Bilgisayara Giriş

V. Erkcan Özcan
Boğaziçi Üniversitesi
Fizik Bölümü

HPFBU'14, 3 Şubat 2014

Amaç

- Doğru sonuçlar veren, istenileni yerine getiren, hızlı çalışan, kolay taşınabilir, sonradan kullanılabilir bilgisayar programları yazmak sanat değil, zanaattir ve bu zanaati öğrenmenin ilk adımı temel bilgileri iyi oturtmaktan geçer.
- Eğer deneyci bir doktora öğrencisinin yazdığı kod, danışmanının yazdığı koddan daha kötüyse, bu öğrenci daha yeterince pişmemiş demektir.
- ~8 yıl kimya okuyup, her türlü kimya problemini çözebilip ama canlıları cansızlardan ayıran atomun hangisi olduğu bilmemek ne kadar vahimse, bilgisayar kullanıp onu hiç anlamamak o kadar vahimdir ve sizi sağlam işler yapmaktan alıkoyabilir!

Bu sunumda bu şekilde sağ alt köşede bulacağınız gri ile yazılmış notlar, daha sonradan işinize yarabilecek ek bilgilerdir.

Bilgisayar Nedir?

- Modern bir bilgisayar hangi parçalardan oluşur?
  - Kasa + çevre birimler (perifer): ekran, klavye, fare
- Kasanın içinde ne vardır?
  - İşlemci, bellek, sabit disk, ana kart, ses kartı, görüntü kartı, vs. vs.
- Pekiyi bilgisayarın içinde en temel ne vardır?
  - Silisyum: $_{14}$Si periyodik tabloda(yinelemeli dizin) karbonun altında => dört valans elektronu => yarı-iletken olabilecek özellikte
  - Yarı iletken => transistör yapılabilir. Transistör = elektrik anahtarı: voltaja bağlı olarak aç ve kapa. => Açık ve kapalı, iki ayrı durum: 0 ve 1.

İkili değil üçlü sistem kullanan bilgisayarlar da yapılmıştır. Bknz. dengeli üçlü sistem (-1,0,1) kullanan Setun.

İkili Sistem

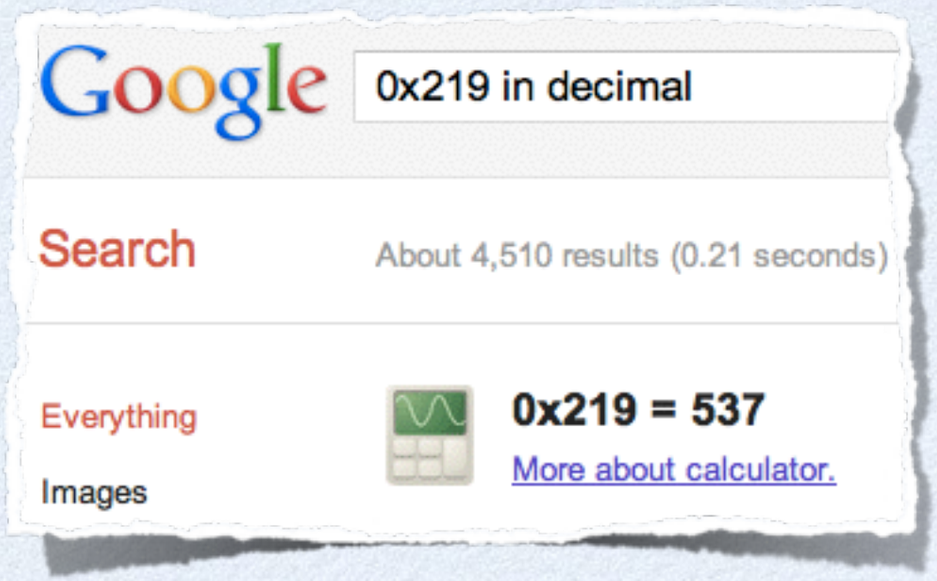
- (Kapalı ve açık) = (0 ve 1) = (iki farklı sembol) ile tüm sayıları ifade etmek mümkündür.
  - 00001, 00011, 00111, 01111, 11111 => Kaç çubuk o kadar sayı. => 537 yazmak için 537 çubuk.
- Ondalık sistem: On farklı sembol = rakam = (0, 1, 2, ⋮, 9)
  - Otuz beş yazmak için 9998 yazmıyoruz.
  - Romen rakamları aslında bu şekilde ama her yeni $1x10^n$ ve $5x10^n$ için yeni bir sembol uydur: XXXIV
  - Daha iyisi: $35 = 3x10^1+5x10^0$ => Basamakların yerleri kaç sembol varsa (10) onun üssünü ifade ediyor.
- İkili sistem: $(35)_{ondalık}=(100011)_{ikilik}$ (35 çubuk değil, 6 basamak)

İkilikten Onluğa

- $(100011)_2 = (1x2^5 + 0x2^4 + 0x2^3 + 0x2^2 + 1x2^1 + 1x2^0)_{10}$
- Tersi? $(537)_{10} = (512 + 25 = 512 + 16 + 9)_{10}$
  $= (512 + 16 + 8 + 1)_{10} = (1000011001)_2$
  - İkinin üslerini bilmekte fayda var.
- İyi de bu kadar çok sıfır ve bir yazmak zor ve insan açısından hataya düşürebilecek bir şey, hele 32-64 bitlik (sayıların 32-64 ikilik basamakla ifade edilebileceği) modern makinalarda...
  - Çözüm: onaltılık sistem. Yanyana her dört ikilik sistem basamağını tek bir simge ile ifade etmek.

Onaltılık Sistem

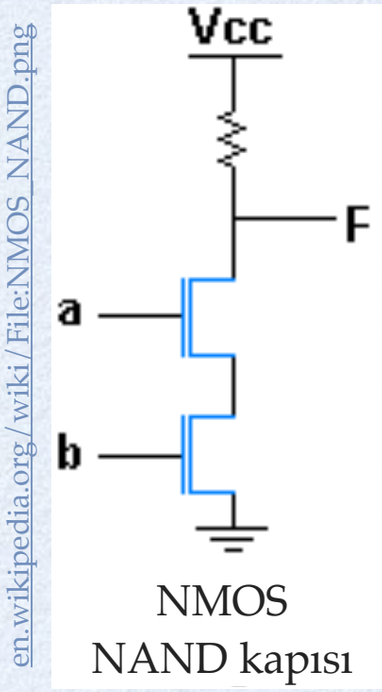
- 16 sembol: 0, 1, 2, …, 9, A, B, C, D, E, F
- İkilik sistemde yazılmış sayısı çevirmek için basamakları dörtlü gruplar halinde yaz ve onaltılık sembolle değiştir.
- $(0010\ 0001\ \ 1001)_2 = (219)_{16}$
- Genellikle 0x diye başlanarak yazılır: 0x219



ikilik	onaltılık	ondalık
(0000)	0	0
(0001)	1	1
(0010)	2	2
(0011)	3	3
(0100)	4	4
(0101)	5	5
(0110)	6	6
(0111)	7	7
(1000)	8	8
(1001)	9	9
(1010)	A	10
(1011)	B	11
(1100)	C	12
(1101)	D	13
(1110)	E	14
(1111)	F	15
0010 0001 1001	219	537

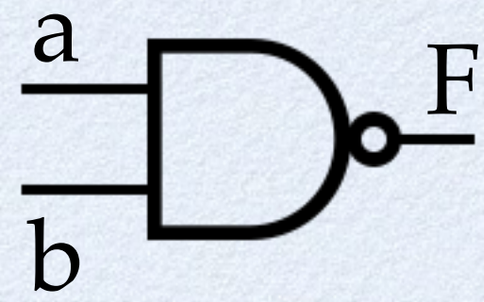
0 ve 1 ile İşlem

- 0 ve 1 ile istediğimiz sayıları yazabiliyoruz. Pekiyi işlemler?
- Kapılar iki biti (0 veya 1 durumu) alıp, onlardan sonuç çıkarmayı sağlar.
  - Örneğin NAND kapısı: a veya b'nin ancak her ikisine birden gerilim (bir) uygulanırsa, F topraklanmış (sıfır) olur.
- Kapılar bir arada kullanılarak işlemler yapılabilir. Örneğin A ve B'nin toplamı, "A xor B", toplamada "elde var" ise "A and B" diye yapılabilir.

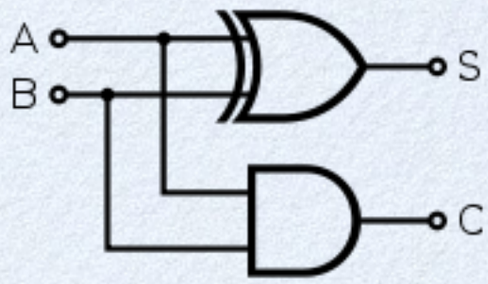


en.wikipedia.org/wiki/File:NMOS_NAND.png

NMOS NAND kapısı



a	b	F
0	0	1
0	1	1
1	0	1
1	1	0

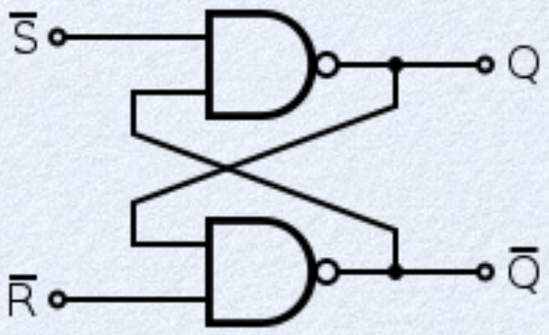


(Yarım/elde girilmeyen) toplama devresi

Sadece NAND kapıları kullanılarak bütün bir bilgisayar yapılabilir. (NAND fonksiyonel bütünlüğü olan bir kapıdır.)

Bellek

- Kapılar geri beslemeli kullanılarak kendi değerini koruyan devreler (**flipflop**) yapılabilir.
- Bir çok flipflop bir arada kullanılarak **SRAM** (static random access memory) yapılabilir.
- Çağdaş PClerdeki belleğin çoğunu ise **DRAM** (dynamic random access memory) oluşturur. Her bir bitin değeri küçük bir kapasitörde yük olup olmamasına göre belirlenir.



S̄	R̄	Görevi
0	0	Kullanılmaz
0	1	Q=1
1	0	Q=0
1	1	Değiştirme

S̄R̄ flipflopu: Q sıfırken, Q̄, R̄'dan bağımsız olarak bir olmak zorundadır (NAND kapısı). Eğer S̄ girişine sıfır, R̄ girişine de aynı anda bir verilirse, Q bire döner, 1 nand 1 sıfır olduğundan Q̄ da sıfıra.

Kapasitörlerdeki yük yavaş yavaş kendiliğinden boşaldığından DRAM'lerdeki bitlerin düzenli olarak yeniden yazılması gerekir. Ancak DRAM daha ucuzdur: her bir bit sadece bir kapasitör ve bir transistörle depolanır.

İşlem Kodları

- Bellekte sadece 0 ve 1 serileri tutuyoruz. Bir program da dolayısıyla sadece 0 ve 1'lerden oluşmalı. Programın verilerini (sayılar) ikilik sistemde yazdık. Bu sayıların üzerinde yapılacak işlemleri de 0 ve 1'ler cinsinden yazmak için bilgisayarda devresini kurduğumuz her temel işlem için (and, or, ekle, tersini al, vs.) bir numara atıyoruz: İşlem kodları (**opkodları - opcodes**)
  - Örneğin topla 000, çıkart 001, kaydet 010, and 011, or 100, xor 101, : diye işlem kodları belirlenmiş olsun.
  - 3 ile 5i topla, cevabı bellekteki 0x34 numaralı adrese yaz: Add 3 5 ; Store 0x34 => 000 0011 0101 010 0011 0100 => Programımız **makine dili**ne çevrilmiş oldu.

Programlama Dilleri

- Makine dilinde program yazmak güç ve hataya meyilli. Ayrıca farklı işlemcilerin opkodları farklı farklı. Aynı işi yapan programın her makinede ayrı yazılması dert.
  - İnsan tarafından okunabilir şekilde yazılmış algoritmaları (**kaynak kodu**) makine diline çevirecek programlar yazarak işi kolaylaştırmak mümkün.
  - İnsan tarafından okunabilir = Kendi basit dilbilgisi kuralları ve kelime dağarcığı olan diller. BASIC, C, Scheme, vs.
  - **Yorumlayıcı** (interpreter): Makine diline çevirme işini kaynak kodunu satır satır inceledikçe yapan program ve çevrilen kodları anında işleten program.
  - **Derleyici** (compiler): Kaynak kodunun tamamını alıp makine diline çeviren program. Bu şekilde oluşturulmuş **binary** dosyayı dilediğimizde işletebiliriz.

Byte-code derleyicileri gibi tanım olarak yorumlayıcı ile derleyici arasında olan programlar da mevcut.

Harfler vs.

```
#include<stdio.h>
main() { printf("Merhaba Dunya"); }
```

```
89 C7 E8 19 00 00 00 F4 90 90 90 90 55 48 89 E5 48 8D 3D 39 00 00 00 30 C0 E8 08 00 00 00 5D
C3 FF 25 0E 01 00 00 FF 25 10 01 00 00 68 00 00 00 00 E9 0A 00 00 00 68 0C 00 00 00 E9 00 00
00 00 4C 8D 1D E5 00 00 00 41 53 FF 25 D5 00 00 00 90 4D 65 72 68 61 62 61 20 44 75 6E 79 61
00 01 00 00 00 1C 00 00 00 00 00 00 00 1C 00 00 00 00 00 00 00 1C 00 00 00 02 00 00 00 D0 0E
00 00 34 00 00 00 34 00 00 00 25 0F 00 00 00 00 00 00 34 00 00 00 03 00 00 00 0C 00 02 00 14
00 02 00 00 00 00 01 40 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 14 00 00 00 00 00
00 00 01 7A 52 00 01 78 10 01 10 0C 07 08 90 01 00 00 2C 00 00 00 1C 00 00 00 38 FF FF FF FF
```

- Makine dünyasında herşeyin 0 ve 1'lerden ibaret olduğunu unutmamak gerek.
- Grafik kartı ekranda "Merhaba Dunya" harflerinin çıkması için gerekli elektriksel sinyalleri yollamakla yükümlü. Onun kabul edeceği bir standard harf tablosu lazım. Örneğin ASCII.
  - 'M' = 0x4d = 01001101 = $(77)_{10}$

ASCII Code Chart

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

ASCII'de toplam $128=2^7$ karakter var, yani 7 bit ile ifade edilebilir tam karakterler. Ancak bunun üstüne genellikle ülkeden ülkeye değişen 128 karakter daha ekleniyordu (extended ASCII). Böylece her bir karakter bir bayt = 8 bit ile ifade edilir oldu.

Günümüzde ise her bir karakterin (ortalama) iki bayt ile ifade edildiği UNICODE var.

İşletim Sistemi

- Derleyicimiz olsa bile bilgisayara faydalı bir iş yaptırabilmek için yazmamız gereken bir sürü kodlar olacak.
  - Örneğin sabit diskten okuma, diske yazma, ekrana şekiller çizdirme, aynı anda çalışacak programların öncelik sırasını belirleme, ağa bağlanma, klavyeden/fareden gelen elektriksel sinyalleri yorumlama, vs. vs.
- Bütün bu işleri yapan programa (programlar yığınına) işletim sistemi diyoruz.
  - Örnekler: Windows 95/XP/Vista, GNU/Linux'un değişik dağıtımları (Ubuntu, Mandriva, Pardus, Scientific vs.), Free/Net/OpenBSD, Mac OSX, Solaris, Android (Linux), ...

linux?

- Windows gibi kapalı kaynak kodlu işletim sistemlerinde beğenmediğiniz birşey, bulduğunuz bir eksik veya hata (bug) varsa, yapacağınız tek şey, programcı firmaya şikayet yazmaktır.
- İşletim sisteminin kodu açık ise, dilediğiniz parçalarını değiştirir, düzeltir, daha iyisini yazabilirsiniz. Farklı konfigürasyondaki makinelere uyarlayabilirsiniz.
  - Sonuç: Tek bir Linux yoktur, Linux'ın değişik parçalarından birleştiren dağıtımlar vardır.
  - Artıları: Her kişiye, her zevke, her duruma uygun bir dağıtım bulmak mümkündür.
  - Eksileri: Çok seçim olanağı sunulduğunda insanın ne istediğini anlayabilmesi için sabırlı olması gerekir!

Linux demek tam doğru değil. Linus Torvalds tarafından 1991'de yazılmaya başlanmış olan işletim sisteminin çekirdek kısmıdır. Tam bir sistem haline gelmesi GNU parçaları ile olmuştur.

Nerede Bu Programlar?

- Bu işletim sistemi, derleyiciler vs. oldukça büyük yer kaplayan programlardır. Onların tamamını ve daha birçok veriyi ve programı sürekli bellekte tutmak mümkün olmadığından, ayrıca hızlı bellekler çoğunlukla elektrik kesildiğinde silinen türden olduğundan sabit diskler kullanılır.
  - Bilgisayar açılırken silinmeyen bellekte (firmware) yer alan BIOS (yeni makinelerde UEFI) sabit disklerden (veya CDden, USBden, vs.) işletim sisteminin açılış ile ilgili temel parçalarını (boot record) okur ve belleğe yükler. Bu temel parçalar sistemin kalanının gerektiğince belleğe yüklenmesini sağlarlar.

BIOS = Basic Input/Output System
UEFI = Unified Extensible Firmware Interface

Sabit Disk Düzeni

- Sabit disklere her bir program veya veri parçası 0 ve 1'ler şeklinde yazılır. Ancak hangi veri parçasının nereden başlayıp nerede bittiğini tutan bir hesap-kitap lazımdır. Bunu işi yapmanın farklı farklı yolları vardır, bunlara **dosyalama sistemleri** (file system) denir.
  - Örnek: FAT, NTFS, ext3, HFS, vs.
- Sabit diskin belli bir dosyalama sistemi ile düzenli kullanılması işinden işletim sistemi sorumludur. Bazen bir işletim sistemi, başka bir işletim sisteminin kullandığı dosyalama sisteminden bihaber olabilir.
  - Kimi zaman belli dosyalama sistemlerinin nasıl çalıştığı tam açıklanmadığı için (kaynak kodu kapalı, yapılan işlerin ne olduğunun da yazılı net ifadesi yok), bir sistemden başkasına geçmek zor olur. Örneğin NTFS'i Windows dışında okumak için epeyce geri mühendislik gerekmiştir.

Dizinler

- Dosyalama sistemi her bir dosyanın diskteki yerini biliyor olabilir ama biz dosyaları kolay bulabilmek, aynı isimde değişik dosyaları karıştırmamak için onları dizinlere ayrırız.
  - Dizin prensipte işletim sistemi için çok bir anlam ifade etmez - asıl insanların aradığını bulması, yeni program yazarken parçaların derli toplu olması için faydalıdır.
- Grafik arayüzü sunan işletim sistemlerinde bilgisayar açıldıktan sonra gördüğümüz masaüstü de aslında sıradan bir dizindir. İşletim sistemi bu dizinde görülen her şeyi ekrana yansıtır.

Linux, Solaris, BSD gibi UNIX-tarzı işletim sistemlerinde bazen sabit disk üzerinde olmayan bazı veriler de özel bazı dizinlerdeki dosyalarmış gibi kullanıcıya sunulur (örneğin /proc dizini).

Kendi Programımızı Yazmak

- Bilgisayarımız var, içinde işletim sistemimiz ve derleyicimiz de var. Pekiyi kendi programımızı nasıl yazacağız?
  - İşi yapacağın bir dizin yarat. Dizini alakasız yerlerde yaratma, makul bir dizin düzenin olsun. (~/okullar/hpfbu14/birinciOdev/)
  - Yapılması gereken işi internetten ara. => Olmadı; arkadaşına sor.
  - İşe yaracak gibi görünen kodu ÖNCE ANLA, sonra kopyala ve yapıştır. Eksiklerini yavaş yavaş doldur. Kopyalamada yazılım lisanslarına dikkat et.
  - Derleyiciyi çalıştır, hata verirse, verdiği İLK hatayı dikkatlice **oku**. Hatayı anlamıyorsan, internetten ara. Bu adımı tüm hatalar giderilinceye kadar tekrar et. MORALİNİ BOZMA, SABIRLI OL!
  - Derlenmiş programı çalıştır. Beklenilen şekilde çalışmazsa, kaynak kodunun orasına burasına her adımda ne olduğunu takip etmeni sağlayacak çıktı komutları ekle. (cout << "Buradayim 001, a degiskeninin degeri = " << a << endl;)
  - Programları baştan bir seferde yazmaya çalışma! Küçücük bir parçasını yaz, yukardaki adımları uygula, sonra biraz daha uzat, tekrar yukarıdaki adımları uygula. Damlaya damlaya göl olur.

Yazılım Kütüphaneleri

- Aynı kodu tekrar tekrar yazmak ve tekrar tekrar derlemek büyük vakit ve enerji kaybıdır.
  - Bu yüzden belli işi yapan algoritmalar sınıf (class) veya fonksiyon haline getirilir. Bunlar kendi başlarına derlenirler. Böylece yazılım kütüphaneleri elde edilmiş olur.
  - Sınıfın veya fonksiyonun neleri girdi olarak aldığı ve neleri çıktı olarak verdiği, yani kısaca arayüzleri (interface) ayrı bir dosyada belirlenir.
  - Başka programlar yazılırken bu arayüz tanımları kullanılır. Bu programlar derlendikten sonra, daha önceden derlenmiş olan kütüphanelere bağlanır (linking).

Son Bir Tavsiye

- Kendinize güvenin.
  - Program yazmak özünde zor bir iş değildir. Ancak sabır gerektirir.
- Kendinize güven konusunda aşırılığa kaçmayın.
  - Gerçekten iyi programcılar öyle olduklarını söylemezler çünkü her zaman daha hızlı / daha az kaynak harcayan / daha kolay taşınabilir ve bakımı yapılabilir bir program yazılabileceğinin farkındadırlar.
  - Programlama dilleri de insan dilleri gibi sürekli değişim halindedirler. 90'ların C++'sı ile günümüz C++'sı arasında önemli farklar vardır.

YEDEKLER

Akşam Sefası

1. Çalıştırıldığında ekrana şu çeşit bir çıktı veren bir program yazınız:
   Karekok(5) = <karekok 5'in degeri>
   Bunun için C dilinin kendi sqrt() fonksiyonunu kullanınız. Kodunuzu derleyiniz, çalıştırıp deneyiniz.
2. Kendi karekök fonksiyonunuzu (işlevinizi) yazınız.
   float mysqrt(float x) { … }
   A. Karekök fonksiyonunun Taylor serisi açılımını çıkarınız.
   B. Açılımın ilk 5 terimi kullanarak kod yazınız.
   C. mysqrt(5) ile sqrt(5)'i karşılaştırınız. Cevap ne kadar yakın oldu?
3. √5 değerini Fibonnaci dizisini kullanarak tekrardan hesaplayınız.
   a. Fibonacci sayılarını üreten bir program yazınız. (Recursive yapabilirsiniz, ama hız sorunu olabilir. Düşünün, nasıl hızlandırabiliriz?)
   b. Birbirini takip eden iki Fibonacci sayısının birbirine oranı, limit sonsuza giderken altın oran denilen sayıyı verir: (1+√5)/2. Bu bilgiyi kullanarak √5'e yaklaşık sonuçlar hesaplayan bir program yazınız. Sonuçlarınızın gerçek √5'e ne kadar yakın olduğunu test ediniz.
4. Azıcık çılgınsanız: Bir Multi-precision hesap kütüphanesi bulup onunla 3. adımı tekrar yapınız.

Notlar ve Lisans

- Şekiller dışında bu sunumda yer alan her şey Creative Commons Attribution-ShareAlike 3.0 lisansıyla sunulmuştur. Tüm şekiller ya hali hazırda kamu malıdır, ya kendileri de CC lisanslıdır ya da adil kullanım kapsamındadır.
- Bu sunum HPFBU'12, Kars için hazırlanmış, HPFBU'14 için güncellenmiştir.