

William Stallings  
Computer Organization  
and Architecture  
10<sup>th</sup> Edition

Orijinal slaytların  
çevirisidir.

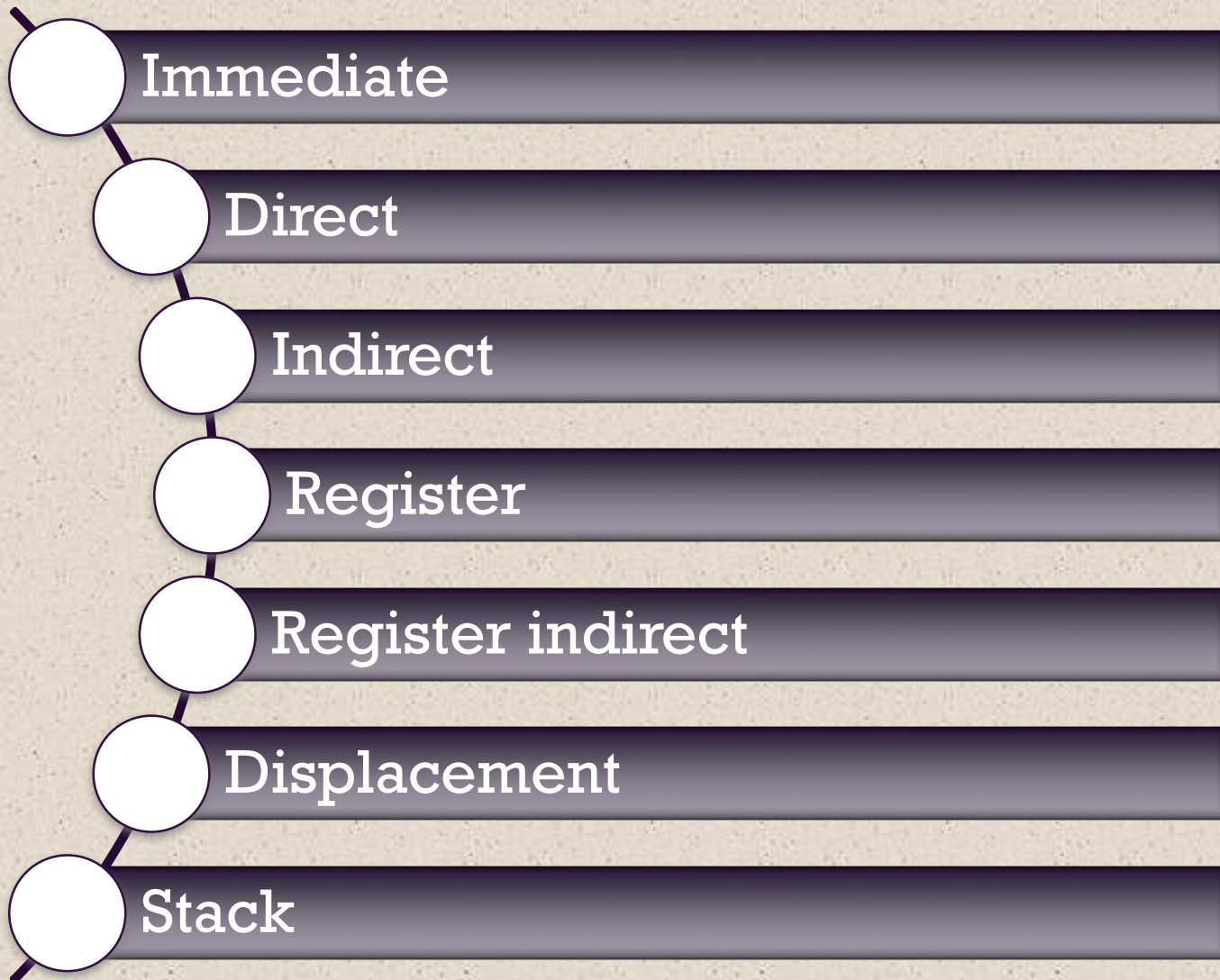
Bu bölüm, komutların operandlarının ve işlemlerinin nasıl belirleneceği sorusuyla ilgilenir. İki sorun ortaya çıkıyor.

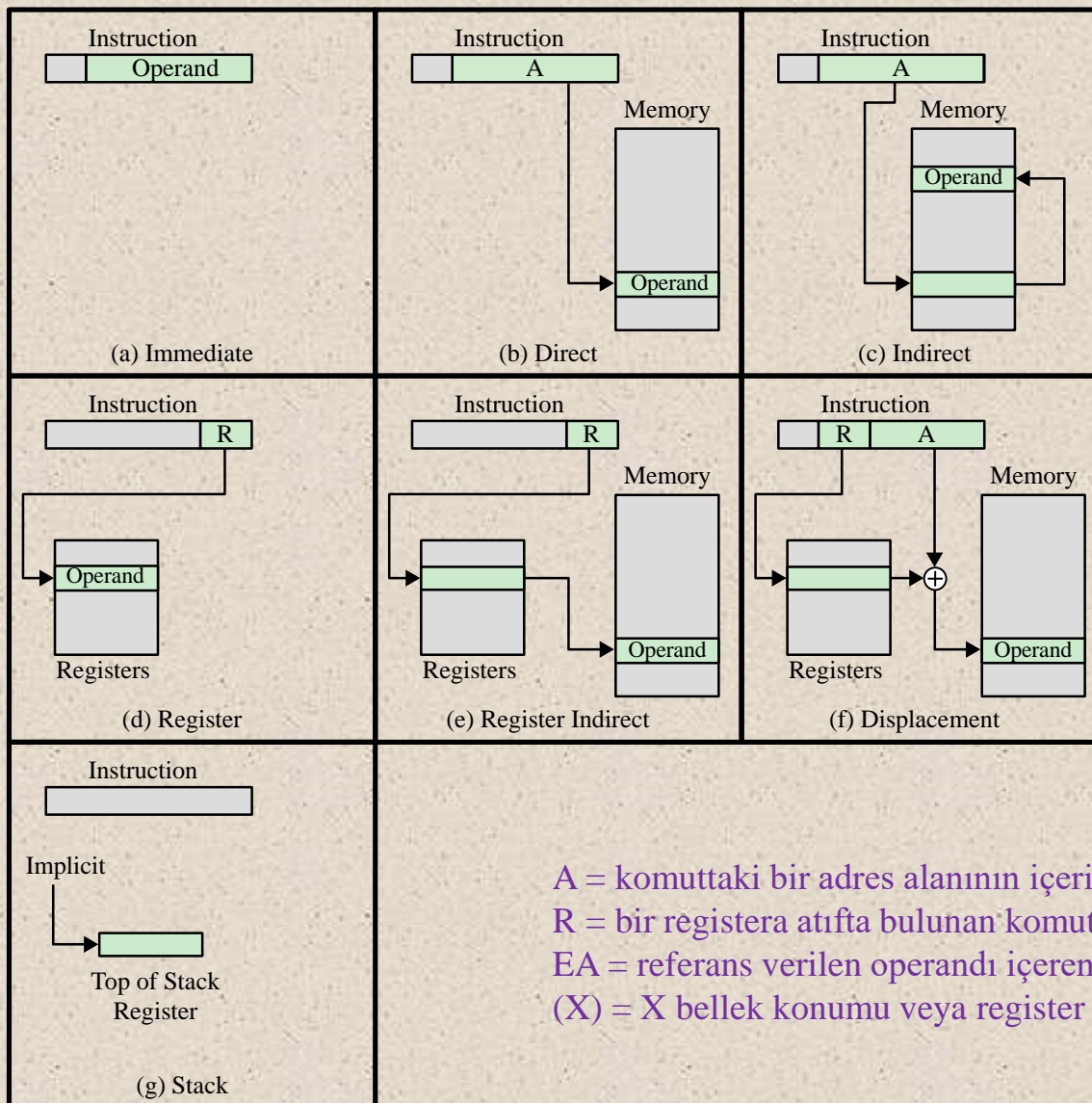
Birincisi, bir operandın adresi nasıl belirtilir ve ikincisi, bir komutun bitleri bu komutun operand adreslerini ve işleyişini tanımlamak için nasıl düzenlenir?

## + Chapter 13

### Instruction Sets: Addressing Modes and Formats


# Addressing Modes





A = komuttaki bir adres alanının içeriği  
R = bir registera atıfta bulunan komuttaki bir adres alanının içeriği  
EA = referans verilen operandı içeren konumun gerçek (efektif) adresi  
(X) = X bellek konumu veya register X'in içeriği

**Figure 13.1 Addressing Modes**



# Table 13.1

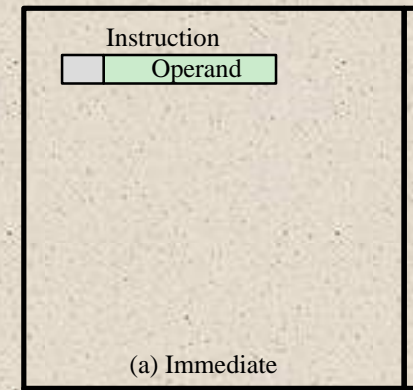
## Basic Addressing Modes

Mode	Algorithm	Principal Advantage	Principal Disadvantage
Immediate	Operand = A	No memory reference	Limited operand magnitude
Direct	EA = A	Simple	Limited address space
Indirect	EA = (A)	Large address space	Multiple memory references
Register	EA = R	No memory reference	Limited address space
Register indirect	EA = (R)	Large address space	Extra memory reference
Displacement	EA = A + (R)	Flexibility	Complexity
Stack	EA = top of stack	No memory reference	Limited applicability



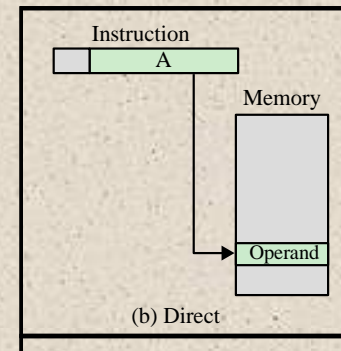


# Immediate Addressing



- En basit adresleme şekli
- Operand = A
- Bu mod, sabitleri tanımlamak ve kullanmak veya değişkenlerin başlangıç değerlerini ayarlamak için kullanılabilir
  - Tipik olarak sayı 2'ye tümleyen formda saklanır
  - Operand alanının en solundaki biti bir işaret biti olarak kullanılır
- Advantage:
  - Operandı elde etmek için komut getirme dışında hiçbir bellek referansı gerekmez, böylece komut döngüsünde bir bellek veya önbellek döngüsünden (*memory or cache cycle*) tasarruf edilir.
- Disadvantage:
  - Sayının boyutu, çoğu komut setinde kelime uzunluğuna (*word length*) kıyasla küçük olan, adres alanının (*address field*) boyutuyla sınırlıdır.

# Direct Addressing



Adres alanı,  
operandın efektif  
adresini içerir

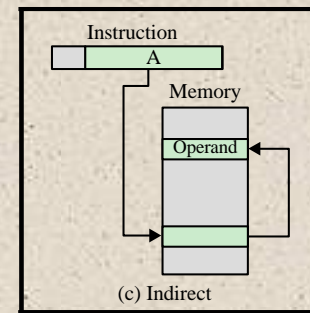
Effective address  
(EA) = address field  
(A)

Eski nesil  
bilgisayarlarda  
yaygındı

Yalnızca bir bellek  
referansı gerektirir  
ve özel bir  
hesaplama  
gerektirmez

Dezavantajı,  
yalnızca sınırlı bir  
adres alanı  
sağlamasıdır

# + Indirect Addressing



- Operandın tam uzunlukta adresini içeren bellekteki bir kelimenin adresine referans
- $EA = (A)$ 
  - Parantezler kastedilen içerikler (*contents*) olarak yorumlanır
- Advantage:
  - $N$  kelime uzunluğu için  $2^N$  adres uzayı artık mevcuttur
- Disadvantage:
  - Komut yürütme, operandı getirmek için iki bellek referansı gerektirir
    - Biri adresini almak ve diğeri değerini almak için
- Nadiren kullanılan bir dolaylı adresleme çeşidi, çok düzeyli veya basamaklı dolaylı adreslemedir (*multilevel or cascaded indirect addressing*).
  - $EA = ( \dots (A) \dots )$
  - Dezavantajı, bir operandı getirmek için üç veya daha fazla bellek başvurusunun gerekli olabilmesidir.



# Register Addressing

Adres alanı, ana bellek adresi yerine bir registra atıfta bulunur

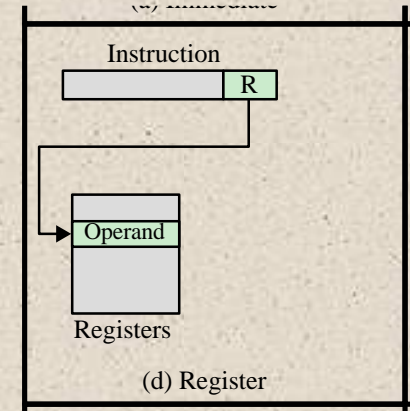
$EA = R$

## Advantages:

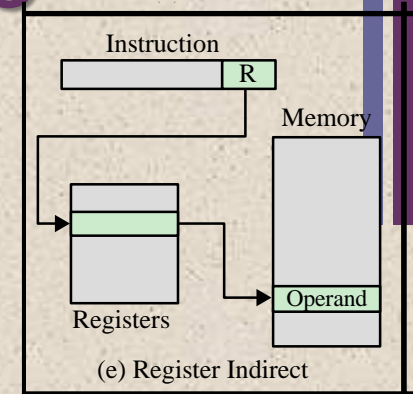
- Komutta sadece küçük bir adres alanına ihtiyaç vardır
- Zaman alan bellek referansları gerektirmez

## Disadvantage:

- Adres alanı çok sınırlıdır



# +Register Indirect Addressing

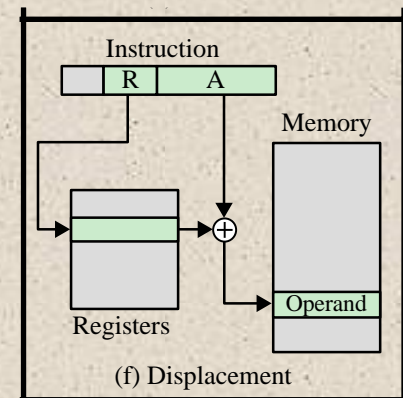


- «indirect addressing» e benzer
  - Tek fark, adres alanının bir bellek konumuna mı yoksa bir register'a mı başvurduğudur.
- $EA = (R)$
- Adres alanının (*address field*) adres uzayı sınırlaması, bu alanın bir adres içeren bir kelime uzunluğundaki konumuna atıfta bulunmasıyla aşılır.
- Dolaylı adreslemeye göre bir tane daha az bellek referansı kullanır



# Displacement Addressing

- «direct addressing» ve «register indirect addressing» in yeteneklerini bir araya getirir
- $EA = A + (R)$
- Komutun en az biri belirgin olmak üzere iki adres alanı olmasını gerektirir
  - Bir adres alanında bulunan değer (value = A) doğrudan kullanılır
  - Diğer adres alanı, efektif adresi üretmek için içeriği A'ya eklenen bir register anlamına gelir.
- Most common uses:
  - Relative addressing
  - Base-register addressing
  - Indexing



# Relative Addressing

For relative addressing, also called PC-relative addressing

İmalı olarak referans verilen register, program sayacıdır (PC)

- Bir sonraki komut adresi, EA'yı oluşturmak için adres alanına eklenir.
- Tipik olarak adres alanı, bu işlem için 2'ye tümleyen sayı olarak değerlendirilir.
- Bu nedenle, efektif adres (EA), komutun adresine göre bir «displacement»tır.

Yerellik kavramını (*concept of locality*) kullanır

Çoğu bellek referansı yürütülen komuta göreceli olarak yakınsa, komuttaki adres bitlerinden tasarruf sağlar.





# Base-Register Addressing



- Referans verilen register, bir ana bellek adresi içerir ve adres alanı, bu adresten itibaren bir «displacement» içerir.
- Register referansı belirgin veya imalı olabilir (*explicit or implicit*)
- Bellek referanslarının yerelliğini kullanır
- Segmentasyonu uygulamanın kullanışlı bir yolu
- Bazı uygulamalarda tek bir «segment base register» görevlendirilir ve imalı olarak kullanılır
- Diğerlerinde, programcı bir segmentin baz adresini tutmak için bir register seçebilir ve komut bunu açıkça/belirgin bir şekilde referans göstermelidir.

# Indexing

- Adres alanı bir ana bellek adresine atıfta bulunur ve başvuru register, bu adresten pozitif bir «displacement» içerir.
- EA'yı hesaplama yöntemi, «*base-register addressing*» ile aynıdır.
- Önemli bir kullanım, iteratif işlemleri gerçekleştirmek için verimli bir mekanizma sağlamaktır.



## ■ Autoindexing

- Her referanstan sonra indeks registerini otomatik olarak artırır veya azaltır
- $EA = A + (R)$
- $(R) \leftarrow (R) + 1$

## ■ Postindexing

- Dolaylamadan sonra indeksleme yapılır (*Indexing is performed after the indirection*)

- $EA = (A) + (R)$  İlk olarak, adres alanının içeriği doğrudan adres içeren bir bellek konumuna erişmek için kullanılır. Bu adres daha sonra registre göre indekslenir. Bu teknik, sabit bir formattaki bir dizi veri bloğundan birine erişmek için kullanışlıdır. Örneğin, Bölüm 8'de işletim sisteminin her proses için bir proses kontrol bloğu (PCB) kullanması gerektiği açıklanmıştır. Gerçekleştirilen işlemler, hangi bloğun manipüle edildiğine bakılmaksızın aynıdır. Bu nedenle, bloğa referans veren komutlardaki adresler, PCB'nin başlangıcını gösteren değişken bir işaretçiyi içeren bir konuma (değer = A) işaret edebilir. İndeks registeri blok içindeki yer değiştirmeyi içerir.

## ■ Preindexing

- İndeksleme, dolaylamadan önce gerçekleştirilir

- $EA = (A + (R))$  Adres, basit indekslemede olduğu gibi hesaplanır. Ancak bu durumda, hesaplanan adres operandı değil, **operandın adresini** içerir. Bu tekniğin kullanımına bir örnek, çok yollu bir dallanma tablosu (**multiway branch table**) oluşturmaktır. Bir programın belirli bir noktasında, koşullara bağlı olarak birkaç konumdan birine bir dallanma olabilir. Konum A'dan başlayarak bir adres tablosu oluşturulabilir. Bu tabloya indekslenerek gerekli konum bulunabilir.

# Indexing

İndekslemenin önemli bir kullanımı, yinelemeli işlemleri gerçekleştirmek için verimli bir mekanizma sağlamaktır.

- Örneğin, A konumundan başlayarak saklanan sayıların bir listesini düşünün. Listedeki her bir öğeye 1 eklemek istediğimizi varsayalım.
- Her bir değeri almalı, ona 1 eklemeli ve geri depolamalıyız.
- İhtiyacımız olan efektif adres dizisi, listedeki son konuma kadar  $A$ ,  $A + 1$ ,  $A + 2$ , ... şeklindedir.
- İndeksleme ile bu kolayca yapılır.
- A değeri, komutun adres alanında saklanır ve bir indeks registeri olarak adlandırılan seçilen register, 0 olarak başlatılır.
- Her işlemten sonra, indeks registeri 1 artırılır.





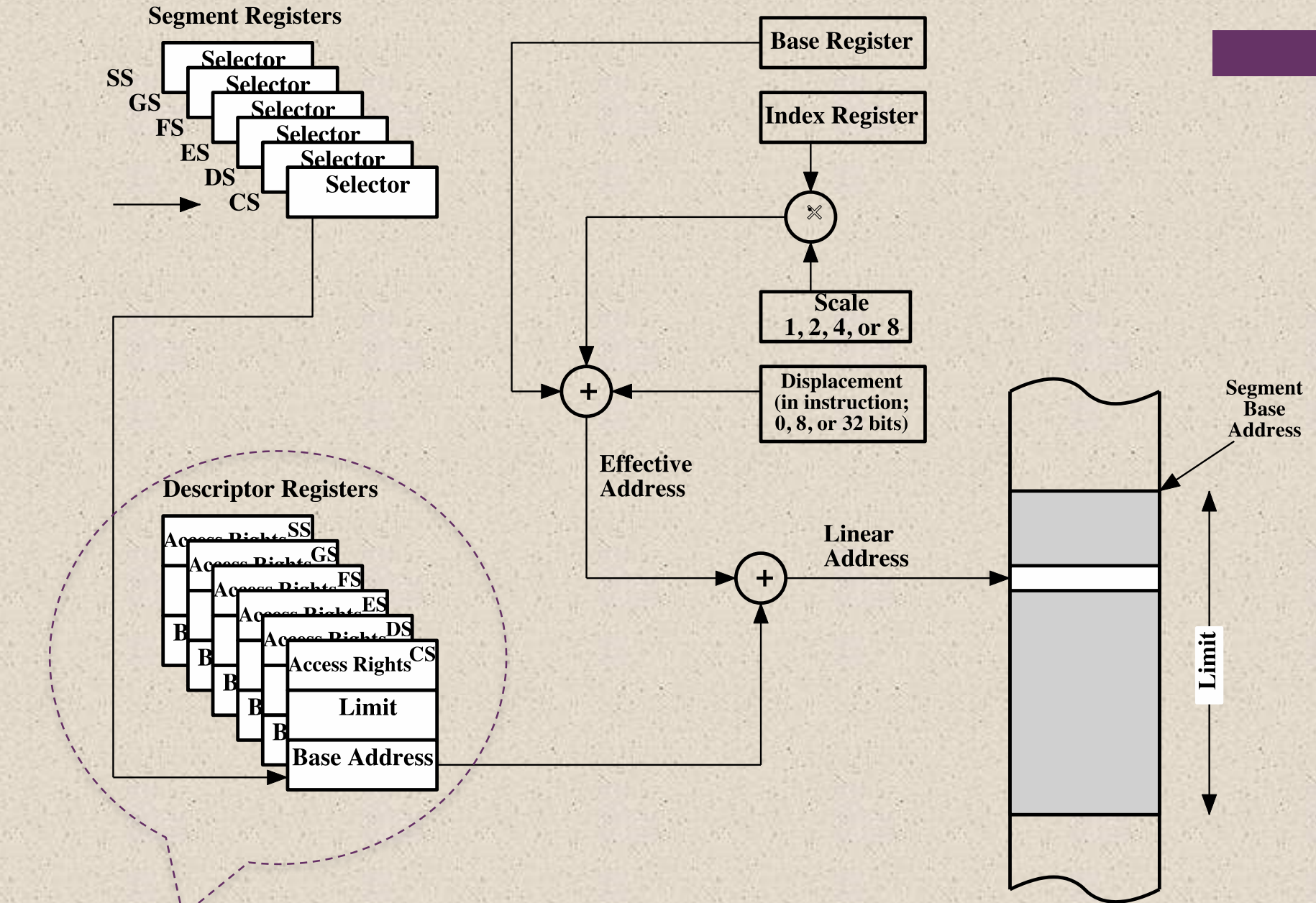


# Stack Addressing

- Yığın, doğrusal bir konum dizisidir
  - Bazen «*pushdown list*» veya «*last-in-first-out queue*» olarak adlandırılır
- Yığın, rezerve edilmiş bir konum bloğudur
  - Öğeler yığının üstüne eklenir, böylece blok kısmen doldurulur
- Yığınla ilişkili olarak, değeri yığının tepesinin adresi olan bir işaretçi vardır
  - Yığın işaretçisi bir register'da tutulur
  - Bu nedenle, bellekteki yığın konumlarına yapılan atıflar aslında «register indirect addresses».
- Bir tür imalı adresleme (*a form of implied addressing*)
- Makine komutlarının bir bellek referansı içermesi gerekmez, ancak imalı olarak yığının tepesinde çalışır.







not programmer visible

**Figure 13.2 x86 Addressing Mode Calculation**

Şekil 8.21'den, x86 adres dönüşüm mekanizmasının (*x86 address translation mechanism*), sanal veya efektif adres adı verilen ve «an offset into a segment» şeklinde bir adres ürettiğini hatırlayın. Segmentin başlangıç adresi ile etkin adresin toplamı doğrusal bir adres (*linear address*) üretir. Sayfalama (*Paging*) kullanılıyorsa, bu doğrusal adres, fiziksel bir adres üretmek için bir sayfa dönüştürme (*page-translation*) mekanizmasından geçmelidir.

x86, yüksek seviyeli dillerin verimli bir şekilde yürütülmesine izin vermeyi amaçlayan çeşitli adresleme modlarıyla donatılmıştır. Şekil 13.2, ilgili lojiği gösterir.

Segment registeri, referansın konusu olan segmenti belirler.

- Altı segment registeri vardır; belirli bir referans için kullanılan, yürütme context'ine ve komuta bağlıdır.

Her segment registeri, ilgili segmentlerin başlangıç adresini tutan segment tanımlayıcı tablosunda (*segment descriptor table*) bir indeks tutar.

Kullanıcı tarafından görülebilen her segment registeri ile ilişkili olan, segmentin erişim haklarını ve ayrıca segmentin başlangıç adresini ve limitini (uzunluğunu) kaydeden bir segment tanımlayıcı registeridir (programcı tarafından görülmez).

Ayrıca, bir adresin oluşturulmasında kullanılabilecek iki register vardır: **base register** ve **index register**.

# Table 13.2

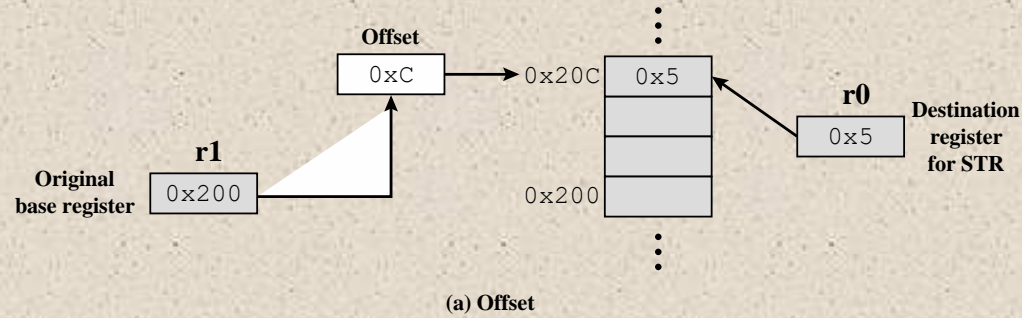
## x86 Addressing Modes

Mode	Algorithm
Immediate	$\text{Operand} = A$
Register Operand	$\text{LA} = R$
Displacement	$\text{LA} = (\text{SR}) + A$
Base	$\text{LA} = (\text{SR}) + (B)$
Base with Displacement	$\text{LA} = (\text{SR}) + (B) + A$
Scaled Index with Displacement	$\text{LA} = (\text{SR}) + (I) \cdot S + A$
Base with Index and Displacement	$\text{LA} = (\text{SR}) + (B) + (I) + A$
Base with Scaled Index and Displacement	$\text{LA} = (\text{SR}) + (I) \cdot S + (B) + A$
Relative	$\text{LA} = (\text{PC}) + A$

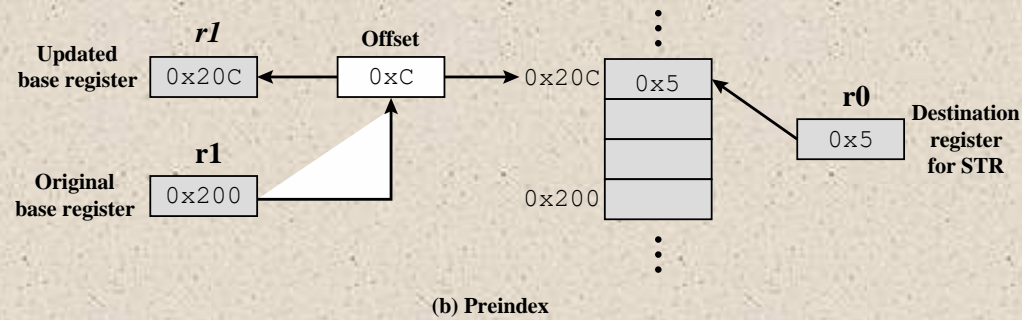
LA = linear address  
 (X) = contents of X  
 SR = segment register  
 PC = program counter  
 A = contents of an address field in the instruction  
 R = register  
 B = base register  
 I = index register  
 S = scaling factor

«scaled index with displacement mode», komut bir registere eklenecek bir «displacement» içerir, bu durumda bu bir indeks registeri olarak anılır. İndeks registeri, genellikle yığın işleme için kullanılan ESP dışında genel amaçlı registerlerden herhangi biri olabilir. Efektif adres hesaplanırken, indeks registerinin içeriği 1, 2, 4 veya 8'lik bir ölçekleme faktörü ile çarpılır ve sonra bir «displacement»a eklenir. Bu mod, dizileri indekslemek için çok uygundur. 16 bitlik bir tamsayı dizisi için ölçekleme faktörü 2 kullanılabilir. 32 bitlik tam sayılar veya kayan noktalı sayılar için 4 ölçekleme faktörü kullanılabilir. Son olarak, bir dizi çift duyarlılıklı kayan noktalı sayılar için 8 ölçekleme faktörü kullanılabilir.

STRB r0, [r1, #12]



STRB r0, [r1, #12]!



STRB r0, [r1], #12

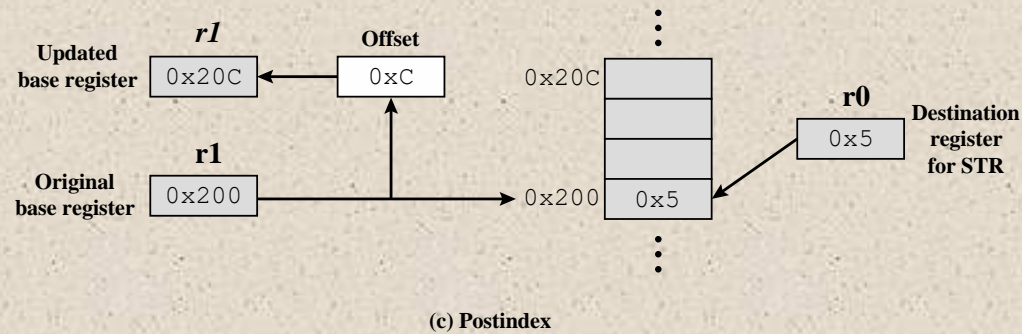


Figure 13.3 ARM Indexing Methods



# Instruction Formats

Bir komutun bitlerinin düzenini, onu oluşturan alanlar açısından tanımlar

Bir opcode içermeli ve imalı veya belirgin bir şekilde her operand için adresleme modunu belirtmelidir

Çoğu komut seti için birden fazla komut formatı kullanılır

# + Instruction Length

- En temel tasarım meselesi
- Şunları etkiler ve şulardan etkilenir:
  - Memory size
  - Memory organization
  - Bus structure
  - Processor complexity
  - Processor speed
- Bellek transfer uzunluğuna eşit olmalı veya biri diğerinin katı olmalıdır
- Genellikle 8 bit olan karakter uzunluğunun ve sabit-noktalı sayıların uzunluğunun katı olmalıdır

# Allocation of Bits

Aşağıdaki birbiriyle ilişkili faktörler, adresleme bitlerinin kullanımının belirlenmesinde rol oynar.

Number of  
addressing  
modes

Number of  
operands

Register  
versus  
memory

Number of  
register sets

Address  
range

Address  
granularity

## Register versus memory:

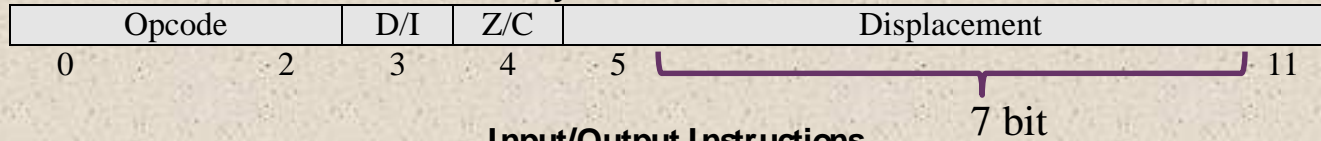
Bir makine, verilerin işlenmek üzere işlemciye getirilebilmesi için registerlara sahip olmalıdır. Kullanıcı tarafından görülebilen tek bir registerla (genellikle akümülatör olarak adlandırılır), bir işlenen adresi imalıdır/örtüktür ve komut biti tüketmez. Ancak, tek registerlı programlama kullanışsızdır ve birden fazla komut gerektirir. Çoğu modern mimaride en az 32 adet kullanıcı tarafından görülebilir register vardır.

## Address granularity:

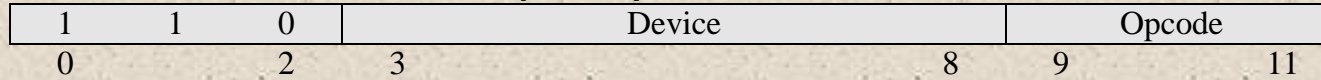
Bayt adresleme, karakter manipülasyonu için uygundur, ancak sabit boyutlu bir bellek için daha fazla adres biti gerektirir.

Bellek, her biri  $2^7 = 128$  kelimelik sabit uzunlukta sayfalara bölünmüştür.

### Memory Reference Instructions

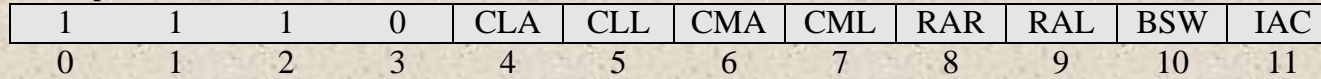


### Input/Output Instructions

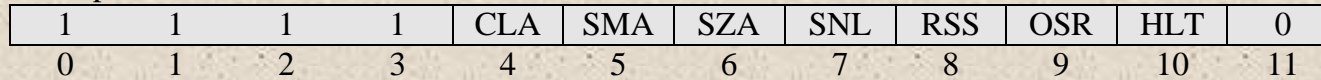


### Register Reference Instructions

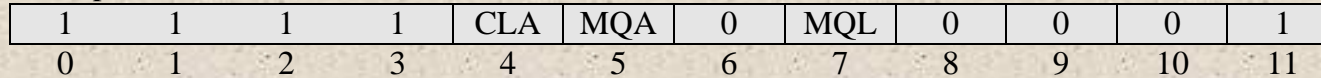
#### Group 1 Microinstructions



#### Group 2 Microinstructions



#### Group 3 Microinstructions



D/I = Direct/Indirect address

Z/C = Page 0 or Current page

CLA = Clear Accumulator

CLL = Clear Link

CMA = CoMplement Accumulator

CML = CoMplement Link

RAR = Rotate Accumulator Right

RAL = Rotate Accumulator Left

BSW = Byte SWap

IAC = Increment ACcumulator

SMA = Skip on Minus Accumulator

SZA = Skip on Zero Accumulator

SNL = Skip on Nonzero Link

RSS = Reverse Skip Sense

OSR = Or with Switch Register

HLT = HaLT

MQA = Multiplier Quotient into Accumulator

MLQ = Multiplier Quotient Load

Genel amaçlı bir bilgisayar için en basit komut tasarımlarından biri PDP-8 içindi. PDP-8, 12 bitlik komutlar kullanır ve 12 bit kelimelerle çalışır.

Akümülatör adında tek bir genel amaçlı register vardır.

3 bitlik bir opcode ve üç tür komut vardır.

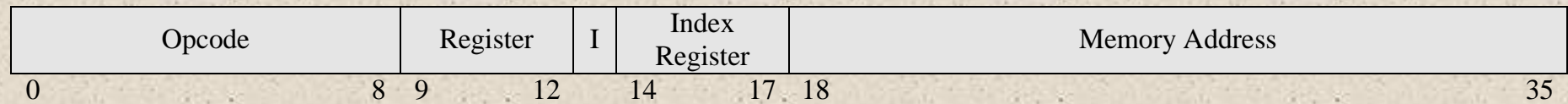
0 ile 5 arasındaki opcode'lar için format, bir sayfa biti ve dolaylı bir bit içeren tek adresli bir bellek referans komutudur (single-address memory reference).

İşlem grubunu büyötmek için, opcode 7 bir register referansı veya mikrokomutu tanımlar.

Bu formatta, kalan bitler ek işlemleri kodlamak için kullanılır. Genel olarak, her bir bit belirli bir işlemi tanımlar (örneğin, clear accumulator) ve bu bitler tek bir komutta birleştirilebilir.

**Figure 13.5 PDP-8 Instruction Formats**





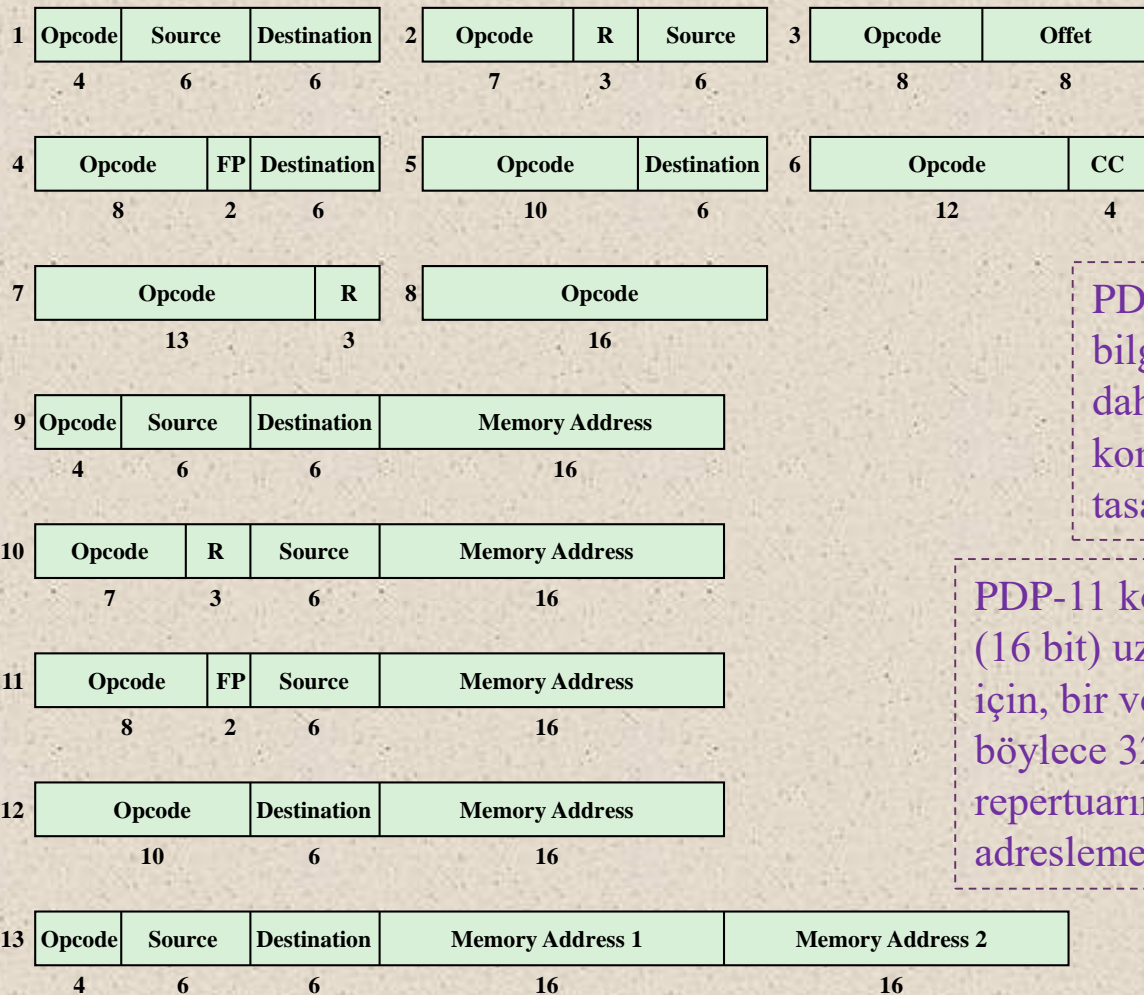
I = indirect bit

**Figure 13.6 PDP-10 Instruction Format**

# + Variable-Length Instructions

Şimdiye kadar incelediğimiz örnekler tek bir sabit komut uzunluğu kullandı. Ancak tasarımcı bunun yerine farklı uzunluklarda çeşitli komut formatları sağlamayı seçebilir. Bu taktik, farklı opcode uzunluklarına sahip geniş bir opcode repertuarı sağlamayı kolaylaştırır. Adresleme, çeşitli register ve bellek referans kombinasyonları artı adresleme modları ile daha esnek olabilir. Değişken uzunluklu komutlarla, bu birçok varyasyon verimli ve kompakt bir şekilde sağlanabilir.

- Varyasyonlar verimli ve kompakt bir şekilde sağlanabilir
- İşlemcinin karmaşıklığını artırır
  - Değişken uzunluktaki komutlar için ödenecek ana bedel budur
  - Düşen donanım fiyatları, mikro programlamanın kullanımı ve işlemci tasarımının ilkelerinin anlaşılmasındaki genel artış, bunun ödenmesi gereken küçük bir bedel olmasına katkıda bulunmuştur.
  - Bununla birlikte, RISC ve süper skalar makinelerin gelişmiş performans sağlamak için sabit uzunlukta komutların kullanımından yararlanabileceğini göreceğiz.
- Değişken uzunluklu komutların kullanılması, tüm komut uzunluklarının kelime uzunluğu ile bütünsel olarak ilişkili hale getirilmesi arzusunun ortadan kaldırmaz.
  - İşlemci, alınacak bir sonraki komutun uzunluğunu bilmediğinden, tipik bir strateji, en azından olası en uzun komuta eşit sayıda bayt veya kelimeyi getirmektir. Bu, bazen birden fazla komutun getirildiği anlamına gelir.



Numbers below fields indicate bit length

Source and Destination each contain a 3-bit addressing mode field and a 3-bit register number

FP indicates one of four floating-point registers

R indicates one of the general-purpose registers

CC is the condition code field

**Figure 13.7 Instruction Formats for the PDP-11**

PDP-11, 16 bitlik bir mini bilgisayarın kısıtlamaları dahilinde güçlü ve esnek bir komut seti sağlamak üzere tasarlanmıştır.

PDP-11 komutları genellikle bir kelime (16 bit) uzunluğundadır. Bazı komutlar için, bir veya iki bellek adresi eklenir, böylece 32-bit ve 48-bit komutlar repertuarın bir parçasıdır. Bu, adreslemede daha fazla esneklik sağlar.

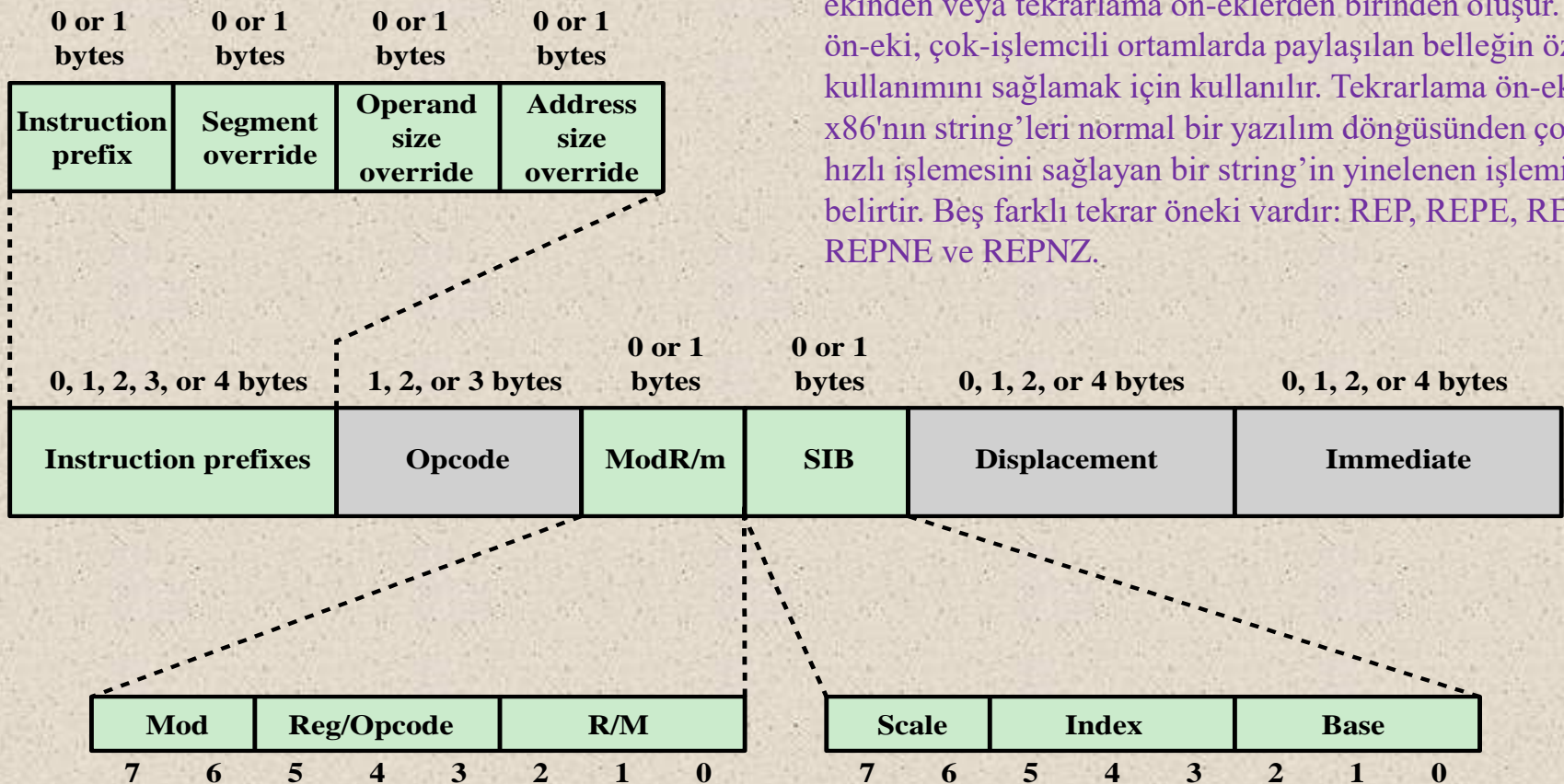
PDP-11 komut seti ve adresleme yeteneği karmaşıktır. Bu, hem donanım maliyetini hem de programlama karmaşıklığını artırır. Avantajı, daha verimli veya kompakt programların geliştirilebilmesidir.

Hexadecimal Format	Explanation	Assembler Notation and Description												
<div><div>8 bits</div><table><tr><td>0</td><td>5</td></tr></table></div>	0	5	Opcode for RSB	RSB Return from subroutine										
0	5													
<table><tr><td>D</td><td>4</td></tr><tr><td>5</td><td>9</td></tr></table>	D	4	5	9	Opcode for CLRL Register R9	CLRL R9 Clear register R9								
D	4													
5	9													
<table><tr><td>B</td><td>0</td></tr><tr><td>C</td><td>4</td></tr><tr><td>6</td><td>4</td></tr><tr><td>0</td><td>1</td></tr><tr><td>A</td><td>B</td></tr><tr><td>1</td><td>9</td></tr></table>	B	0	C	4	6	4	0	1	A	B	1	9	Opcode for MOVW Word displacement mode, Register R4 356 in hexadecimal Byte displacement mode, Register R11 25 in hexadecimal	MOVW 356(R4), 25(R11) Move a word from address that is 356 plus contents of R4 to address that is 25 plus contents of R11
B	0													
C	4													
6	4													
0	1													
A	B													
1	9													
<table><tr><td>C</td><td>1</td></tr><tr><td>0</td><td>5</td></tr><tr><td>5</td><td>0</td></tr><tr><td>4</td><td>2</td></tr><tr><td>D</td><td>F</td></tr><tr><td colspan="2"></td></tr></table>	C	1	0	5	5	0	4	2	D	F			Opcode for ADDL3 Short literal 5 Register mode R0 Index prefix R2 Indirect word relative (displacement from PC) Amount of displacement from PC relative to location A	ADDL3 #5, R0, @A[R2] Add 5 to a 32-bit integer in R0 and store the result in location whose address is sum of A and 4 times the contents of R2
C	1													
0	5													
5	0													
4	2													
D	F													

Sonuç, oldukça değişken bir komut formatıdır. Bir komut, işlem koduna bağlı olarak 1 veya 2 baytlık bir opcode ve ardından sıfırdan altıya kadar operand belirleyiciden oluşur. Minimum komut uzunluğu 1 bayttır ve 37 bayta kadar komutlar oluşturulabilir. Şekil 13.8 birkaç örnek VAX komutunu verir.

**Figure 13.8 Examples of VAX Instructions**





■ **Instruction prefixes:** Eğer mevcutsa, komut LOCK ön-ekinden veya tekrarlama ön-eklerden birinden oluşur. LOCK ön-eki, çok-işlemcili ortamlarda paylaşılan belleğin özel kullanımını sağlamak için kullanılır. Tekrarlama ön-ekleri, x86'nın string'leri normal bir yazılım döngüsünden çok daha hızlı işlemlerini sağlayan bir string'in yinelenen işlemini belirtir. Beş farklı tekrar öneki vardır: REP, REPE, REPZ, REPNE ve REPNZ.

**Figure 13.9 x86 Instruction Format**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
data processing immediate shift	cond			0 0 0			opcode			S	Rn			Rd			shift amount				shift		0		Rm							
data processing register shift	cond			0 0 0			opcode			S	Rn			Rd			Rs			0		shift		1		Rm						
data processing immediate	cond			0 0 1			opcode			S	Rn			Rd			rotate			immediate												
load/store immediate offset	cond			0 1 0			P	U	B	W	L	Rn			Rd			immediate														
load/store register offset	cond			0 1 1			P	U	B	W	L	Rn			Rd			shift amount				shift		0		Rm						
load/store multiple	cond			1 0 0			P	U	S	W	L	Rn			register list																	
branch/branch with link	cond			1 0 1			L		24-bit offset																							

S = For data processing instructions, signifies that the instruction updates the condition codes

S = For load/store multiple instructions, signifies whether instruction execution is restricted to supervisor mode

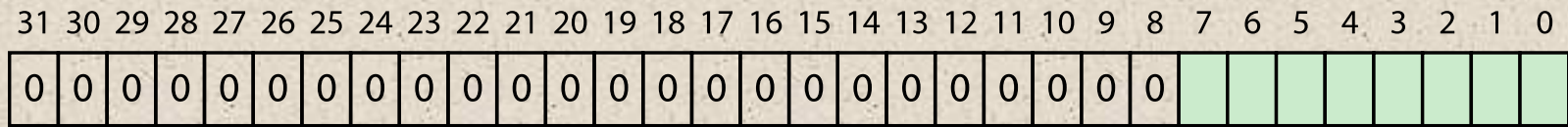
P, U, W = bits that distinguish among different types of addressing\_mode

B = Distinguishes between an unsigned byte (B==1) and a word (B==0) access

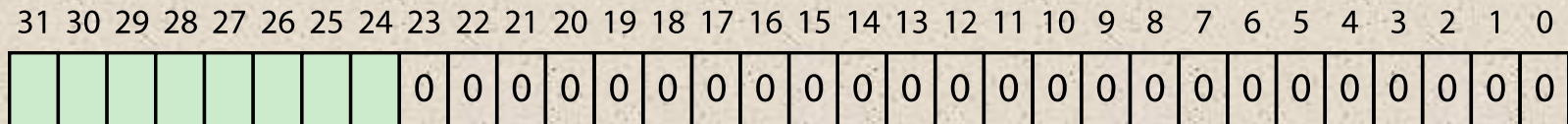
L = For load/store instructions, distinguishes between a Load (L==1) and a Store (L==0)

L = For branch instructions, determines whether a return address is stored in the link register

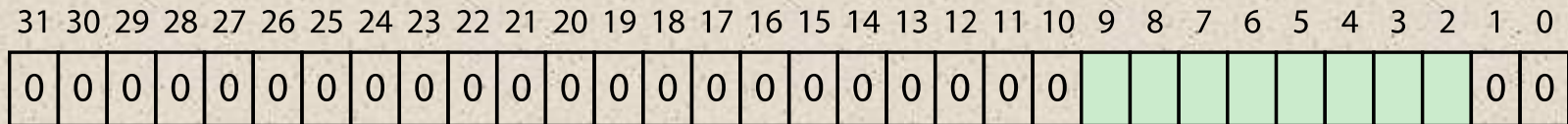
**Figure 13.10 ARM Instruction Formats**



ror #0 - range 0 through 0x000000FF - step 0x00000001



ror #8 - range 0 through 0xFF000000 - step 0x01000000



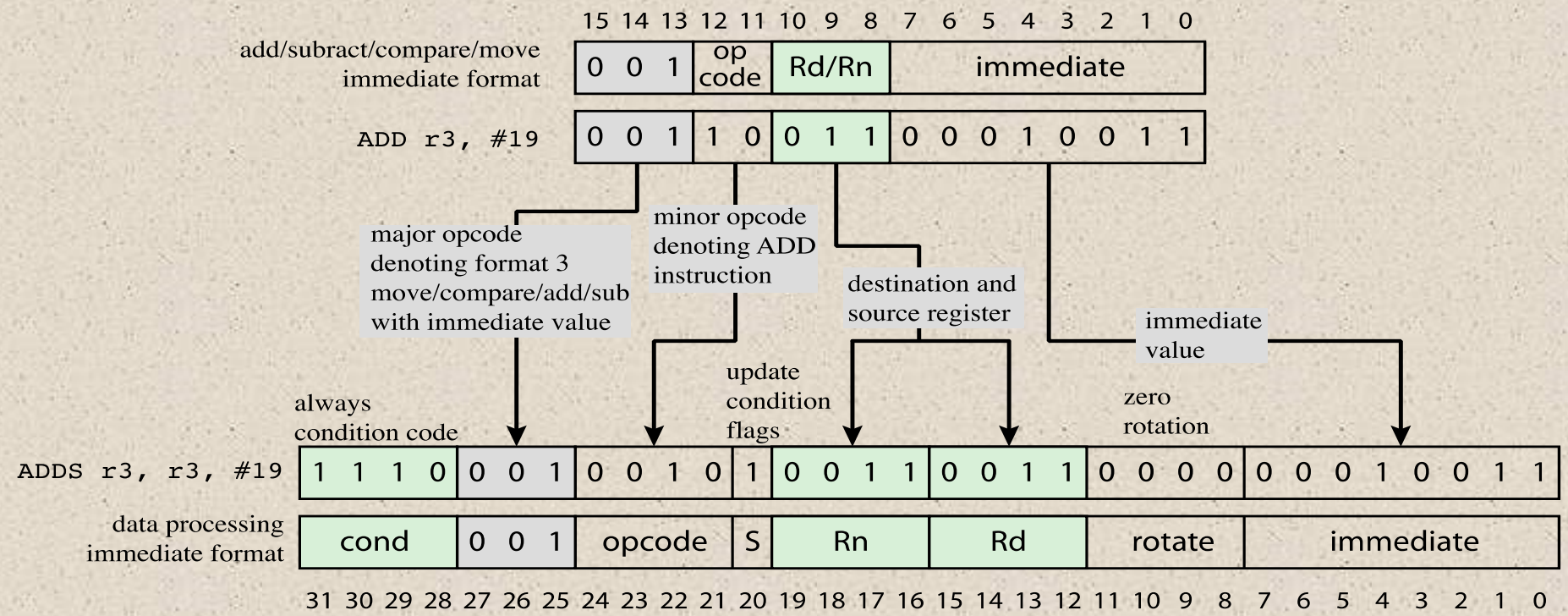
ror #30 - range 0 through 0x000003FC - step 0x00000004

**Figure 13.11 Examples of Use of ARM Immediate Constants**

Thumb komut seti, ARM komut setinin yeniden kodlanmış bir alt kümesidir. Thumb, 16 bit veya daha dar bir bellek veri yolu kullanan ARM uygulamalarının performansını artırmak ve ARM komut seti tarafından sağlanandan daha iyi kod yoğunluğuna izin vermek için tasarlanmıştır.

Thumb komut seti, 16 bitlik komutlarla kodlanmış ARM 32-bit komut setinin bir alt kümesini içerir. Tasarruf şu şekilde sağlanır:

Örneğin Thumb komutları koşulsuzdur, bu nedenle koşul kodu alanı (condition code) kullanılmaz. Ayrıca, tüm Thumb aritmetik ve lojik komutları durum bayraklarını günceller, böylece güncelleme bayrağı bitine gerek kalmaz. Tasarruf: 5 bit.



**Figure 13.12 Expanding a Thumb ADD Instruction into its ARM Equivalent**

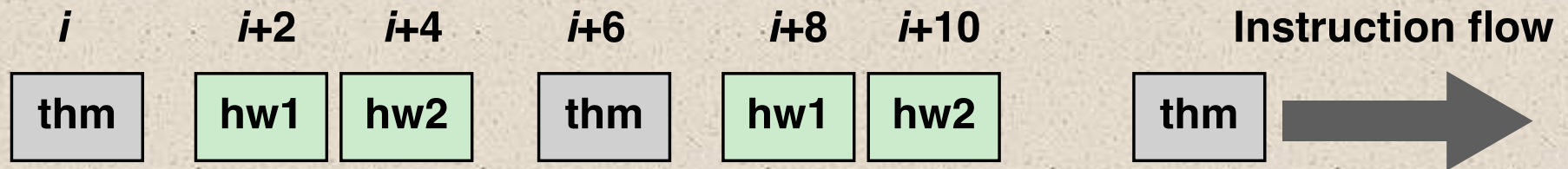




# Thumb-2 Instruction Set



- The only instruction set available on the Cortex-M microcontroller products
- Is a major enhancement to the Thumb instruction set architecture (ISA)
  - Introduces 32-bit instructions that can be intermixed freely with the older 16-bit Thumb instructions
  - Most 32-bit Thumb instructions are unconditional, whereas almost all ARM instructions can be conditional
  - Introduces a new If-Then (IT) instruction that delivers much of the functionality of the condition field in ARM instructions
- Delivers overall code density comparable with Thumb, together with the performance levels associated with the ARM ISA
- Before Thumb-2 developers had to choose between Thumb for size and ARM for performance



Halfword 1 [15:13]	Halfword1 [12:11]	Length	Functionality
Not 111	xx	16 bits (1 halfword)	16-bit Thumb instruction
111	00	16 bits (1 halfword)	16-bit Thumb unconditional branch instruction
111	Not 00	32 bits (2 halfwords)	32-bit Thumb-2 instruction

**Figure 13.13 Thumb-2 Encoding**

Address		Contents		
101	0010	0010	101	2201
102	0001	0010	102	1202
103	0001	0010	103	1203
104	0011	0010	104	3204
201	0000	0000	201	0002
202	0000	0000	202	0003
203	0000	0000	203	0004
204	0000	0000	204	0000

(a) Binary program

Address	Contents
101	2201
102	1202
103	1203
104	3204
201	0002
202	0003
203	0004
204	0000

(b) Hexadecimal program

Address	Instruction	
101	LDA	201
102	ADD	202
103	ADD	203
104	STA	204
201	DAT	2
202	DAT	3
203	DAT	4
204	DAT	0

(c) Symbolic program

Label	Operation	Operand
FORMUL	LDA	I
	ADD	J
	ADD	K
	STA	N
I	DATA	2
J	DATA	3
K	DATA	4
N	DATA	0

(d) Assembly program

**Figure 13.14 Computation of the Formula  $N = I + J + K$**

# + Summary

## Chapter 13

- Addressing modes
  - Immediate addressing
  - Direct addressing
  - Indirect addressing
  - Register addressing
  - Register indirect addressing
  - Displacement addressing
  - Stack addressing
- Assembly language

## Instruction Sets: Addressing Modes and Formats

- x86 addressing modes
- ARM addressing modes
- Instruction formats
  - Instruction length
  - Allocation of bits
  - Variable-length instructions
- X86 instruction formats
- ARM instruction formats