
var1 BYTE 10h
.code
mov al,var1              ; AL = 10h
mov al,[var1]            ; AL = 10h
```

alternate format – Use consistently if you chose to use it

Use it only when an arithmetic expression is involved

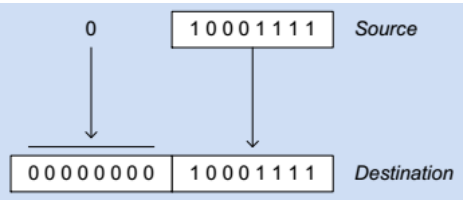
mov al, [var1 +5]

Mov Hataları

```
.data
    bVal   BYTE   100
    bVal2  BYTE   ?
    wVal   WORD   2
    dVal   DWORD  5

.code
    mov al,wVal              ; byte <- word
    mov ax,bVal              ; word <- byte
    mov eax,bVal             ; dword <- byte
    mov ds,45                ;immediate value not permitted
    mov eip,dVal             ;invalid destination (eip)
    mov 25,bVal              ;invalid destination (25)
    mov bVal2,bVal           ;move in mem not permitted
```

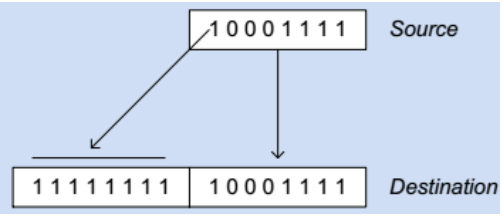
Zero Extension (Sıfır Uzatma): Hedef register olmalı!

```
mov    bl,10001111b
movzx ax,bl                  ; zero-extension
```



Sign Extension (işaret Uzatma): Hedef register olmalı!

```
mov    bl,10001111b
movsx ax,bl                  ; sign extension
```



XCHG Komutu (swap)

XCHG iki işleneni(operand) yer değiştirir. En azından bir tanesi register olmalıdır. Sabit değerleri kabul etmez.

```
.data

    var1 WORD 1000h
    var2 WORD 2000h

.code
    xchg ax,bx           ; exchange 16-bit regs
    xchg ah,al           ; exchange 8-bit regs
    xchg var1,bx         ; exchange mem, reg
    xchg eax,ebx         ; exchange 32-bit regs

    xchg var1,var2       ; error: two memory operands
```

LAHF ve SAHF Komutları

LAHF: AH'ye durum bayrakları yükler. İşaret, Sıfır ve Taşıma bayrakları da dahil olmak üzere EFLAGS registerinin düşük baytını kopyalar.

```
.data
    saveflags BYTE ?
.code
    lahf                 ; load flags into AH
    mov saveflags,ah     ; save them into a variable
```

SAHF: AH'yi durum bayraklarına depolar. AH'yi EFLAGS kaydının düşük baytına kopyalar.

```
.code
    mov ah, saveflags    ; load save flags into AH
    sahf                 ; copy into flags register
```

Intel x86-16 bit Bayraklar

Bit #	Abbreviation	Description	Category
		FLAGS	
0	CF	Carry flag	Status
1	1	Reserved	
2	PF	Parity flag	Status
3	0	Reserved	
4	AF	Adjust flag	Status
5	0	Reserved	
6	ZF	Zero flag	Status
7	SF	Sign flag	Status
8	TF	Trap flag (single step)	Control
9	IF	Interrupt enable flag	Control
10	DF	Direction flag	Control
11	OF	Overflow flag	Status
12-13	IOPL	I/O privilege level (286+ only), always 1 on 8086 and 186	System
14	NT	Nested task flag (286+ only), always 1 on 8086 and 186	System
15	0	Reserved, always 1 on 8086 and 186, always 0 on later models	

Bit #	Abbreviation	Description	Category
		EFLAGS	
16	RF	Resume flag (386+ only)	System
17	VM	Virtual 8086 mode flag (386+ only)	System
18	AC	Alignment check (486SX+ only)	System
19	VIF	Virtual interrupt flag (Pentium+)	System
20	VIP	Virtual interrupt pending (Pentium+)	System
21	ID	Able to use CPUID instruction (Pentium+)	System
22	0	Reserved	
23	0	Reserved	
24	0	Reserved	
25	0	Reserved	
26	0	Reserved	
27	0	Reserved	
28	0	Reserved	
29	0	Reserved	
30	0	Reserved	
31	0	Reserved	
		RFLAGS	
32-63	0	Reserved	

Direct-Offset Operands: Etkili bir adres (EA) üretmek için bir veri etiketine sabit ofset eklenir.

```
.data

   arrayB BYTE 10h,20h,30h,40h

.code
   mov al,arrayB+1              ; AL = 20h
   mov al,[arrayB+1]            ; alternative notation
```

1. arrayB etiketli tarafından belirtilen adres alın
2. Adrese 1 ekleyin (ikinci dizi öğesini almak için)
3. Değer elde etmek için adres ayırma adresi (20h)

ÖRNEK

```
.data

   arrayW  WORD 1000h,2000h,3000h
   arrayD  DWORD 1,2,3,4

.code
   mov ax,[arrayW+2]            ; AX = 2000h
   mov ax,[arrayW+4]            ; AX = 3000h
   mov ax, [arrayW+6]           ; EAX = 00000002h
   mov eax,[arrayD+4]
                                ; AX = 3000h
   mov ax,[arrayD-2]            ; Possible Seg Fault!
   mov eax,[arrayD+20]
```

arrayW word olduğu için 2byte

10 00, 20 00, 30 00

dword 4 byte olduğu için

00 00 00 01, 00 00 00 02, 00 00 00 03, 00 00 00 04

0	00	**6**	01	**12**	00	**18**	04
1	10	**7**	00	**13**	00	**19**	00
2	00	**8**	00	**14**	03	**20**	00
3	20	**9**	00	**15**	00	**21**	00
4	00	**10**	02	**16**	00		
5	30	**11**	00	**17**	00		

ÖRNEK

```
.data

  myBytes     BYTE    80h,66h,0A5h

.code
          movzx ax,myBytes
          mov   bl,[myBytes+1]
          add   ax,bx
          mov   bl,[myBytes+2]
          add   ax,bx           ; AX = sum
```

INC ve DEC Komutları: işlenene bir ekleyip bir çıkartmak. İşlenen register veya hafıza olabilir.
INC hedef, **Logic**: hedef=hedef + 1
DEC hedef, **Logic**: hedef=hedef - 1

```
.data

   myWord  WORD  1000h
   myDword DWORD 10000000h

.code
   inc myWord                   ; 1001h
   dec myWord                   ; 1000h
   inc myDword                  ; 10000001h

   mov ax,00FFh
   inc ax                       ; AX = 0100h
   mov ax,00FFh
   inc al                       ; AX = 0000h
```

ADD ve SUB Komutları

ADD hedef, kaynak Logic: hedef=hedef+kaynak
SUB hedef, kaynak Logic: hedef=hedef-kaynak
Mov komutunu işletme kuralları geçerlidir.

```
.data
   var1    DWORD 10000h
   var2    DWORD 20000h

.code
   mov eax,var1                 ; 00010000h
   add eax,var2                 ; 00030000h
   add ax,0FFFFh                ; 0003FFFFh
   add eax,1                    ; 00040000h
   sub ax,1                     ; 0004FFFFh
```

NEG (negate) Komutu

Bir hafızada veya registerdeki değerin tersini alır (2'ye tümleyen).

```
.data
   valB    BYTE -1
   valW    WORD +32767

.code
   mov al,valB                  ; AL = -1
   neg al                       ; AL = +1
   neg valW                     ; valW = -32767
```

Aritmetik Gerçekleştirme

Rval = -Xval + (Yval - Zval)

```
.data
   Rval DWORD ?
   Xval DWORD 26
   Yval DWORD 30
   Zval DWORD 40

.code
   ;first term :-Xval
   mov eax,Xval
   neg eax                      ; EAX = -26
   ;second term :Yval - Zval
   mov ebx,Yval
   sub ebx,Zval                 ; EBX = -10
   ;add the terms and store the result
   add eax,ebx
   mov Rval,eax                 ; -36
```

Flags Affected by Arithmetic(Aritmetikten Etkilenen Bayraklar)

ALU, aritmetik işlemlerin sonucunu yansıtan bir dizi durum bayrağı içerir. Bu değer hedef işlenenine bağlı olarak belirlenir.

Temel Bayraklar:

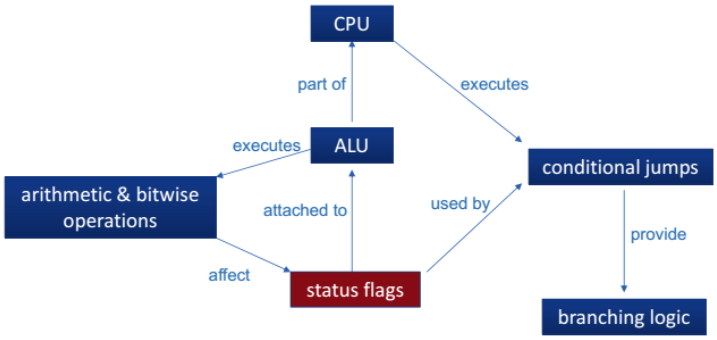
Zero flag – hedef değeri 0 olduğunda 1 olur

Sign flag – hedef değeri negatif olduğunda 1 olur

Carry flag – İşaretsiz değer aralık dışında olduğunda 1 olur

Overflow flag – İşaretli değer aralık dışında olduğunda 1 olur

MOV KOMUTU ASLA BAYRAKLARI ETKİLEMEZ



Zero Flag ZF:

```
mov cx,1
sub cx,1              ; CX = 0, ZF = 1
mov ax,0FFFFh
add ax,1              ; AX = 0, ZF = 1
add ax,1              ; AX = 1, ZF = 0
```

Sign Flag (SF): hedefin en yüksek bitinin bir kopyasıdır

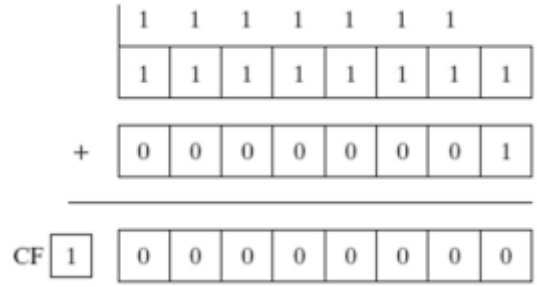
```
mov cx,0
sub cx,1            ; CX = -1, SF = 1
add cx,2            ; CX =  1, SF = 0
mov al,0
sub al,1          ; AL = 11111111b, SF = 1
add al,2          ; AL = 00000001b, SF = 0
```

Sayıların İşaretleri:

Tüm CPU komutları tam olarak işaretli ve işaretsiz tamsayılarda aynı çalışır. İşlemci işarteli veya işaretsiz sayıyı ayırt edemez. Her aşamada doğru veri türünü kullanmakla programcı sorumludur.

Carry Flag (CF): Elde bayrağı boyut hatasıyla ilgilidir (işaretsiz aritmetik).

```
mov al,0FFh
add al,1          ; CF = 1, AL = 00

mov ax,0FFh
add ax,1          ; CF = 0, AX = 0100h
```



Al 8 bit olduğu için elde oluşur, ax 16 bit olduğu için elde oluşmaz.

```
mov ax,00FFh

add ax,1          ; AX=0100h, SF=0 ZF=0 CF=0
sub ax,1          ; AX=00FFh, SF=0 ZF=0 CF=0
add al,1          ; AL=00h,   SF=0 ZF=1 CF=1

mov bh,6Ch
add bh,95h        ; BH=01h,   SF=0 ZF=0 CF=1

mov al,2
sub al,3          ; AL=FFh,   SF=1 ZF=0 CF=1
```

Overflow Flag (OF): Taşma bayrağı işaret hatası (işaretli aritmetik) ile ilgilidir.

```
mov al,+127
add al,1          ; OF = 1,   AL = -128

mov al,7Fh        ; OF = 1,   AL = 80h
add al,1
```

İki örneğin sonucu da aynıdır çünkü -128=80h'dır. Genellikle hexedcimal sayıların işlenmesi daha kolaydır.

```
What will be the values of the Overflow flag?
   mov al,80h             ; (-128) + (-110)
   add al,92h             ; OF = 1, al= 26

   mov al,-2
   add al,+127            ; OF = 0
```

İpucu!

İki pozitif işlenen toplandı ve bunların toplamı negatif çıktı, İki negatif işlenen toplandı ve bunların toplamı pozitif çıktı. Sonuç işareti, işlenenlerin işaretinin tam karşısındadır. İki toplananın işaretleri farklı olduğunda taşma meydana gelmez.

Bayraklar İçin Özel Durumlar

INC ve DEC talimatları Elde bayrağını (CF) **etkilemez**.

NEG komutu, sıfırdan farklı bir işlenene uygulanırsa, Elde (CF)bayrağı **daima 1** olur.

CPU, hangi bayrakların alakalı olursa olsun, bir dizi aritmetik işlemin ardından Boolean kuralları kümesini kullanarak tüm durum bayraklarını ayarlar.

Siz (programcı) gerçekleştirilen işlem türüne ilişkin bilginize dayanarak hangi bayrakları yorumlayacağınıza ve yok sayacağınıza karar vermelisiniz.

Data-Related Operators and Directives (Veri ile İlgili Operatörler ve Komutlar)

Align Komutu: ALIGN komutu bir değişkeni bir bayt, word, double Word veya paragraf sınırında hizalamaktadır.

```
.data
bVal    BYTE ?      ; 00404000
ALIGN 2
wVal    WORD ?      ; 00404002
bVal2   BYTE ?      ; 00404004
ALIGN 4
dVal    DWORD ?     ; 00404008
dVal2   DWORD ?     ; 0040400C
```

PTR Operator

Varsayılan bir etiket türünü geçersiz kılar (değişken), Bir değişkenin bir bölümüne erişme esnekliği sağlar, Ön ekli bir boyut belirteci gerektirir.

```
.data
   myDouble DWORD 12345678h
.code

   mov ax,myDouble                  ;error! word<-dword

   mov ax,WORD PTR myDouble         ;loads 5678h

   mov WORD PTR myDouble,4321h      ;saves 4321h
```

Little Endian Order (again) (Küçük Sonda Sıralaması)

byte	offset
78	0000
56	0001
34	0002
12	0003

LEO Intel'in tam sayıları bellekte saklama biçimini belirtir. Çok baytlı tamsayılar ters sırayla saklanır, en az anlamlı bayt en düşük adresde saklanır. Örneğin DWORD 12345678h değişkeni hafızada yandaki şekilde saklanır. Tamsayılar bellekten kayıtlara yüklendiğinde, baytlar otomatik olarak doğru konumlarına geri döndürülür.

PTR Operatör Örneği

```
.data
   myDouble DWORD 12345678h
```

doubleword	word	byte	offset	
12345678	5678	78	0000	myDouble
		56	0001	myDouble + 1
	1234	34	0002	myDouble + 2
		12	0003	myDouble + 3

```
   mov al,BYTE PTR  myDouble              ; AL = 78h

   mov al,BYTE PTR [myDouble+1]           ; AL = 56h

   mov al,BYTE PTR [myDouble+2]           ; AL = 34h

   mov ax,WORD PTR  myDouble              ; AX = 5678h

   mov ax,WORD PTR [myDouble+2]           ; AX = 1234h
```

Joining Words (Kelimeleri Birleştirme)

PTR aynı zamanda daha küçük bir veri türünün elemanlarını birleştirmek ve bunları daha büyük bir işlenene taşımak için kullanılabilir. CPU otomatik olarak baytları tersine çevirir.

```
.data
   myBytes BYTE 12h,34h,56h,78h

.code

   mov ax,WORD PTR [myBytes]          ; AX = 3412h

   mov ax,WORD PTR [myBytes+2]        ; AX = 7856h

   mov eax,DWORD PTR myBytes          ; EAX = 78563412h
```

ÖRNEK

```
.data
   varB BYTE 65h,31h,02h,05h
   varW WORD 6543h,1202h
   varD DWORD 12345678h

.code

     mov ax,WORD PTR [varB+2]         ; ax=0502h

     mov bl,BYTE PTR varD             ; bl=78h

     mov bl,BYTE PTR [varW+2]         ; bl=02h

     mov ax,WORD PTR [varD+2]         ; ax=1234h

     mov eax,DWORD PTR varW           ; eax=12026543h
```

TYPE Operator (Tip Operatörü)

TYPE, bir veri bildiriminin tek bir öğesinin bayt cinsinden boyutunu döndürür.

```
.data
    var1  BYTE ?
    var2  WORD ?
    var3  DWORD ?
    var4  QWORD ?

.code

    mov eax,TYPE var1       ; 1

    mov eax,TYPE var2       ; 2

    mov eax,TYPE var3       ; 4

    mov eax,TYPE var4       ; 8
```

LENGTHOF Operator

LENGTH OF operatörü, tek bir veri bildiriminde eleman sayısını sayar.

```
.data                                   LENGTHOF
   byte1    BYTE 10,20,30               ; 3

   array1   WORD 30 DUP(?),0,0          ; 32

   array2   WORD 5 DUP(3 DUP(?))        ; 15

   array3   DWORD 1,2,3,4               ; 4

   digitStr BYTE "12345678",0           ; 9

.code
    mov ecx,LENGTHOF array1             ; 32
```

SIZEOF Operator

SIZEOF operatörü tip ve lenght nin çarpılması işlemine eşdeğerdir.

SIZEOF = LENGTHOF * TYPE

```
.data                                   SIZEOF

byte1    BYTE 10,20,30                  ; 3

array1   WORD 30 DUP(?),0,0             ; 64

array2   WORD 5 DUP(3 DUP(?))           ; 30

array3   DWORD 1,2,3,4                  ; 16

digitStr BYTE "12345678",0              ; 9

.code
         mov ecx,SIZEOF array1          ; 64
```

Spanning Multiple Lines (Çoklu Satırları Ölçme)

Eğer her satır (son hariç) virgülle bitiyorsa, bir veri bildirimi birden fazla satırı kapsar. LENGTHOF ve SIZEOF operatörleri tanımlamaya ait tüm satırları içerir.

```
.data
array WORD  10,20,
            30,40,
            50,60

.code
            mov eax,LENGTHOF array    ; eax=6

            mov ebx,SIZEOF array      ; eax=12
```

Contrast: Anonymous Data(Zıttı Anonim Veri)

Aşağıdaki örnekte, dizi yalnızca 2 kelimeyle birlikte ilk WORD bildirimini tanımlar; buna rağmen 6 kelimeye erişmek için ad kullanılabilir.

```
.data
array  WORD 10,20     ; array ends here
       WORD 30,40     ; anonymous data, array+4
       WORD 50,60     ; array+8

.code
       mov eax,LENGTHOF array     ; 2

       mov ebx,SIZEOF array       ; 4
```

LABEL Directive:

Varolan bir saklama konumuna alternatif bir etiket adı ve türü atar. LABEL kendine ait depolama alanlarını tahsis etmez. PTR operatörüne gerek duyulmaz.

```
.data
       dwList   LABEL DWORD
       wordList LABEL WORD
       byteList  BYTE 00h,10h,00h,20h
.code
       mov eax,dwList         ; 20001000h
       mov cx,wordList        ; 1000h
       mov dl,intList         ; 00h
```

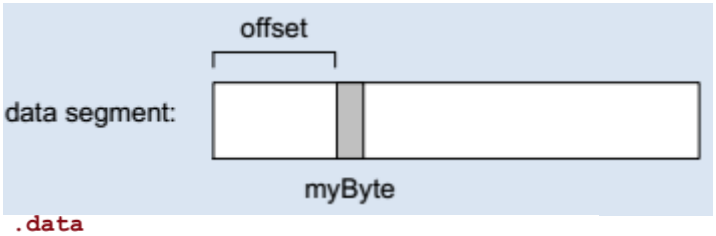
dwList, wordList, intList are the same offset (address)

OFFSET Operator:

OFFSET, bir etiketin bayt cinsinden alanını çevreleyen bölümün başına döndürür.

Protected mode: **32 bits**

Real mode: **16 bits**




```
.data
   bVal   BYTE ?
   wVal   WORD ?
   dVal   DWORD ?
   dVal2 DWORD ?

.code
   mov esi,OFFSET bVal   ; ESI = 00404000

   mov esi,OFFSET wVal   ; ESI = 00404001

   mov esi,OFFSET dVal   ; ESI = 00404003

   mov esi,OFFSET dVal2 ; ESI = 00404007
```

Indirect Operands (Register as a pointer): Dolaylı işlenene genelde string veya dizinin değişkenin adresini tutar. [Ve] düğmesini kullanarak referans alınabilir (sadece bir işaretçi gibi). Referans adresine erişmek için OFFSET değerini kullanır.

```
.data
   val1 BYTE 10h,20h,30h
.code
   mov esi,OFFSET val1    ; esi stores address of val1
   mov al,[esi]           ; dereference ESI (AL = 10h)

   inc esi
   mov al,[esi]           ; AL = 20h

   inc esi
   mov al,[esi]           ; AL = 30h
```

NOTE: We tend to use esi and edi to store addresses

```
// C++ version:
char array[1000];
char * p = array;
```

```
; Assembly language:
.data
  array BYTE 1000 DUP(?)
.code
  mov esi,OFFSET array
```

PTR Kullanımı

Bir bellek işlenmesinin boyut özniteliğini açıklığa kavuşturmak için PTR'yi kullanabiliriz. Bir adrese sahip olduğumuzda (ofset), bu uzaklıktaki değerlerin boyutunu bilmiyoruz ve bunları açıkça belirtmeliyiz.

```
.data
   myCount WORD 0

.code
   mov esi,OFFSET myCount

   inc [esi]           ; error: operand must have size
   inc WORD PTR [esi]  ; ok

   add [esi],20        ; error:..
   add ax, [esi]       ; ax or al specifies the size
   add WORD PTR [esi],20 ; ok
```

DİZİ TOPLAMA ÖRNEĞİ

```
.data
   arrayW WORD 1000h,2000h,3000h

.code
       mov esi,OFFSET arrayW
       mov ax,[esi]
       add esi,2
       ;or    add esi,TYPE arrayW     ; good clarity
       add ax,[esi]
       add esi,2
       add ax,[esi]             ; AX = sum of the array
```

DEĞİŞKENİ İŞARETÇİ OLARAK KULLANMA

Uzaklıkların boyutu: DWORD

DWORD boyutunda bir değişken, bir ofset tutabilir

Örneğin başka bir değişkenin ofsetini içeren bir işaretçi değişkeni bildirebilirsiniz

```
.data
   arrayW WORD  1000h,2000h,3000h
   ptrW   DWORD arrayW        ; ptrW = offset of arrayW
   ; Alternative - same as above
   ;ptrW  DWORD OFFSET arrayW

.code
       mov esi,ptrW

       mov ax,[esi]        ; AX = 1000h
```

Indexed Operands:

İndekslenmiş bir işlenen, etkili bir adres oluşturmak için bir adres ve bir kayıt ekler. İki farklı Gösterimi vardır.

[label + reg]

label[reg]

```
.data
   arrayW WORD 1000h,2000h,3000h
.code
   mov esi,0
   mov ax,[arrayW + esi]      ; AX = 1000h
   mov ax,arrayW[esi]         ; alternate format
   add esi,TYPE arrayW
   add ax,[arrayW + esi]
```

Index Scaling: dizi offset'ine bir çarpma işlemi yapılarak erişilebilir.

```
.data
   arrayB  BYTE  0,1,2,3,4,5
   arrayW  WORD  0,1,2,3,4,5
   arrayD  DWORD 0,1,2,3,4,5

.code
    mov esi,4
    mov al,arrayB[esi*TYPE arrayB]          ; 04
    mov bx,arrayW[esi*TYPE arrayW]          ; 0004
    mov edx,arrayD[esi*TYPE arrayD]         ; 00000004
```

JMP Komutu

Example

```
top:
  .
  .
  jmp top
```

Kontrol akışının çoğunun temelini oluşturur. HLL derleyicileri döngüler, if durumları, switches vb. durumları bir çeşit jump a çevirir. JMP genellikle aynı prosedür içindeki bir etikete koşulsuz bir atlama yapar.
Syntax: JMP hedef
Logic: EIP <- hedef

LOOP Komutu: LOOP komutu, ECX'i kullanarak sayılmış bir döngü oluşturur.

```
    mov  ax,0
    mov  ecx,5
L1:
    add  ax,cx
    loop L1
```

This loop calculates the sum: 5 + 4 + 3 +2 + 1

```
    mov ax,6
    mov ecx,4     ;Loop 4 times
L1:
    inc ax        ;Each iteration ax++ (7,8,9,10)
    loop L1
    mov ecx,0     ;ecx starts at 0! (an error)
X2:
    inc ax        ;ax++ until ecx holds 0
    loop X2       ;ecx- (-1,-2,-3, . . .)

;ax = 4294967296 when you exit the loop
```

Nested Loops (İç içe döngüler): Bir döngü içinde bir döngü kodlamanız gerekiyorsa, dış döngü sayacının ECX değerini kaydetmeniz gerekir. Bu örnekte dış döngü 100 kez, iç döngü ise 20 kez yürütülür.

```
.data
   count DWORD ?
.code
    mov ecx,100         ; set outer loop count
L1:
      mov count,ecx     ; save outer loop count
      mov ecx,20        ; set inner loop count
L2:
      loop L2           ; repeat the inner loop
      mov ecx,count     ; restore outer loop count
      loop L1           ; repeat the outer loop
```

Dizinin Toplanması

```
.data
    intarray WORD 100h,200h,300h,400h
.code
    mov edi,OFFSET intarray       ; address of intarray
    mov ecx,LENGTHOF intarray     ; loop counter
    mov ax,0                      ; zero the accumulator
L1:

    add ax,[edi]                   ; add an integer
    add edi,TYPE intarray          ; point to next integer

    loop L1                        ; repeat until ECX = 0
```

String Kopyalama

```
.data
    source  BYTE  "This is the source string",0
    target  BYTE  SIZEOF source DUP(0)

.code
    mov  esi,0                ; index register
    mov  ecx,SIZEOF source    ; loop counter
L1:
    mov  al,source[esi]       ; get char from source
    mov  target[esi],al       ; store it in the target
    inc  esi                  ; move to next character

    loop L1                   ; repeat for entire string
```