BİLGİSAYAR MÜHENDİSLİĞİNE GİRİŞ
⟨INTRODUCTION TO COMPUTER ENGINEERING⟩

İÇİNDEKİLER⟨CONTENTS⟩

BÖLÜM 1: HAZIRLIK KONULARI ⟨PREPARATORY SUBJECTS⟩

HK-0: İŞARETLER VE TERİMLER (BAŞVURU CETVELİ)
PS-0: ⟨SIGNS AND TERMS (REFERENCE TABLE)⟩

⟨ ⟩ ..... **açılı parantez**, bir İngilizce karşılığı belirtir.
⟨**angle parentheses**, specify an English counterpart.⟩
‘ ’ ..... **tırnak**, anlamsal alıntı veya anlamsal vurgu için.
⟨**quotes**, for semantic quotation or semantic emphasis⟩
“ ”..... **ikiz tırnak**, harfî alıntı için.
⟨**double quotes**, for literal quotation⟩
‹ › ..... **açılı tırnak**, özel gösterimler için.
⟨**angle quotes**, for special notations⟩
« »..... **açılı ikiz tırnak**, bir gösterimin okunuşunu belirtir.
⟨**double angle quotes**, specify how a notation is read⟩

Örnek⟨Example⟩: Bu gösterimde, ‘Türkçe’ ile ‘İngilizce’ arasındaki ayırt ediş kolaylığı ⟨The ease of differentiation between ‘Turkish’ and ‘English’ in this notation⟩:

10**«on⟨ten⟩»**	**“on”: Türkçe⟨Turkish⟩**
üstünde⟨on⟩	**“on”: İngilizce⟨English⟩**

‹V› ..... **«aslı Var ⟨true⟩»**
‹Y› ..... **«aslı Yok ⟨false⟩»**
~..... **«değil⟨not⟩»**, anlamı tam aksine çevirmek için kullanılır.
⟨used for negation, turning the meaning exactly to the opposite⟩
∧ ..... **«VE⟨AND⟩»** bir mantıksal işleç ⟨a logical operator⟩
∨ ..... **«VEYA⟨OR⟩»** bir mantıksal işleç ⟨a logical operator⟩
[ ] ..... **köşeli parantez**, bazen $[y_1, y_2]_S$ şeklinde, bir gerçek sayı aralığını belirtmekte kullanılır; bazen de parantezli bir ifadeyi tekrar paranteze almak gerektiğinde, kolay seçilebilen bir dış parantez çifti olarak kullanılır.
⟨**square parentheses**, sometimes used like $[y_1, y_2]_S$ in denoting a real number interval; at other times, used just as a pair of easily distinguishable outer

parentheses when a parenthesized expression has to be enclosed within parentheses.⟩

⊆ ..... **«altküme⟨subset⟩»**

⊂ ..... **«öz-altküme ⟨proper subset⟩»**

⊇ ..... **«üstküme⟨superset⟩»**, **«kapsar⟨encloses⟩»**

⊃ ..... **«öz-üstküme⟨proper superset⟩»**, **«fazlasıyla kapsar⟨properly encloses⟩»**

∪ ..... **«bileşim⟨union⟩»** ("**birleşim**" ile karıştırılmamalıdır ⟨not to be confused with "**combination**"⟩)

∩ ..... **«kesişim⟨intersection⟩»**

≡ ..... **«denk⟨equivalent⟩»**

{ } ..... süslü parantez, küme parantezi (küme belirtimlerinde çokça kullanıldığından) ⟨curly brackets, set parenthesis (for being widely used in set specifications)⟩

A_1 ..... **«*A* alta bir ⟨*A* sub one⟩»** altyazılı gösterim⟨subscripted notation⟩

A_{11}, A_{12}, a_{11}, a_{12} ... **«*A* alta bir bir virgül *A* alta bir iki virgül *a* alta bir bir virgül *a* alta bir iki ⟨*A* sub one one comma *A* sub one two comma *a* sub one one comma *a* sub one two⟩»** altyazı bazen aynı cinsten nesneleri indislendirmek ve/veya sıralandırmak için kullanılır. Bazen de, nesneleri indislendirmek veya sıralandırmak gibi bir amaç **olmaksızın**, sadece '**farklı simge**' ihtiyacını karşılamak için ve *A*, *B*, *C*, *x*, *y*, *z* gibi tamamen farklı simgelerin yerlerini tutmak üzere kullanılır (*x* ile *y* yerine $\boldsymbol{a_1}$ ile $\boldsymbol{a_2}$; x_1 ile x_2 yerine $\boldsymbol{a_{11}}$ ile $\boldsymbol{a_{12}}$ gibi). Altyazıların indislendirmek amaçlı olup olmadığı bağlamdan anlaşılır.
⟨Subscripts are sometimes used to index and/or enumerate objects of the same kind. At other times, used just to obtain different generic symbols to hold the places of totally different symbols like *A*, *B*, *C*, *x*, *y*, *z* (such as $\boldsymbol{a_1}$ and $\boldsymbol{a_2}$ instead of *x* and *y*; $\boldsymbol{a_{11}}$ and $\boldsymbol{a_{12}}$ instead of x_1 and x_2) whether the subscripts are used for indexing or not is understood from the context.⟩

$\mathbf{A_1 \times A_2}$ ..... **«A_1 çapraz A_2 ⟨A_1 cross A_2⟩»**

$\mathbf{A_1'}$ ..... **«A_1 üssü ⟨A_1 prime⟩»** $\mathbf{A_1}$ için tanımlıysa tümleyeni, değilse sadece nazire bir simge. ⟨The complement of $\mathbf{A_1}$, in case defined; otherwise just another paralleling variable.⟩

A_1 -den A_2 -ye ..... **"A_1'den A_2'ye"** ifadesi yerine. ('tırnak' ile 'üssü' karışmasın diye) ⟨(this item is specific to Turkish)⟩

∈ ..... **«ait⟨belongs to⟩»** bir öğenin bir kümeye aidiyetini ifade için kullanılır. ⟨used to denote that an element is a member of a set⟩

∉ ..... **«ait değil ⟨does not belong to⟩»** ~∈ anlamında ⟨meaning ~∈⟩.

$\mathbf{y_1}$**--** $\mathbf{y_2}$ ..... '$\mathbf{y_1}$ **eksi** $\mathbf{y_2}$' değil de '$\mathbf{y_1}$ **tire** $\mathbf{y_2}$' anlaşılsın istendiğinde "$\mathbf{y_1}$**–** $\mathbf{y_2}$" yerine. ⟨Instead of "$\mathbf{y_1}$**–** $\mathbf{y_2}$" where ever '$\mathbf{y_1}$ **dash** $\mathbf{y_2}$' but not '$\mathbf{y_1}$ **minus** $\mathbf{y_2}$' is meant⟩

| ..... **«cirit⟨vertical bar⟩»**

$|x|$ ..... **«cirit arası x ⟨x enclosed in vertical bars⟩»** *x* -in niceliği ⟨quantity of *x*⟩

E&A ..... **«Eğer ve Ancak»**

Æ ..... **E&A «Eğer ve Ancak»**

iff ..... **«⟨if and only if⟩»**

☺ ..... **«İspat tamam ⟨Proof is completed⟩!»**

⛌..... **«Çelişki⟨Contradiction⟩!»**
mod..... **«modulo»**
⊕..... mantıkta zıtlık işleci, aritmetikte mod-2 toplam işleci
⟨exclusive-or operator in logic, mod-2 addition operator in arithmetic⟩
→..... mantıkta imâ (gerektirim) işleci, matematikte nişanlayış işareti.
⟨implication (entailment) operator in logic, mapping sign in mathematics⟩
≥..... **«en az ⟨at least⟩»** (a≥ b **«**a en az b -dir. ⟨a is at least b⟩**»**)
≤..... **«en çok ⟨at most⟩»** (a≤ b **«**a en çok b -dir. ⟨a is at most b⟩**»**)
∴ ..... **«demek ki ⟨therefore⟩»**, **«öyleyse⟨if so⟩»**, **«binaenaleyh⟨thereupon⟩»**
∃..... **«vardır⟨there exists⟩»** Varsal Niceleyen ⟨Existential Quantifier⟩.
∃**x**..... **«vardır x öyle ki... ⟨there exists x such that...⟩»**
∀ ..... **«hepsi⟨all⟩»** Evrensel Niceleyen ⟨Universal Quantifier⟩.
∀**x** ..... **«her x için ⟨for each x⟩»**, **«bütün x -ler için ⟨for all x⟩»**.
ϕ..... **boş küme ⟨empty set⟩**
ε ..... **boş dizgi ⟨empty string⟩**
ƀ..... «**boşluk⟨blank⟩**» **boş simge ⟨empty symbol⟩**
..... «**aralık⟨space⟩**» **boş şekilcik ⟨space character⟩ (kendisi⟨itself⟩)**
␣..... «**aralık⟨space⟩**» **boş şekilcik ⟨space character⟩ (görülebilir temsili ⟨visible representation⟩)**
0..... «**sıfır⟨zero⟩**»**, boş rakam ⟨empty digit⟩**
☽..... **«sağ-hilal ⟨right-crescent⟩» dipnot işareti ⟨footnote sign⟩**
☾ ..... **«sol hilal ⟨left-crescent⟩» işlem işareti ⟨operation sign⟩**

Bazı Harfleri Simgeler İçin Kullanılan Yunan Alfabesi:
⟨The Greek Alphabet, Some Letters of Which Are Used For Symbols⟩:

Bilgisayar Bilimlerinde ⟨In Computer Science⟩		**Elektronikte, Fizikte ⟨In Electronics, Physics⟩**		**Geriye Kalanlar ⟨The Remaining Ones⟩**	
Α **α**	**«Alfa⟨Alpha⟩»**	**Ω** ω	**«Omega»**	Ι ι	**«**Ayota⟨Iota⟩**»**
Β **β**	**«Beta»**	Τ **τ**	**«Tau»**	Κ κ	**«**Kepa⟨Kappa⟩**»**
Γ γ	**«Gama⟨Gamma⟩»**	Θ **θ**	**«Teta⟨Theta⟩»**	Ν ν	**«**Nu**»**
Δ δ	**«Delta»**	Ψ **ψ**	**«Say⟨Psi⟩»**	Ο ο	**«**Omikron⟨Omicron⟩**»**
Φ **ϕ** φ	**«Fi⟨Phi⟩»**	Ρ **ρ**	**«Ro⟨Rho⟩»**	Υ υ	**«**Yupsilon⟨Upsilon⟩**»**
Π **π**	**«Pi»**	Η **η**	**«Eta»**	Χ χ	**«**Çi⟨Chi⟩**»**
Σ **σ**	**«Sigma»**	Μ **μ**	**«Mü⟨Mu⟩»**	Ξ ξ	**«**Zay⟨Xi⟩**»**
Ε **ε**	**«Epsilon»**	Ζ **ζ**	**«Zeta»**		
Λ **λ**	**«Lambda»**				
⟨The ones whose reading in English are not shown separately are read just as shown.⟩					

⋹-⋽

HK-1: BAZI GENEL HUSUSLAR
PS-1: ⟨SOME GENERAL ISSUES⟩

ÖN UYARI: Bu girişte yaş tahtalar var; çok dikkatli ilerleyiniz. Bilgisayar Biliminde zaten yaş tahta çok. Madem girdiniz bu yola, şimdiden alışırsanız dikkatli yürümeğe, rahat edersiniz sonra.

Bundan böyle düşünerek atın adımlarınızı, Elbet bir gün mutluluktan yana alırız payımızı...	**Ali KOCATEPE**

Ele aldığımız ne olursa olsun, özelliklerine dikkat etmek gerekir. Dikkat gerektiren özelliklerin en başında **cins⟨type⟩** gelmektedir. Bir elma bir armut daha iki meyve eder.

1 elma + 1 armut= 2 meyve

Bir kuş gibi hür olsam, dey dey, dibi dibi dey dey Sorulmadan yaşasam, hey gidi dünya hey	**Ali KOCATEPE** (devamı)

Sorun: Ali 18 yaşında 1.78 boyunda yakışıklı bir delikanlı. Hasan 1.60 boyunda onun ağabeyi, bir kaza geçirdiği için yüzünde yara izleri var. Ali ile Hasan'ı karşılaştırıyoruz.....(Satır Y1-1)

(a) Hangisi önce? **Cevap**: Ali (abece sırasında)

(b) Hangisi uzun? **Cevap**: Hasan (5 harfli)

(c) Hangisi güzel? **Cevap**: Hasan (Türk Dil Kurumu Türkçe Sözlüğü öyle diyor.)

Siz belki şöyle cevaplardınız:

(a) Hangisi önce? **Cevap**: Hasan (dünyaya geliş sırasında)

(b) Hangisi uzun? **Cevap**: Ali (1.78 boyunda)

(c) Hangisi güzel? **Cevap**: Ali (yakışıklı, yüzünde yara izi yok.)

Çıkarılan Ders:
Farklar, "Ali" ile "Hasan"ın farklı yorumlanışından kaynaklanmaktadır. Kastedilen isimler midir, şahıslar mıdır? Muhatap algısının murada uygunluğu, farklı yorumlara mahal bırakmayan **yeterli güçte** bir anlatımı ve böyle bir anlatım için mümkün olduğunca dikkatle geliştirilmiş bir mutabakat zeminini gerektirir. Böyle bir zemine "**resmiyet⟨formalism⟩**", böyle bir resmiyeti yaşatmak için **dilimizin özenle seçtiğimiz belli bir kısmına** "**resmi ifade dili ⟨formal expression language⟩**" diyoruz.

Örneğin resmi dairelerde, üste hitap ederken "arz ederim", asta hitap ederken "rica ederim" denişi, resmi dairelerde kullanılan resmi ifade dilinin kurallarındandır.

Kullanılan resmi ifade dilinin **yeterince zengin** oluşu aranan bir özelliktir. Bu itibarla, öncelikli çabamız, kendimiz için bir resmi ifade dili geliştirmektir.

Dostluk benim bayrağım, dey dey, dibi dibi dey dey Cennet olsun durağım, hey gidi dünya hey !..	**Ali KOCATEPE** (devamı)

-Bilgisayar bilimi bağlamında çok daha genel olan bir "**resmi dil ⟨formal language⟩**" kavramı vardır ve buradaki "**resmi ifade dili**" kavramıyla karıştırılmamalıdır.-

Yukarıda "Ali ile Hasan" derken kastedilen isimler miydi, şahıslar mıydı?

Bundan böyle düşünerek atın adımlarınızı, Elbet bir gün mutluluktan yana alırız payımızı...

Ali KOCATEPE
(devamı)

Bu gibi hususlarda daha fazla netlik sağlamak için **tırnak işaretleri**nden yararlanıyoruz:

TIRNAK KURALLARIMIZ:

TIRNAK İŞARETİ	İFADE	İŞLEV	KASTEDİLEN
-tırnak yok-	Ali ile Hasan	aynen belirtim	şahıslar
tırnak	'Ali' ile 'Hasan'	anlam belirtimi	isimlerin ifade ettiği anlamlar
ikiz tırnak	"Ali" ile "Hasan"	harfiyen belirtim	isimler

Ör. "Hasan"≠ "güzel" fakat 'Hasan'= 'güzel'

Buna göre yukarıdaki soruların cevapları:

KIYASLANANLAR	Hangisi önce?	Hangisi uzun?	Hangisi güzel?
Ali ile Hasan	Hasan	Ali	Ali
'Ali' ile 'Hasan'	**-tanımsız-**	**-tanımsız-**	'Hasan'
"Ali" ile "Hasan"	"Ali"	"Hasan"	**-tanımsız-**

Burada "-tanımsız-" ifadesi, verilenlerin ilgili hüküm için yeterli bir ölçü teşkil etmediği anlamındadır. Örneğin "Ali" mi yoksa "Hasan" mı daha güzel bir isimdir? Sorusuna isabetli bir cevap vermek için burada bir nesnel ölçü bulunmamaktadır.

İkiz tırnakla yapılan 'harfiyen belirtim'e alıntı diyoruz.
(Kayıt: **"'harfiyen belirtim'"in kendisi bir 'harfiyen belirtim' değildir.)**

Alıntı Örnekleri:	**Yeri**	**Yorum**
Yukarıda **"Ali ile Hasan"** geçmektedir.	(Satır Y1-1)	(alıntı)
Yukarıda **""""Ali ile Hasan""""** geçmemektedir.	-Yukarıda yok-	(bu nokta için doğru)
Yukarıda **""Ali ile Hasan""** geçmektedir.	İki önceki satır	(alıntının alıntısı)
Yukarıda **""""Ali ile Hasan""""** geçmektedir.	Bir önceki satır	(alıntının alıntısının alıntısı)

Örnek: Yukarıda geçen... "**Yukarıda "Ali ile Hasan" derken kastedilen**" derkenki **""Ali ile Hasan"" «Ali ile Hasan** alıntısının alıntısı», metnin daha önceki kısımlarından bir alıntıdır.

Soru: Bu noktaya kadar açıklanan kurallardan anladığımız kadarıyla "**xyz**" görürsek isim olarak mı, yoksa **alıntı** olarak mı değerlendireceğiz?
Cevap (örnek üzerinden):
Şahıslar olarak Ali ile Hasan'dan bahsederken isimlerini kullandık.
"Ali ile Hasan" dedik. (Burada alıntı var!)

"Ali ile Hasan" derken içinde "Ali" var. ("Ali" isim olmakla beraber, ismin alıntısı)

("Ali ile Hasan" derken içinde "Ali" olduğu gibi,
"Ali ile Hasan", "ile", "Ali ile", "ile Hasan" ve "Hasan" da var.)

"Ali ile Hasan" alıntısından nasıl bahsediyoruz? İsmiyle! Nedir o isim? "Ali ile Hasan"!

(Dikkat, "Ali ile Hasan", "Ali" ile "Hasan" gibi iki isim değil,
tırnakla sarılı bir bütün! Bir şahıs ismi de değil, bir tek isim, o da alıntının ismi!)

"Ali" hem şahıs ismi, hem de o ismin alıntısının ismi!

Özetle görülüyor ki, isim aynı zamanda ismin alıntısı ve alıntı aynı zamanda alıntının ismi. Yani ikisi arasında fark yok! Sonuç olarak "**xyz**" gördüğümüzde **isim** ile **alıntı**dan hangisi olarak değerlendireceğimiz keyfi bir tercih meselesi.

Kullandığımız "Ali", bahsettiğimiz Ali.

Doğruluktan hiç şaşma, dey dey, dibi dibi dey dey
Eden bulur, üzülme, hey gidi dünya hey

Ali KOCATEPE
(devamı)

Harfî Alıntı vasıtasıyla Kavramsal Atıf ⟨Conceptual Reference⟩:

Metnin bazı yerlerinde, önceki kısımda verilen başvuru cetvelinin "**bileşim**" satırında görülen

(**"birleşim"** ile karıştırılmamalıdır ⟨not to be confused with **"combination"**⟩)

gibi, harfî alıntı kullanımı vardır. Kastedilen,

(**"birleşim"** *denen kavram* ile karıştırılmamalıdır
⟨not to be confused with *the concept called* **"combination"**⟩)

şeklindedir. Dolayısıyla, her ne kadar **"birleşim"** bir harfî alıntı ise de, burada **""birleşim"** *denen kavram***"** şeklindeki kavramsal atıfı kasteden bir **"kısaltım"** olarak kullanılmaktadır.

(**Kayıt**: Son cümlede **""kısaltım"** olarak**"** derken yine bir harfî alıntı ile kavramsal atıf örneği ortaya çıkmaktadır ve **""kısaltım"** denen bir şekilde**"** ifadesinin anlamı kastedilmektedir.)

Geçmiş muhabbetlerden:

Sayın Hocam, Ben ..., dersinizi alan öğrencilerinizdenim. Dersinizin notlarını ve videoları çalışmama ve düzenli takip etmeme rağmen hem konularda hem de verdiğiniz ödevleri yapmakta çok zorlanıyorum.
Cevap: Zorlanıyorum demeniz, yeterli olabilecek asgari gayretin daha azını yeterli zannetmenizdendir.
Sayın hocam, dersinizi daha iyi öğrenebilmem için bana birkaç kaynak kitap veya site veya ders notu önerebilir misiniz?
Cevap: Üniversite düzeyinde bulunduğumuza göre, bunları araştırmak sizin vazifenizdir.
Ya da bir öneride bulunabilir misiniz? Bu konuda yardımlarınızı bekliyorum. İlginiz için şimdiden teşekkürler. Saygılarımla...
Cevap: Matematiksel resmiyet içindeki anlatımları bir okuyuşta anlayamamak gayet doğaldır. Dolayısıyla, işin yolu şudur: Öncelikle, evvelki konularda anlamadığınız yer kalmadığına emin olunuz.

-Halkası noksan bir zincir işe yaramaz.- [☽]

"Okuyunuz, okurken yazıp çizerek, kendinizce örnekler oluşturunuz. Anlamadıysanız, bu tırnak içinde isteneni daha fazla bir dikkatle, tekrarlayınız."

Sonunda ya anlatılanı anlayacaksınız, ya da neden anlamadığınızı. İkinci hâlde, notlarda bir hata veya noksanlık var demektir. Bana ulaşınız.

Başarı dileklerimle...

Hiç acısı olmayan, dey dey, dibi dibi dey dey
Kim var ki bu dünyada, hey gidi dünya hey !..

Ali KOCATEPE

(sonu)

[☽] Sonraları birisi çıkıp dedi: "**Halkası noksan bir zincir**, daha kısa bir zincir olarak işe yarayabilir." Cevapladım: "**Evet, fakat o daha kısa zincirde noksan halka yoksa!**"

HK-2: ANLAMAK HAKKINDA
PS-2: ⟨ABOUT UNDERSTANDING⟩

"**Öğrenmeyi Öğrenme!**" derler. Biz ise, "**ÖĞRENİŞİ ÖĞRENİŞ!**" deriz. Olumlu bir işe olumsuz bir temenni ile başlamak niye?

Haydi, "**Öğrenişi Öğreniş**" diye başlamış olsunlar, "**ANLAYIŞI ANLAYIŞ**" üzerinde pek durmazlar. Seneler boyu tecrübeler göstermektedir ki, öğrenci kitlesinde '**anlamak**' konusunda ciddi sıkıntılar vardır. Bu sıkıntıların en başında, anlayış sıkıntısına karşı '**hissizleşmek**', bir nevi '**alışkanlık**' kazanmak ve '**anlamak**' zorunluluğuna '**fazla kafayı takmadan**' öylesine bir yaşayış içine girmek vardır. Sonuçta kişi, '**anlayıp anlamadığını anlayamaz**' bir hale gelebilir ki, başarı arayan için bu bir '**çıkmaz sokak**'tır.

Anlamamaktan daha kötü, anlamamağa başladığı zaman farkına varamamak. Yani anlamamaya başladığında, anlamadığını da anlayamıyor olmak.

Bazen derste soruyoruz: "**Anlamayan var mı?**"... Birkaç tane el kalkıyor.
İkinci defa şöyle soruyoruz: "**Anlayan var mı?**"... Yine birkaç tane el kalkıyor.
Geriye kalanlar, '**anlayıp anlamadığını anlayamayanlar**'

Anlamak bir süreçtir: **Anlamadım; azıcık anladım; daha iyi anladım; daha iyi anladım; ...;** sanırım, **daha iyi anlamak** adımlarında **yeterli bir seviyeye geldim**!
Bu '**yeterli**' hissettiğimiz seviyeye ulaşınca kısaca "**anladım**" ifadesini kullanıyoruz; fakat ortada bir süreç söz konusu olduğundan dolayı, bu "**anladım**" seviyesini bir süre sonra yeni bir "**şimdi daha iyi anladım**" seviyesinin izlemeyeceğini kim iddia edebilir.

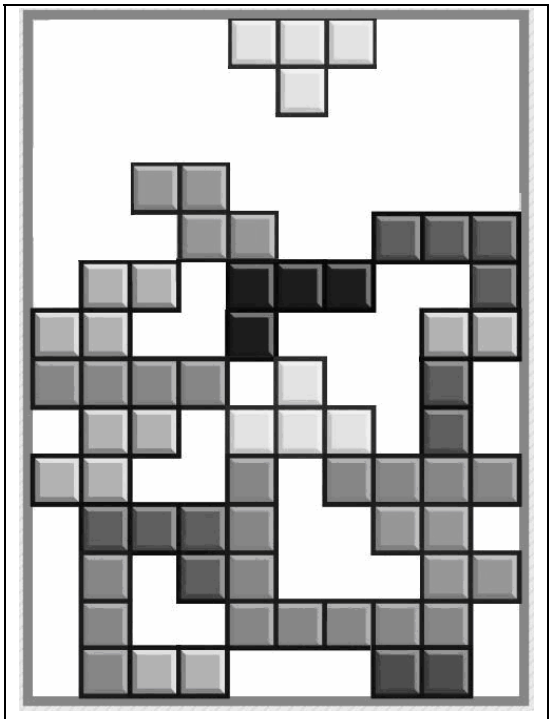
'ANLAMAK', Bütün Olarak Nedir? Bu soruyu tam olarak cevaplamak kolay değildir. Belki de bu sebepten dolayı, "**Öğrenmeyi Öğrenme**"ciler, "**Anlayışı Anlayış**" konusuna fazla yanaşmamaktadırlar. "**Öğrenmeyi Öğrenme**"cilerin öğretilerine dikkatle bakarsak, '**öğreniş**' eyleminin daha ziyade, 'kavramları hafızaya, kolay çağırılabilir (hatırlanabilir) bir şekilde doldurmak' olarak ele alındığını görürüz. Yani, '**beynin hafıza yeteneğinden nasıl daha çok istifade edebiliriz**' konusu üzerinde dururlar. Bunun için hatırlayışı kolaylaştırmak üzere kavramlara **çağrışım tutamakları iliştirmek** ve bir kavramın çağrışım tutamağını diğer bir kavrama tutturmak tekniğini kullanırlar. Böyle bir teknikle ilk kavramdan başlandığında arkadan gelen kavramlar, zincirin halkaları gibi sırayla şuur merkezine getirilebilir.

Amaç ezber olduğunda, bahsedilen teknik faydalıdır. Ne var ki, beynin yetenekleri hafıza ile beraber muhakemeyi de kapsamaktadır. Muhakeme, zihindeki kavramlar arasında, **yeni tutarlı kavramlar doğurabilecek** potansiyelde münasebetler kurmak eylemini kapsar.

Yani zihinde sadece, kolay hatırlamak için kavramlara **çağrışım iliştirmek** değil, **yeni tutarlı kavramlar üretebilmek** için kavramlar arası münasebetleri düzenlemek ve güçlendirmek de gereklidir.

Sıklıkla rastlayabildiğimiz sıradan öğrenci modeli:

Derste verilenleri zihninde yerleştirmeğe çalışır, tam oturmayanları zihninin bir köşesine yığar. "**Bunları daha sonra zihnimde yerleştiririm**." Der. Tam oturmayanlar zihninin bir köşesine yığılmağa başlayınca, arkadan gelenler için tam oturmak şansı gittikçe azalır. Evine gittiğinde '**derslerini çalışmak**' çerçevesinde, zihninin köşesine yığdıklarını, tuttuğu kayıtlardan ve kitap satırlarından da yararlanarak, zihnindeki raflara yerleştirmeğe çabalar. Bu arada muhakeme açısından gerekli münasebetleri düzenlemek ve güçlendirmek hususlarında da gayret sarf eder. Ne var ki, yığdıkları geçen zaman içinde tazeliğini kaybettiği için, bazılarının hangi rafa yerleşeceğini tayinde zorluk çeker. Sonuçta, yığdıklarının bir kısmı, üzerlerine yenilerinin yığılışına mahkûm olarak raf dışı, yani anlaşılmamış olarak kalır.



Tam oturmayanlar zihninin bir köşesine yığılmağa başlayınca, arkadan gelenler için tam oturmak şansı gittikçe azalır.

Böyle bir öğrenci, konuyu esasından kavrayamadığı için, örneklerden medet umar. Muhakemesini geliştirmediği için ufku, gördüğü örnekler kadardır. Yeteneği, örnekleri taklit düzeyinde kalır. Yanlış yöntem izlediği için başarıdan kaybeder, çabalarının tam karşılığını alamaz.

Başarıya aday öğrenci modeli:

Dersi dikkatle izler ve her aldığını sıcağı sıcağına, zihnindeki raflarda mevcut olanlara göre münasebetinin gerektirdiği yerini bularak yerleştirir. Eğer herhangi bir anda aldığı elinde kalır da zihnindeki hangi rafa nasıl yerleştireceğini bilemezse, "şimdilik bir kenarda dursun, sonra yerini bulur yerleştiririm" **demez**! Demek ki son verileni **ANLAMAMIŞTIR**. Bu durumu **fark eder** ve zaman kaybetmeksizin bu son verilenin zihnindeki yerini bulup yerleştirebilmek için yardım ister. Yani, **SORU SORAR**.

Böyle bir öğrenci, konuyu esasından kavradığı için, duruma göre bazen kendi kendine ortaya örnek koyabilir. Dersi derste öğrendiği için zamandan kazanır.

Soru sormak konusunda bazı küçük kurallar:

- Soru sormaktan çekinmemelidir.
- Soru sormakta biraz sabır olabilir fakat aşırı gecikilmemelidir.
- Soru, dersleri düzenli olarak izleyen, tam devamlılık gösteren bir öğrencinin doğal hakkıdır.
- Düzenli izleyiş ve tam devamlılık şartlarını sağlayan bir öğrenci, sadece kendisinin bilmediğini düşündüğü bir soruda dahi tereddüt etmeden ders içinde sorusunu sormalıdır.

- Ders kaçırmak kusurdur. Bu durumdaki öğrenci, arkadaşlarından gerekli telafiyi yapmadan, kaçırdığı konulara dayalı soru sorduğu takdirde, hakkını aşmış olacağını değerlendirmelidir.

Yukarıdaki hususlara dikkat ederek, hepimiz başarıya aday öğrenciler olalım.

HK-3: **DİL⟨LANGUAGE⟩ ve ÖTEDİL⟨METALANGUAGE⟩ HAKKINDA**
PS-3: ⟨ABOUT LANGUAGE AND METALANGUAGE⟩

Bilindiği üzere, **Türkçe**, **İngilizce**, **Arapça**, **Çince** gibi **diller** vardır. Bu dillerin her birisi, anlatımlarda kullanılan birer **ifade sistemi**dir.

Bazen anlatım konusu, bizzat bu ifade sistemlerinden birinin kendisi, örneğin **İngilizce**, olur. Böyle olduğunda, anlatım için yine kendisi kullanılabilir mi? Yani örneğin **İngilizce** anlamayan birisine tamamen **İngilizce** içinde kalarak **İngilizce** -yi anlatmak mümkün müdür? Tabiidir ki, **İngilizce** -yi anlamayan, onun **İngilizce** olarak anlatımını da anlamaz. Görülüyor ki, bilinmeyen bir dili anlatmak, o bilinmeyen dilin içinde mümkün değildir; bilinen, başka bir dilden yararlanmak zorunludur.

Anlatım konusu dile "**Konu-Dil⟨Object-Language⟩**", onu anlatmakta kullanılan dile de (konu-dile izafeten) "**Ötedil⟨Metalanguage⟩**" denir. Konu-dile değil de öte-dile ait olan öğeler veya simgeler, "**Ötedil-Öğesi**", **Öte-Öğe** veya **Öte-Simge** denerek ayırt edilir. Örneğin **İngilizce** -ye başlarken **Türkçe** -den yararlanıldığı takdirde, anlatım konusunu teşkil eden konu-dil **İngilizce**, anlatımda kullanılan ötedil ise **Türkçe** olmuş olur. Bir çocuğun anadilini öğrenişinde yararlanılan ötedil, çevredeki kişilerin beden dillerini ve ayrıntıların görünümlerini kapsayan, "**hâl dili**"dir.

Bazen matematiğin bir alt sistemi matematiksel bir dil ile anlatılırken kullanılan bir öğenin, anlatılan alt sisteme mi, yoksa anlatımda kullanılan matematiksel dile, yani ötedile mi ait olduğu açıkça belirtilmeyebilir. Böyle durumlarda, bağlama dikkat etmek gerekir.

Meselâ aritmetikteki "x+ 1= 5" gibi bir ifadede x, değeri sayı cinsinden, ismi "x" olan, belli bir **aritmetiksel değişkendir**. Hâlbuki şöyle desek:

"Herhangi bir aritmetiksel değişken x için, x+ 1= 1+ x olur."

Böyle bir ifadede x bizzat bir aritmetiksel değişken değil, herhangi bir aritmetiksel değişkeni temsil eden bir öte-öğe, öte-değişkendir. Taşıdığı değerin cinsi sayı değil, '**aritmetiksel değişken**'dir. Bu ifadedeki x kimi zaman y, kimi zaman z, vs. gibi bir aritmetiksel değişken olabilir. Yani bu ifade bir denklemi değil, ne kadar aritmetiksel değişken varsa, her birisiyle oluşmuş denklemlerin hepsini, bir "**tarz⟨schema⟩**" veya "**silsile⟨schemata⟩**" olarak ortaya koymaktadır.

HK-4: TÜRKÇE DİLİ HAKKINDA
PS-4: ⟨ABOUT TURKISH LANGUAGE⟩

Yukarıda ifade sistemi dediğimiz dile, **anlam aktarım ortamı** da diyebiliriz. Günlük anlatım ihtiyaçlarımızda anadilimiz Türkçeye sarılırız. Aynı şekilde, bilimsel ve teknik anlatım ihtiyaçlarımızda da anadilimiz Türkçeden vazgeçemeyiz.

Dil Kuramı Açısından Türkçe:

⟨Turkish From the Language Theory Point of View⟩:

Bilgisayar biliminin önemli alanlarından birisi olan dil kuramında diller, "**doğal diller ⟨natural languages⟩**" ve "**resmi diller ⟨formal languages⟩**" olarak iki temel sınıfa ayrılır. Doğal diller, toplumlarla doğar, gelişir, bazen de Sümerce gibi, konuşanı kalmayarak ölür. Dilbilgisi kurallarına sahip olsalar da, hudutlarını çizen kesin tanımları yoktur. Diğer taraftan resmi diller tanıma bağlıdır. Bilgisayarlarda kullanılan diller resmi dil örneklerinden olduğu gibi, askerî eğitimde öğretilen muhtelif çevrim içi hitap kuralları da resmi dil tanımı örneklerindendir. Kullanılmakta olan resmi dillerde gözetilen özelliklerden bir tanesi, **muğlâklık⟨ambiguity⟩** bulunmayıştır. Her ifadenin tamı tamına bir anlamı vardır. Birden fazla anlama sahip ifadeler karışıklık kaynağı olacağından, ne bilgisayarlarda, ne askeriyede, ne de diğer bilimsel ve teknik konularda istenir.

Doğal diller resmi tanıma dayanmadıklarından, elde olmayan muğlâklıklara rastlanmaktadır. Bu muğlâklıkları mümkün olduğunca önlemek lâzımdır. Türkçe için bu gibi konularda görev Türk Dil Kurumuna düşmektedir. Biraz durup geriye doğru bakılırsa, geçmişte Türkçe üzerinde ne gibi çabalar sarf edildiği gözlenebilir. Bu çabaların kimisi olumlu, kimisi olumsuz olmuştur.

Dilimizi, dediğimizi yeterince düşünüyor muyuz? Ne demek istiyoruz da, ne diyoruz?
Diyeceğiz : Muhtemel saldırıya mukavemet Diyoruz : Olası saldırıya karşı koyma Kastımız : **Olmayası** saldırıya **karşı koy**
İngilizce - Operations are, sort, search, compare, Yanlışça - İşlemler: sıralama, arama, karşılaştırma Kastımız - İşlemler: sırala, ara, karşılaştır

Biz burada kısaca şu görüşlerimizi ortaya koyarak, buradaki metinde kullanılan Türkçede dikkat edilen hususları tanıtmak istiyoruz:

Türkçe diline dair metinde benimsenen ilkeler:

1- İngilizcede '**⟨definite article⟩**' "**the**" ve '**⟨indefinite article⟩**' "**a**" vardır. Özel isimler hariç, hemen hemen her ismin başında bunlardan birisi olur ve o ismin belirli bir şeye mi, yoksa belirsiz

bir şeye mi ait olduğunu ifade eder. Paralel ifadeleri Türkçeye yansıtmak için "**bu**" ve "**bir**" kelimeleri sıklıkla kullanıldı.

2- Bazen bilgisayarda bir seçim kutusu konup, yanına "ortalama", "yazdırma", "gölgelendirme" gibi kelimelerle biten ifadeler konuyor. Böyle ifadeler Türkçede hem olumlu hem de olumsuz anlama gelebiliyor dolayısıyla böyle bir kutu seçilirse sonucun ne olacağı hususunda tereddüt oluşuyor. Türkçenin bu zafiyetiyle mücadele için, metinde mümkün olduğunca, -me, -ma ile biten olumlu eylem kelimelerinden kaçınıldı. Bu doğrultuda, örneğin, "açıklama" değil de "açıklayış" dendi; "alıştırma" değil de "alıştırım" dendi.

3- Kelime seçimlerinde serbest davranıldı. Yakın zamanda türetilmiş kelimelerin yanı sıra evvelce kullanılanlar veya batıdan gelenler, dile nasıl doğal gelirse öyle kullanıldı. Sonuç itibariyle gelen kuşaklara evvelce kullanılan kelimeleri kaybettirmekte değil, kazandırmakta yarar olduğu değerlendirildi. Aksi halde gelen kuşakların, **Atatürk'ün** öz üslubunun güzelliğinden haberdar olamayacakları, İstiklâl Marşı yazarı milli şairimizin mısralarının lezzetinden mahrum kalacakları düşünüldü.

BÖLÜM 2: BİLGİSAYARLAR HAKKINDA:

Bilgisayarların, amaç yönünden, hesaplama esası yönünden ve donanım teknolojisi yönünden cinsleri vardır. Yaygın olan ve "bilgisayar" dendiğinde anlaşılan, genel amaçlı sayısal elektronik bilgisayarlardır. Bunlar bir taraftan fiziksel, bir taraftan da yeteneksel olarak boy boydur. Bir kullanıcılı olanları bulunduğu gibi, aynı anda çok sayıda kullanıcıya hizmet edenleri de vardır.

Bilgisayara, bir tür **akıl gücü yükselteci ⟨brain power amplifier⟩** diyebiliriz. Elektronik ses yükselteçleri nasıl sesin güç seviyesini yükseltiyorlarsa, bilgisayarlar da insanların akıl güçlerini yükseltirler, bir insan aklının yapabileceği bazı işleri çok daha büyük kapasitede ve hızda yapabilirler.

1. *Hesaplayıcı ile İzlence ⟨Computer and Program⟩:*

Bilgisayara "**Hesaplayıcı**" da diyebiliriz. Dikkat edersek, "**Hesaplayıcı**" dediğimiz zamanki ifademizde '**hesap** yapan' ve dolayısıyla '**hesap** yeteneğine sahip olan' anlamları yatmaktadır. Gerçekten de bir hesaplayıcıyı (bilgisayarı), 'hesap makinası' olarak bilinen daha basit düzeneklerden ayıran en önemli özellik, ona yapacağı hesabın **tarif**inin verilebilişi ve onun da verilen tarifi harfi harfine uygulayabilişidir. Bir hesaplayıcıya verilebilecek türdeki bir '**tarif**'e "**bilgisayar programı**", "**bilgisayar izlencesi**", "**hesaplayıcı izlencesi**" veya kısaca "**izlence**" diyoruz.

İlerideki konulara geçmeden önce bilgisayarlarla ilgili bazı tanımları ortaya koymakta fayda vardır.

1.1. Tanımlar :

İzlence⟨Program⟩: Her hangi bir yöntemin adım adım nasıl uygulanacağının, tam anlaşılan ve birden fazla yoruma meydan vermeyen bir dil ile anlatımı.
Bilgisayar⟨Computer⟩: Hafızası olan, o hafızaya uygun bir biçimde izlenceler yerleştirilebilen, kendi kendine o izlenceleri takip ederek, tarif edilen işlemleri yerine getirebilen bir cihaz.
İzlenceleyiş⟨Programing⟩: İzlence gerçekleştiriş.
İzlenceleyiş Dili ⟨Programing Language⟩: İzlenceleyiş için ortaya konmuş bir dil. (PASCAL, C^{++}, FORTRAN, BASIC, Makine Dili, Simgesel Dil gibi.)
Yazılım⟨Software⟩: Belli bir bilgisayar için üretilmiş, belli bir amaç, işlev ve ticari kıymeti haiz olan izlence veya izlenceler manzumesi.
Donanım⟨Hardware⟩: (Bir bilgisayardaki) fiziki kısımlar.
Bilgi-İşlem Düzeni ⟨Information-Processing System⟩: İşleyişi için gerekli tüm çevre birimleri ve yazılımı üstünde hazır bulunan bir bilgisayar veya bilgisayarlar dizisi.
Veri⟨Data⟩: Her hangi bir hesaplayış işinde girdi olarak kullanılan veya arada üretilen veya sonuçta elde edilen bilgi.
Bit⟨Bit⟩: 0 veya 1 şeklinde, yalnız iki durumdan birini ifade edebilen (en küçük) bilgi birimi. Bazen bir bitlik bilgi yerine de kısaca "**bir bit**" denir. İkilik sayılarda "**hane**" olarak kullanılır.

Bayt⟨Byte⟩: 8 bit.
Kelime⟨Word⟩: İki bayt. (Bazen dört bayt veya daha farklı olarak da tanımlanabilir.)
Kayıtlık⟨Register⟩: Belli sayıda bitten oluşan bir bilgi parçasının yazılıp okunabildiği sabit sığalı fiziksel ortam. Sığası n-bit olan bir kayıtlığa n-bit uzunluğunda kayıtlık veya n-bitlik kayıtlık denir.
Kayıtlık Hanesi ⟨Register's Digit⟩: Kayıtlığın bir bitlik bir yeri.
Hafıza⟨Memory⟩: Aynı nitelikte ve sığada kayıtlıklardan oluşan bir kayıtlıklar dizisi.
Hafıza Yeri (Yer⟨Location⟩): Hafızayı oluşturan kayıtlıklardan herhangi biri.
Adres⟨Address⟩: Hafızanın her hangi bir yerini belirtmek için kullanılan sayı. Genellikle adresler 0,1,2,... şeklinde gider. Böyle bir sayıyı ifade için kullanılan simgelere de adres denir.
Değişken⟨Variable⟩: Bir **değerin** simgesel belirtimi.
İşaretçi⟨Pointer⟩: Adres. Tuttuğu değerin cinsi işaretçi olan bir değişken veya kayıtlığa da işaretçi denebilir.
Yafta⟨Label⟩ (Fransızca:"**etiquette**"): Bir **adresin** simgesel belirtimi.
Değişken İsmi ⟨Identifier(Variable Name)⟩: Ait olduğu değişkenin değerinin tutulduğu yerin yaftası.
Ör. Adresi 0005 olan hafıza yeri "Mesafe" isimli bir değişkenin değerini tutuyor olsun; ve o an için değişken değeri olarak Mesafe= 0003 olsun. Buna göre, 0005 Adresli hafıza yeri "Mesafe" ismiyle anılır, taşıdığı değer yani 0005 Adresli hafıza yerinde tutulan ise 0003 -dür. Bu durumda, yafta olarak "Mesafe" -nin değeri 0005 (Adres); değişken olarak Mesafe -nin değeri ise 0003 -dür. Başka bir deyişle, "Mesafe" -nin (Adres) değeri 0005, Mesafe -nin değeri (muhtevası) ise 0003 -dür.
Bit-Muhtevası ⟨Bit-Contents⟩: Bir hafıza yerindeki bit-dizilimi.
Değer-Muhtevası ⟨Value-Contents⟩: Bit-muhtevasının temsil ettiği değer.
Değerin Temsil Biçimi ⟨Form of Representing Data⟩ (Kod⟨Code⟩): Hangi değer-muhtevasının hangi bit-muhtevasına denk geldiği.
Bit-Muhtevasının Yorumu ⟨Interpretation of Bit-Contents⟩: Bir bit-muhtevasının hangi değer-muhtevasına denk geldiği.
Not: Bit-içeriğinin yorumu, temsil edilen değerin cinsine ve kullanılan koda (hangi kurala göre temsil edildiğine) bağlıdır. Bu bilgiler olmadan, değer-içeriği olmaz. Örneğin, 0100000101000110 şeklindeki bit-içeriğinin salt olarak bir anlamı yoktur. Halbuki, bir tamsayı olduğu ve 2 tabanında yazıldığı verilirse, 16710 olarak yorumlanır. Öyle değil de, bir metin parçası olduğu ve ASCII kodunun kullanıldığı verilirse, iki harflik "AF" kelimesi olarak yorumlanır. Dolayısıyla, bir bit-içeriğinin tekabül ettiği değer-içeriğini belirlerken değerin cinsini ve temsil biçimini dikkate almak zorunludur.
Komut⟨Instruction⟩ (Makine Komutu ⟨Machine Instruction⟩): Bilgisayarın ifa edebildiği bir birimlik işlem. Bazen o işlemin belirteci (ismi veya kodu) de "**komut**" olarak anılır.
Komutun Yorumlanışı ⟨Interpretation of Instruction⟩: Komutun hangi işlemi belirttiğinin çözümlenişi.
Komutun İfası ⟨Execution of Instruction⟩: Komutun belirttiği işlemin tamamen yapılabilişi için gerekli hareketlerin yerine getirilişi.
Makine Dili ⟨Machine Language⟩: Komut ve verilerin bit dizileriyle ifade edildiği izlence dili. (Makine tarafından yorumlanışa uygun olan bu dilin anlaşılışı insanlar için çok zordur.)
Simgesel Makine Dili ⟨Symbolic Machine Language⟩ (Simgesel Dil ⟨Symbolic Language⟩): Adreslerin, komutların ve verilerin simgelerle ifade edilişlerine imkan tanıyarak makine dilinin zenginleştirilişi suretiyle elde edilen dil. (Bunun insanlar tarafından anlaşılışı makine diline göre daha kolaydır.)
Simgesel İzlence ⟨Symbolic Program⟩: Simgesel dilde yazılmış izlence.

Yürütülebilir Halde İzlence ⟨Program in Executable Form⟩: Makine dilinde bir izlence. Böyle bir izlence, bit dizileriyle ifade edilen komutlar ile aralarındaki yardımcı verilerden oluşur ve makine tarafından yorumlanarak yürütülüşe hazır durumdadır. Makine diline çevrilmemiş bir izlence (makine tarafından) yürütülemez.
Çevirici⟨Translator⟩: Belli bir dildeki bir izlenceyi, diğer bir dile çeviren bir **izlence**.
Kaynak İzlence ⟨Source Program⟩: Çeviricinin girdisini teşkil eden izlence.
Amaç İzlence ⟨Object Program⟩: Çeviricinin çıktısını teşkil eden izlence.
Birleştirici⟨Assembler⟩: Bir simgesel dilden makine diline çeviri yapan çevirici.
Yüksek Düzeyli Dil ⟨High Level Language⟩: Simgesel dillere nispetle kullanımı ve anlaşılışı kolay, insan diline yakın izlence dili. (FORTRAN, BASIC, ALGOL, PL/1, PASCAL, ADA, C^{++} gibi.)
Deyim⟨Statement⟩: Yüksek düzeyli dillerde kullanılan birim ifade. Böyle dillerde izlenceler deyimlerden oluşur.
Alçak Düzeyli Dil ⟨Low Level Language⟩: Makine dili düzeyinde izlence dili. (Simgesel diller dahil.)
Derleyici⟨Compiler⟩: Bir yüksek düzeyli dilden bir alçak düzeyli dile çeviri yapan çevirici.
Derleyiş⟨Compilation⟩: Derleyicinin yaptığı çeviri işi.
Ana Hafıza ⟨Main Memory⟩:Yürütülebilir haldeki izlenceler ile verilerinin yürütüm esnasında tutulduğu esas hafıza.
Aritmetik ve Mantık Birimi ⟨Arithmetic and Logic Unit⟩: Bilgisayardaki aritmetik ve mantık işlemleri gibi genel işlemlerin yapıldığı birim.
Denetim Birimi ⟨Control Unit⟩: Ana hafızada yürütülebilir halde bulunan bir izlencenin makine tarafından yürütülüşünü sağlayan birim. Bu birim, izlencenin komutlarını mantıki sırasıyla tek tek alıp yorumlar ve yürütülüşü (gereğinin yerine getirilişi) için gerekli denetim işaretlerini üreterek gerekli noktalara gönderir.
İşlem Birimi ⟨Processing Unit⟩: Aritmetik ve mantık birimi ile denetim biriminin bileşimi.
Merkezi İşlem Birimi (MİB) ⟨Central Processing Unit (CPU)⟩: Bir adet işlem birimi olan bir bilgisayarın işlem birimi.
Bilgisayarın Çekirdeği ⟨Core of the Computer⟩: Bir işlem birimi ile ana hafızanın birleşimi.
Ön Hafıza ⟨Cash-Memory⟩: Bazen bir işlem birimi ile ana hafızanın arasına konan hızlı hafıza.
Giriş-Çıkış Birimleri(Aygıtları) ⟨Input-Output Units (Devices)⟩: Optik okuyucular, klavyeler gibi giriş birimleri, yazıcılar, ekranlar gibi çıkış birimleri, mıknatıslı şerit ve diskler gibi giriş-çıkış birimleri, gerilim/sayı, sayı/gerilim dönüştürücüleri gibi algılayış ve uyarış birimleri.
Çevre Birimleri(Aygıtları) ⟨Peripheral Units (Devices)⟩: Giriş-çıkış birimleri.
Aygıt Denetleyici ⟨Device Controller⟩: Bazen bilgisayarın çekirdeği ile giriş-çıkış birimleri arasına yerleştirilen ve aradaki veri alış-verişini düzenleyen birim.
Giriş-Çıkış İşlemcisi ⟨Input-Output Processor⟩: Bazı büyükçe sistemlerde aygıt denetleyiciler veya doğrudan giriş-çıkış birimleri ile bilgisayarın çekirdeği arasına yerleştirilen ve kendine bağlı çevre birimlerinin işleyişlerini düzenleyen özel amaçlı bir işlemci. Bir işlem birimine göre çok kısıtlı yetenekte olan böyle bir işlemcinin, aynen bir işlem birimi gibi ana hafızada yürüttüğü kendine ait bir izlencesi olur. Giriş-çıkış işlemcisi bu izlence uyarınca, üzerinden cereyan eden veri alış-verişini yönlendirir ve sıraya koyar. Böylece giriş ve çıkışların işlem birimi meşgul edilmeksizin doğrudan ana hafıza ile çevre birimleri arasında gerçekleştirilişini sağlar.
Çevresel İşlemci ⟨Peripheral Processor⟩: Giriş-çıkış işlemcisi.
İşletim Sistemi ⟨Operating System⟩: Kullanıcı izlencelerinin yürütülüşlerini düzenleyen, giriş-çıkış işlemlerini ve bazı diğer standart işlemleri gerçekleştirmek için hazır yordamlara sahip bulunup, bu işlemleri kullanıcı izlenceleri için basit hale sokan, bu arada sistem kaynaklarının hem kullanıcı, hem de sistem açısından en iyi bir şekilde değerlendirilişine yardımcı olan bir yazılım.

Arıza Teşhis İzlencesi ⟨**Diagnostic Program**⟩: Bir bilgi-işlem düzenini donanım arızaları yönünden rapor eden yazılım. Ne var ki, arıza olduğunda işlemeyebilir.
Yedekleyiş İzlencesi ⟨**Back-up Program**⟩: Bir bilgi-işlem düzeninde saklanan verilerin kaybolmaması için sistematik yolla yedekleyiş yapan izlence.
Bakım Yazılımı ⟨**Maintenance Software**⟩: Bir bilgi-işlem düzeninin bakımında kullanılan, arıza teşhis izlencesi ve yedekleyiş izlencesi gibi izlenceler.
Sistem Yazılımı ⟨**System Software**⟩: İşletim sistemi ve bakım yazılımı gibi, bir bilgi-işlem düzeninin kullanımını kolaylaştırmak için hazırlanmış yazılım. (Bazen çeviriciler de sistem yazılımına girebilir.)
Uygulayış Yazılımı ⟨**Application Software**⟩: Bir bilgi-işlem düzeninde bazı standart uygulayışların yapılabilişi için gerçekleştirilmiş hazır yazılım.
Mikroişlemci⟨**Microprocessor**⟩: Bir tümleşik devre halinde gerçekleştirilmiş bir MİB.
Mikrobilgisayar⟨**Microcomputer**⟩: MİB'i mikroişlemci olan küçük kapasiteli bilgisayar.
Kişisel Bilgisayar ⟨**Personal Computer**⟩: Hep aynı kişi tarafından kullanılacağı varsayımına göre tasarımlanmış mikrobilgisayar.
Çok Kullanıcılı Bilgi-İşlem Düzeni ⟨**Multi-User Information-Processing System**⟩: Farklı kullanıcıların ayırt edilebildiği ve bilgilerin bütünlüğü, güvenliği, sahibince verilen izinler nispetinde paylaşılabilişi gibi kullanıcı haklarının gözetilebildiği bilgi-işlem düzeni.
İş İstasyonu ⟨**Work Station**⟩: Kişisel bilgisayarlardan biraz daha güçlü yapısı olan çok kullanıcılı bilgi-işlem düzeni.
Sunucu⟨**Server**⟩: Büyükçe bir veri saklama kapasitesi olan ve iş istasyonlarının kendisine bağlanması suretiyle bundan istifade edilen, çok kullanıcılı bilgi-işlem düzeni.
Minibilgisayar⟨**Minicomputer**⟩: Sunuculardan bir boy büyük bilgisayar.
Orta, Büyük, Çok Büyük Bilgisayarlar ⟨**Medium, Large, Super Computers**⟩: Minibilgisayardan daha büyük olan ve daha fazla kullanıcıya hizmet edebilen bilgisayarlar. Bunlar arasındaki ayırım kapasitelerine göre daha ziyade sezgisel olarak yapılmaktadır, kabul edilmiş kesin ölçüler bulunmamaktadır.
Bilgisayar Ağı ⟨**Computer Network**⟩: Bilgisayarlar arasında iletişim kurulmasıyla gerçekleştirilen bir ağ. Bir binanın içinde bulunan bilgisayarlardan müteşekkil yerel ağlar (örneğin, bir sunucu ve ona bağlı iş istasyonları) olduğu gibi, Dünyayı saran, (Internet gibi) çok geniş alan ağları da bulunmaktadır.
İzlence Belirtimi ⟨**Program Specification**⟩: Bir izlencenin nasıl işleyeceğinin, gereken giriş-çıkış ilişkilerinin, uygun bir resmiyetle tanımlanışı.
Güvenilirlik⟨**Reliability**⟩: Bir izlenceden beklenen, belirtimine sadakatı.
Güvenilir⟨**Reliable**⟩: Güvenilir bir izlence, güvenilirliği %100 olan, yani hiç mantık hatası bulunmayan bir izlencedir.
Taşınabilirlik⟨**Portability**⟩: Belli bir sistemde geliştirilmiş olan bir izlencenin değiştirilmeksizin değişik sistemlerde kullanılabiliş özelliği.
Yaşatılabilirlik⟨**Maintainability**⟩: Bir izlence üzerinde sonradan düşünülecek değişikliklerin mantık hatalarına yol açmadan gerçekleştirilebiliş özelliği.
Verimlilik⟨**Efficiency**⟩: Bir izlence için üç açıdan verimlilik söz konusudur. Kullandığı hafıza açısından, harcadığı zaman açısından, geliştiriliş emeği açısından. Genellikle bunların biri açısından verimin yükseltilişi için diğerlerinden fedakarlık gerekir.

Soru : Yazılım ile donanım arasında ne fark vardır? Cevap : Birisine yer çekimi etki etmez, diğerine eder.

SOYUT⟨ABSTRACT⟩-SOMUT⟨CONCRETE⟩

Bazen soyut simgeler, kâğıt veya ekran gibi bir görüntü ortamı üzerinde somut **şekilcikler ⟨characters⟩** ile gösterilirler. **Şekilcik⟨character⟩** derken, kabaca ifadeyle, herhangi bir klavye **⟨keyboard⟩** (eskiye göre bir daktilo **⟨a typewriter of the old days⟩**) düğmesine basıldığında görüntü ortamına çıkabilecek bir **işareti ⟨sign⟩** kastetmekteyiz. Bu bir harf veya rakamın yazılışı veya bir noktalayış işareti olabileceği gibi, boşluk**⟨blank⟩** (boş şekilcik **⟨empty character⟩**) veya özel olarak hazırlanmış bir şekilcik de olabilmektedir.

Sayılardan rastgele birini, mesela **5** -i alalım. Bu sayı ile ilgili çeşitli varlıklar bulunmaktadır. Beş, 5, "Beş" sözü ve bir de o sayının kendisi. Bu dördünden ilki, o sayının yazıyla ifadesidir. İkincisi o sayının bir şekilcik (veya rakam) ile ifadesidir. Üçüncüsü yani "Beş", o sayının adıdır. Dördüncü ilgili varlık ise kendisidir. Her sayı gibi bu sayı da soyut olduğu için buraya yazmamız mümkün değildir. Yazdığımız ancak onu temsil eden "5" gibi somut bir şekilcik veya "Beş" gibi somut bir harf dizisidir. Ne var ki, biz çoğunlukla "Beşin temsilcisini yazıyorum" diyeceğimize, kısaca "beş yazıyorum" der ve "5" yazarız.

Soyut ve somut, bilinen bir sınıflayış tarzıdır. Biz bu sınıflardaki öğelere soyutlar ve somutlar diyelim. Somutlar, yukarıda "fiziksel gerçekler" derken kastettiklerimizdir. Soyutlar ise baş soyut olan akıldan kaynaklanan varlıklardır. (İdrak gibi benzer kavramları aklın kapsamı içinde düşünüyoruz). Demek ki insan, soyut aklı vasıtasıyla somut dünya ile uyum sağlayış çabası içinde bulunmaktadır. Bu çaba içerisinde sıkça yapılan işlemlerden birisi **soyutlayış ⟨abstraction⟩**, birisi de onun tersi bir işlem olan **eşleyiş** veya **çağrıştırıştır ⟨association⟩**. Soyutlayış, Somutların bir takım esas özelliklerini ayrıntılardan sıyırarak akılda bir yer vermektir. Doğada "beş" diye bir şey mevcut değildir. Ancak, beş koyun, beş kavun, beş taş olabilir. Bunlara bakıp, koyun, kavun veya taş oluşlarını ayrıntı kabul ederek ortak özelliklerini ayırırsak 'beş' kavramı soyutlanmış olur.

Eşleyişe gelince, basit bir örnek, somut "5" şekilciği ile soyut 'beş' sayısı arasında yapılan çağrıştırıştır.

Buna benzer konular, bilgisayar programlayışta da geçmekte olduğundan ayrıca önem taşımaktadır. Mesela bir hafıza yeri vardır; bir bunun adı veya adresi vardır; bir de muhtevası vardır. Muhtevası, somut olarak mevcut bulunan bir takım fiziksel şartların temsil ettiği soyut bir sayı veya diğer bir bilgidir.

BÖLÜM 3: BİLGİSAYARDA TEMEL KONULAR ⟨FUNDAMENTAL TOPICS IN COMPUTERS⟩:

İşleyiş Esasları ⟨Operating Principles⟩:

Bir bilgisayarın fizikî kısmına donanım, onun üzerinde çalışan izlence kısmına da yazılım denir. Donanımda aradığımız önemli özelliklerden bir tanesi **hatasızlık⟨faultlessness⟩** veya **güvenilirlik⟨reliability⟩** özelliğidir. Bir işlem birimi milyarda bir hata yapsa, günümüzün hızlarıyla hatasız dakikası olmaz. Hâlbuki biz sürekli olarak günlerce, aylarca çalıştığı halde hiç hata çıkmamasını isteriz. Bu bakımdan bilgisayarlarda bilgi temsil etmek için kullanılan

fiziksel nicelikler yalnızca '**Var ⟨On⟩**' veya '**Yok ⟨Off⟩**' şeklinde değerlendirilirler. Daha ince ölçümlere tabi tutulmazlar. Bu son derece basit değerlendirim esası, bilgisayarlara sahip oldukları yüksek güvenilirliği kazandırmaktadır.

Var-yok değerlendirimine tâbî tutulan bazı fiziksel nicelikler şunlardır:

Gerilim⟨Voltage⟩ (bir noktanın üzerinde ⟨at a point⟩)

Akım⟨Current⟩ (bir telde ⟨in a wire⟩)

Bağlantı⟨Connection⟩ (anahtar veya röle üzerinde ⟨over a switch or a relay⟩)

Mıknatıslanış⟨Magnetization⟩ (belli bir noktada ve belli bir yönde ⟨at a certain point and in a certain direction⟩)

Böyle bir fiziksel niceliğin belli bir yerdeki '**Var**' değerlendirimi **1**, '**Yok**' değerlendirimi de **0** ile temsil edilir. Bu suretle o noktada "bir **bit**" denen ve **0** veya **1** şeklindeki iki değerden birinde ve yalnız birinde olan bir bilgi öğesi temsil edilir.

Düşünelim: Bir bitten daha küçük bir bilgi öğesi olabilir mi? Sabit durumlu bir ortamda bilgi temsil edilebilir mi? Bilgi varlığı için değişebilir durumlu bir ortam gerekirse, "değişebilirlik" için en az kaç durum gerekli?

Bir **bit** aynı zamanda 'bir **bit**lik bilgi sığası' anlamında, bilgi sığası birimi olarak kullanılmaktadır. Bazen 8 bit yan yana geldiğinde ona **1 Bayt** denir. İkilik sayılarda **bit**, '**hane ⟨digit⟩**' yerine kullanılır.

Örnek: A ve B denen iki bağımsız noktadaki gerilimler **a** ve **b** bitlerini temsil etsin. Her birisi bağımsız olarak 0 ve 1 değerleri alabilen bu bitlere ayrı ayrı anlamlar tayin edelim.

a	Anlamı	**b**	Anlamı
0	= 'su akmıyor'	0	= 'soğuk'
1	= 'su akıyor'	1	= 'sıcak'

Bu iki bit, **ab** olarak birlikte, ikiden fazla anlam taşımaktadır: 'su akmıyor', 'su soğuk akıyor', 'su sıcak akıyor'. Demek ki tek bir bit ile yalnız iki durumdan biri ifade edilebildiği halde, yan yana bitler kullanarak daha fazla bilgi ifade edilebilmektedir.

Genelde, n-bit ile 2^n adet anlam ifade edilebilmektedir. Yukarıdaki örnekte, su akmadığı takdirde soğuk veya sıcak oluşunun anlamı bulunmayacağı için 00 ve 01 birleşimlerinin ikisi de 'su akmıyor' bilgisini ifade etmektedir. Biz **istersek**, 'su akmıyor' anlamında yalnızca 00 birleşimini **⟨combination⟩** kullanırız ve 01 birleşimine **yeni bir anlam** tayin edebiliriz, '**su ılık akıyor**' gibi. Bu takdirde a ve b bitlerinin her biri ayrı bağımsız anlamlar taşıyacak yerde, **birleşimler**i anlamlar taşır hale gelmiş olur. ⟨In this case, **combination**s of the bits a and b happen to carry meanings rather than each bit carrying some independent meaning separately.⟩

ab	Anlam
00	su akmıyor
01	su ılık akıyor
10	su soğuk akıyor
11	su sıcak akıyor
2 bit: $2^2 = 4$ anlam.	

Bitlerle Sayıların İfadesi ⟨Expressing Numbers Using Bits⟩:

Yukarıda geçen örnekte anlatıldığı üzere, bit dizileriyle çeşitli bilgiler ifade edilebilir. Bu bilgiler özellikle komutlar veya sayılar olabilir. Bilgisayarlarda sayıların ifadesi için temel olarak 2-lik ⟨**binary**⟩ (2-tabanlı ⟨**base-2**⟩) sayı düzeni ⟨**number system**⟩ kullanılır. Bu 2-lik sayı düzenini kolay anlayabilmek için önce günlük hayatta kullandığımız sayı temsil düzenini ⟨**number representation system**⟩ incelemekte yarar vardır.

Günlük hayatta sayıları ifade etmek için kullandığımız düzen, **10«on» -luk ⟨decimal⟩** (10-tabanlı ⟨**base-10**⟩) sayı düzeni olarak adlandırılmaktadır. Rasgele bir örnek alalım: "**1983**". Bu, "bindokuzyüzseksenüç" şeklinde okuduğumuz sayının 10-luk ifadesidir. (Yoksa o sayının kendisi değildir.) **1, 9, 8 ve 3 rakamları, konumlarına göre tabanın kuvvetlerinin kat sayılarıdır ⟨The digits 1, 9, 8 and 3 are the coefficients of the powers of the base, according to their positions⟩**:

$$1983_{10} = 1 \times 10^3 + 9 \times 10^2 + 8 \times 10^1 + 3 \times 10^0 = 1000 + 900 + 80 + 3 = 1983$$

Böylece kastettiğimiz sayı ortaya çıkmaktadır. Görüldüğü üzere, ifadedeki bir rakamın hangi rakam olduğu kadar, hangi konumda olduğu da, bir düzen dairesinde, ifade edilen sayıyı belirlemektedir. Örneğin, 1893, aynı rakamları ihtiva etmesine rağmen, farklı bir değer ifade etmektedir. Bu düzende tabanın 10 oluşuna karşılık, 10 adet rakam (0, 1, ..., 9) gerekmektedir.

10-luk sayı düzeni, genel olarak "**konumsal sayı temsil düzenleri ⟨positional number representation systems⟩**" olarak bilinen sınıfa dâhildir. Aynı sınıftaki düzenlerden birisi de 2-lik sayı düzenidir. Bir bit ile sadece 0 ve 1 şeklindeki iki rakam temsil edilebildiğine göre, sayıların bitlerle ifadesi için 2-lik bir sayı ifade düzeni gereklidir. 10-luk düzenden örnek alınırsa, 2-lik düzende sayılar yalnız 0 ve 1 rakamları kullanılarak ifade edilir ve bu rakamlar 2-nin (tabanın) kuvvetlerinin katsayıları olur. 1-ler, 10-lar 100-ler 1000-ler vs. haneleri yerine, 1-ler, 2-ler, 4-ler 8-ler, vs. haneleri olur.

Bir ifade için farklı tabanlar söz konusu olduğunda, hangi tabanın esas alındığını, ifadenin sağ alt köşesinde belirtmek gerekir.

Kayıt: Ara hesaplarda yine eski alışkanlığımıza bağlı kalınarak 10-luk ifadelerden yararlanılmaktadır. Gerçekte aynı hesap başka tabanlara göre ifadelerle de yapılabilir; ancak buna gerek olmadığı gibi ara hesaplarda geçen sayı ifadelerinin 10-luk olduğunu sağ alt köşelerinde belirtmeye de gerek duyulmamaktadır.

Örnek:

1011_2**«**bir-sıfır-bir-bir-taban-iki⟨one-zero-one-one-base-two⟩**»**

$= 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0 = 8+0+2+1 = 11_{10}$**«**onbir⟨eleven⟩**»**

Kesirli sayılarda da ilke aynıdır:

Örnek:

0.6875_{10}«sıfır-nokta-altı-sekiz-yedi-beş-taban-on
⟨zero-point-six-eight-seven-five-base-ten⟩»

$= 6\times10^{-1} + 8\times10^{-2} + 7\times10^{-3} + 5\times10^{-4}$

$= 0.6 + 0.08 + 0.007 + 0.0005$

$0.1011_2 = 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3} + 1\times2^{-4}$

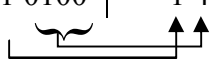
$= 1\times0.5+0\times0.25+1\times0.125+1\times0.0625$

$= 0.5 + 0 + 0.125 + 0.0625 = 0.6875_{10}$

Alışkın olduğumuz 10-luk düzen ve makineye uygun olan 2-lik düzen dışında bizi ilgilendiren diğer bir konumsal sayı temsil düzeni "**16-lık düzen ⟨hexadecimal system⟩**" -dir. Tabanının $16= 2^4$, yani ikinin bir kuvveti, oluşundan dolayı bu düzenin, 2-lik düzen ile çok basit bir ilişkisi vardır; ve ileride görüldüğü gibi, 2-lik ifadelerin kısa biçimde yazılıp okunuşunda kullanılırlar. İfade düzeni 16-lık olunca, 16 adet rakam gerekmektedir. Bu rakamların on tanesini, 10-luk düzeninkiler, geri kalan altı tanesini de A,B,C,D,E,F, simgeleri teşkil etmektedir. Her üç düzende sıfırdan itibaren sayılar şöyledir:

Cetvel⟨Table⟩: 1

10-luk düzen ⟨decimal system⟩	2-lik düzen ⟨binary system⟩	16-lık düzen ⟨hexa-decimal system⟩	10-luk düzen ⟨decimal system⟩	2-lik düzen ⟨binary system⟩	16-lık düzen ⟨hexa-decimal system⟩	10-luk düzen ⟨decimal system⟩	2-lik düzen ⟨binary system⟩	16-lık düzen ⟨hexa-decimal system⟩
0	0	0	7	111	7	14	1110	**E**
1	1	1	8	1000	8	15	1111	**F**
2	10	2	9	1001	9	16	1 0000	1 0
3	11	3	10	1010	**A**	17	1 0001	1 1
4	100	4	11	1011	**B**	18	1 0010	1 2
5	101	5	12	1100	**C**	19	1 0011	1 3
6	110	6	13	1101	**D**	20	1 0100	1 4



Bundan sonraki kısımlarda (doğrusu olan) "sayının 10-luk temsili" demek yerine kısaca "10-luk sayı" denmektedir. Yine "2-lik sayı" veya "16-lık sayı" deyişleri de benzer şekilde kullanılmaktadır.

1.3.2. Sayı İfade Düzenleri Arasında Geçişler ⟨Conversions Between Number Representation Systems⟩:

Genel olarak T tabanında abc.def gibi yazılan bir sayının 10 -luk ifadesi doğrudan genel bir formülden hesaplanır:

$a\times T^2 + b\times T^1 + c\times T^0 + d\times T^{-1} + e\times T^{-2} + f\times T^{-3}$

Arada (c ile d -nin arasında) görülen noktaya **Taban Noktası ⟨Base Point⟩** denir. Taban noktasının sağındaki rakam, T^{-1} -in katsayısıdır. Bu noktanın sağına yazılacak rakam bulunmadığı zaman genellikle nokta kullanılmaz.

2 -lik bir sayının 10 -luk karşılığı ⟨decimal representation of a binary number⟩: Doğrudan genel formülden çıkar.

Örnek: $1001101.1011_2 = (?)_{10}$

$1001101_2 = 1\times2^6 + 0\times2^5 + 0\times2^4 + 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0$
$= 64 + 8 + 4 + 1$
$= 77_{10}$

$0.1011_2 = 0.6875_{10}$ (önceki bir örnekte hesaplanmıştı.)
Demek ki, $1001101.1011_2 = 77.6875_{10}$

10 -luk bir sayının 2 -lik karşılığı ⟨binary equivalent of a decimal number⟩: Sayıları 10 -luktan 2 -liğe çevirmekte, tamsayı ve kesir kısımları ayrı-ayrı ele alınır. ⟨In converting numbers from decimal to binary, the integral and fractional parts are handled separately⟩:

Tamsayı kısmı için "ikiye böl" yöntemi ⟨The "divide-by-two" method for the integer part⟩: Tamsayı kısmı, sıfıra ulaşana kadar sürekli ikiye bölünür. Bu arada artanlar sırasıyla 1 -ler, 2 -ler, 4 -ler vs. hanelerini verir.

Örnek: $77_{10} = (?)_2$

77	artan
77/2 = 38	1
38/2 = 19	0
19/2 = 9	1
• 4	1
• 2	0
• 1	0
• 0	1

Demek ki $77_{10} = 1001101_2$ (sonuç)

Kesir kısmı için "ikiyle çarp" yöntemi:
⟨The "multiply-by-two" method for the fractional part⟩:
Kesir kısmı alınır, ikiyle çarpılır, tamsayı kısmı ayrılır, kalan kesir kısmıyla aynı işlem tekrarlanır. Ayrılan tamsayı kısımları (bazen 0 da olabilir) sırasıyla 1/2 -ler, 1/4 -ler, vs. hanelerini verir. Bu işlemler sırasında elde edilen kesir kısmı sıfır çıkarsa durulur.

Örnek : $0.6875_{10} = (?)_2$

tamsayı		
	.6875	(.6875×2)
1	.375	(.375×2)
0	.75	(.75×2)
1	.5	(.5×2)
1	.0	=**durulur**=

Demek ki $0.6875_{10} = 0.1011_2$

Bazen elde edilen kesir kısmı sıfır çıkmak yerine daha önce elde edilmiş bir kesir kısmının tekrarı çıkabilir. Bu takdirde ikilik sayıda devir var demektir.

Örnek: $0.3_{10} = (?)_2$

tamsayı		
	.3	(.3×2)
0	.6	(.6×2)
1	.2	(.2×2)
0	.4	•
0	.8	•
1	.6	(.6 tekrar)

Demek ki $0.3_{10} = 0.0\overline{1001}_2$

Bazen işlemlere devam ederken ikilik kesiri temsil edecek yeterli sayıda bit elde edildiği halde, sıfır veya tekrara rastlanmamış olabilir. Böyle olduğu zaman işlem kesilir ve sonucun elde edilen kadarı kullanılır. Bilgisayarlarda ise, kesir sayılar devirli-devirsiz diye ayırt edilmezler. Doğrudan belli sayıda bit kapasitesi olan bir yere sayının düz yazılmayla sığan kadarı konur, geriye kalanı ihmal edilir.

Örnek: Kesir için ayrılmış 8 bitlik bir yere 0.3_{10} kesiri konduğu zaman 0.01001100_2 şeklinde görünür.

Alıştırım:

1. Son örnekteki ikilik sayının onluk karşılığını bulunuz ve hatanın 2^{-8} mertebesinde olduğunu gösteriniz.
2. İkilik düzende tam karşılığı olmayan (yani devirli çıkan) bütün diğer tek basamaklı onluk kesirleri buluruz.
3. Genel olarak her hangi bir kesir
   a) 16 bitlik bir yere
   b) 32 bitlik bir yere
   c) 64 bitlik bir yere

   konmak için sığmayan kısmı kesilirse, kesme hatası ne mertebede olur? Onluk düzende ifade ediniz.

İkilik ve Onaltılık düzenler arasında geçişler ⟨Conversions between the Binary and Hexadecimal systems⟩:

Onaltılık rakamlar ile dört bitlik ikilik sayılar arasında tam olarak bire-bir eşleme vardır. Bu eşleme tablo 1 -de görülmektedir. Dolayısıyla, onaltılık bir sayının ikilik karşılığının bulunuşu, onaltılık sayıdaki her rakam yerine karşılığı olana dört bitin yerleştirilmesinden ibarettir. Bu arada dikkat edilecek bir husus, dörtten az bitle yazılabilen l6 -lık rakamların da sıfır eklenerek dört bitle yazılışıdır. Aksi halde solda kalacak bitlerin yerleri korunmamış olur.

Örnek: $91.3A_{16} = (?)_2$

$$\underbrace{9}_{1001}\ \underbrace{1.}_{0001.}\ \underbrace{3}_{0011}\ \underbrace{A}_{1010}$$

Demek ki $91.3A_{16} = 10010001.00111010_2$

İkilik bir sayının onaltılık karşılığının bulunmasında yukarıdaki işlemin tersi yapılır. Yani 16 -lık rakamların karşılığı olacak dörtlü bit gurupları bulunur. Burada dikkat edilmesi gerekli husus, bitler dört-dört guruplanırken taban noktasına yakın yerden başlanması ve özellikle (varsa) kesir tarafının en sağındaki gurubun eksik kalması halinde bitlerin sıfırlarla dörde tamamlanmasıdır.

Örnek: $111100.101101_2 = (?)_{16}$

$$= 11,1100.1011,01$$
$$= \underbrace{0011}_{3},\underbrace{1100}_{C.}.\underbrace{1011}_{B},\underbrace{0100}_{4}$$

Demek ki $111100.101101_2 = (3C.B4)_{16}$

Alıştırım:

1. Genel formülle son iki örnekteki sayıların 10 -luk karşılıklarını bularak oradaki işlemlerin sağlayışlarını yapınız.
2. $32.5C_{16} = (?)_2$
3. $52.375_{10} = (?)_{16}$

İKİ TABANINDA DÖRT İŞLEM ⟨FOUR OPERATIONS IN BASE-2⟩:

Toplayış⟨Addition⟩: Artırılan⟨Augend⟩+ Artıran⟨Addend**⟩= toplam⟨sum⟩= yekûn⟨total⟩**
Çıkartış⟨Subtraction⟩: Eksiltilen⟨minuend⟩- eksilten⟨subtrahend⟩= fark⟨difference⟩
Çarpış⟨Multiplication⟩: Çarpan⟨multiplier⟩× Çarpılan⟨multiplicand⟩= Sonuç⟨product⟩
Bölüş⟨Division⟩: Bölünen⟨dividend⟩÷ Bölen⟨divisor⟩= Bölüm⟨quotient⟩, artan⟨remainder⟩

İkilik sayılarla dört işlem, ilkesel olarak onluk sayılarda olduğu gibidir. Şu farkla ki, ikilik sayıların rakamları 0 ve 1 den ibaret olduğundan, işlemler daha basittir. Burada tamsayılar ele alınmaktadır. Sayılarda kesir haneleri olduğu takdirde yapılacak ayarlayışlar, yine onluk sayılarda yapılanlara benzer tarzdadır.

1- İkilik Tamsayılarda Toplayış ⟨Addition in Binary Numbers⟩:

İki ikilik tamsayı toplanacağı zaman, "**birler hanesi ⟨ones digit⟩**" yani **en sağ bit ⟨rightmost bit⟩** aynı hizaya gelecek şekilde alt alta yazılır ve onluk sayılara benzer şekilde **en sağdan başlanarak** toplamın bitleri bulunur. Her hangi bir bit konumunda sağdan bir elde⟨**carry**⟩ (e) bitinin gelip gelmeyişine göre sonuçlar:

	0 + 0 0	e← 0 + 0 1	0 + 1 1	1 + 0 1	e← e← 0 + 1 0	e← e← 1 + 0 0	e← 1 + 1 0	e← e← 1 + 1 1
Sağdan elde:	**Yok**	**Var**	**Yok**	**Yok**	**Var**	**Var**	**Yok**	**Var**
Sola elde:	**Yok**	**Yok**	**Yok**	**Yok**	**Var**	**Var**	**Var**	**Var**

Örnekler: Toplamın elde edilen bitleri, altta t_i olarak gösterilmektedir.

Ö1.	Ö2.	Ö3.	Ö4.
0 1 + 1 0 1 1 t_1 t_0	e ↓ 1 1 +↓ 1 0 1 0 1 t_2 t_1 t_0	e e ↓ 0 1 +↓ 1 1 1 0 0 t_2 t_1 t_0	e e e 1 1 0 1 0 + 1 0 1 0 1 0 0 1 0 0 t_5 t_4 t_3 t_2 t_1 t_0
Bir sıfır daha bir(t_0), elde yok. Sıfır bir daha bir(t_1), elde yok.	Bir sıfır daha bir(t_0), elde yok. Bir bir daha sıfır(t_1), elde var bir(t_2).	Bir bir daha sıfır(t_0), elde bir. Elde ile sıfır bir eder, bir daha sıfır(t_1), elde bir(t_2).	Sıfır sıfır daha sıfır(t_0). Bir bir daha sıfır(t_1), elde bir. Elde ile sıfır ve yine sıfır, bir(t_2). Bir bir daha sıfır(t_3), elde bir. Bir daha sıfır(t_4), elde bir(t_5).

Alıştırımlar: (Kendinizce üretip irdeleyebilirsiniz.)

2- İkilik Tamsayılarda Çıkartış ⟨Subtraction in Binary Numbers⟩:

X ve Y gibi iki sayı için X- Y şeklindeki **fark⟨difference⟩** bulunmak istendiğinde, Y konumundakine "**eksilten⟨subtrahend⟩**", X konumundakine de "**eksiltilen⟨minuend⟩**" denir.

Dikkat: Çıkartışın bazı klasik anlatılışlarında, onluk düzende yapılan **92- 7** gibi bir işlem şöyle nakledilir:

"2 -den 7 çıkmaz, 9 -dan 1 alınıp sağındaki konuma verilir, böylece oluşan 12 -den 7 çıkar ve 5 kalır, sol basamakta kalan 8 olduğuna göre, sonuç 85 olur."

Halbuki aynı işlem, bilgisayardaki işleyişe göre anlatılırsa şöyle olur:

"2 -den 7 çıkar ve farkın ilgili hanesi 5 olur, ancak soldaki konuma da **borçluluk bildirimi ⟨borrow notice⟩** yapılır. Borçluluk hesabı, 9 -dan 1 çıkartılarak karşılanır ve sonuç 85 olur."

Burada iki husus belirginleşmektedir:

a- Eksiltilenin basamağındaki rakam, eksilteninkinden küçük olduğunda, farkın mütekabil basamağı yine de tanımlıdır.
b- Böyle bir durumda bilgi akış yönü sağa doğru değil, sola doğrudur.

İkilik tamsayılarda çıkartım yapmak için eksiltilen ve eksilten, "birler hanesi ⟨ones digit⟩" aynı hizaya gelecek şekilde alt alta, sırasıyla yazılır ve onluk sayılara benzer şekilde **en sağdan başlanarak** fark bitleri bulunur.

Her hangi bir bit konumunda sağdan bir **borçluluk bildirimi ⟨borrow notice⟩ ‹b›** gelip gelmeyişine göre sonuçlar şöyledir:

‹b›=Borçluluk Bildirimi	0 - 0 0	b← 0 - 0 1	b← 1 - 0 1	b← 1 - 0 0	b← 0 - 1 1	b← 0 - 1 0	b← 1 - 1 0	b← 1 - 1 1
Sağdan ‹b›:	**Yok**	**Var**	**Yok**	**Var**	**Yok**	**Var**	**Yok**	**Var**
Sola ‹b›:	**Yok**	**Var**	**Yok**	**Yok**	**Var**	**Var**	**Yok**	**Var**

Şayet en soldaki bit konumundan sola borçluluk bildirimi olursa, eksiltilen eksiltenden küçük demektir. Bu duruma "**iflas⟨underflow⟩**" denir.

Örnekler: Farkın elde edilen bitleri, altta f_i olarak gösterilmektedir.

ö1.	ö2.	ö3.	ö4.
1 1 1 - 1 0 1 0 1 f_2 f_1 f_0	b b 1 0 0 1 - 1 1 1 1 0 f_2 f_1 f_0	b b b 1 1 0 0 0 - 1 0 1 0 0 1 1 1 0 f_4 f_3 f_2 f_1 f_0	b 0 1 - 1 0 **iflas** 1 1 f_1 f_0
Birden sıfır çıktı bir(f_0). Birden bir çıktı sıfır(f_1). En solda kalan bir ise f_2 olur.	Birden bir çıktı sıfır(f_0). Sıfırdan bir çıktı bir(f_1), sola **‹b›**. Sıfırdan **‹b›** çıktı bir(f_2), sola **‹b›**. Birden **‹b›** çıktı bitti.	Sıfırdan sıfır çıktı sıfır(f_0). Sıfırdan bir çıktı bir(f_1), sola **‹b›**. Sıfırdan **‹b›** ile sıfır çıktı bir(f_2), sola **‹b›**. Birden **‹b›** ile bir çıktı bir(f_3), sola **‹b›**. Birden **‹b›** çıktı sıfır(f_4).	Birden sıfır çıktı bir(f_0). Sıfırdan bir çıktı bir(f_1), sola **‹b›** "**iflas**".

Alıştırımlar: (Kendinizce üretip irdeleyebilirsiniz.)

3- İkilik Tamsayılarda Çarpış ⟨Multiplication in Binary Numbers⟩:

İki ikilik tamsayı çarpılacağı zaman, aşağıdaki örneklerde görüldüğü gibi, çarpılan sayı "Çarpılan Alanı ⟨Multiplicand Field⟩" kısmına, çarpan sayı da "Çarpan Alanı ⟨Multiplier Field⟩" kısmına yazılır. Daha sonra çarpanın haneleri sağdan sola doğru tek-tek ele alınır. "0" -a rast gelindiği zaman çarpanın sıradaki hanesine geçilir. "1" -e rast gelindiği zaman çarpılan, (en sağ biti çarpanın ele alınmış olan bitinin hizasına denk gelecek şekilde) "Ara İşlem Alanı ⟨Intermediate Operations Field⟩" kısmına yazılır. Nihayet, ara işlem alanına yazılmış olan sayılar, hizalara dikkat edilerek "Sonuç Alanı ⟨Result Field⟩" kısmında toplanır. Çarpanın en sağında sıfırlar varsa, aynı adet sıfır, elde edilen toplamın en sağına yapıştırılır. Böylece çarpım sonucu elde edilir.

Örnek-1:

```
     11011101   → Çarpılan Alanı
   ×   100100   → Çarpan Alanı
     11011101
+ 11011101      } Ara İşlem Alanı
  1111100010100 → Sonuç Alanı
```

Örnek-2: (Aynı sayılar)

```
      100100   → Çarpılan Alanı
  × 11011101   → Çarpan Alanı
      100100
     100100
    100100
   100100      } Ara İşlem Alanı
  100100
+ 100100
  1111100010100 → Sonuç Alanı
```

Görüldüğü üzere yapılan işlem, çarpılan sayının kaydırılarak toplanışı olarak özetlenebilir.

Çarpınız ve doğrulayınız, aynı işlemi çarpılan ile çarpanı takaslayıp tekrar ediniz:

```
11011 × 1001= ?
101001 × 110101= ?
1010010 × 11010110= ?
```

4- İkilik Tamsayılarda Bölüş ⟨Division in Binary Numbers⟩:

Yapılan işlemler aynen on tabanındaki işlemler gibidir. Şu farkla ki; on tabanında bölüş yapılırken sorulan "kaç kere var?" sorusu yerine, iki tabanında sadece "var mı?" sorusu sorulur; çünki varsa, zaten 1 kere vardır.

Bir ikilik tamsayı diğer bir ikilik tamsayıya bölüneceği zaman, bölen sayı "Bölen Alanı ⟨Divisor Field⟩" kısmına, bölünen sayı da "Bölünen Alanı ⟨Dividend Field⟩" kısmına, yazılır. Daha sonra bölünenin sol başından başlatılan bir "Arayış Aralığı ⟨Search Interval⟩", sağa doğru genişletilerek içinde bölen aranır. Arayış aralığı içindeki kısımda bölen ya "0" kere vardır, ya "1" kere vardır. Eğer "0" kere varsa, "Sonuç Alanı ⟨Result Field⟩" kısmının en sağına "0" hanesi ilave edilir ve arayış aralığı, bölünenin sıradaki hanesi dahil edilerek, sağa doğru bir hane genişletilir. Eğer "1" kere varsa, sonuç alanının en sağına "1" hanesi ilave edilir ve önce o andaki arayış aralığı içinde bulunan kısım bölen kadar eksiltilir, sonra bölünenin sıradaki hanesi dahil edilerek, arayış aralığı sağa doğru bir hane genişletilir. Nihayet, bölünenin bütün haneleri biter de, sıradaki hanesi arayış aralığına dahil edilmek istendiğinde sırada hane kalmadığı anlaşılırsa durulur. Bu noktaya gelindiğinde, sonuç alanında **Bölümün Sonucu ⟨Quotient⟩**, arayış aralığında da **Bölümün Artanı ⟨Remainder⟩** elde edilmiş olur. Bölümün artanı sıfırdan farklıysa, bölümün sonucu tam değil demektir. Bu

takdirde, bölünenin en sağında taban noktası olduğu ve onun da sağında sıfırlar olduğu kabul edilerek, bölüş işlemi, kesirli bir sonuç elde etmek üzere sürdürülebilir.

Örnek-1: **Kesirsiz (tam) sonuç**
⟨Exact result, without fraction⟩

```
                 000011011101   ← Bölüm
Bölen → 10010)111110001010     ← Bölünen
        -10010
         011010
         -10010
          0100000
           -10010
           0011101
            -10010
             010110
             -10010
              0010010
               -10010
                00000          ← Kalan
```

Örnek-2: **Kesirli (ve devirli) sonuç**
⟨Result with recurring fraction⟩

```
     001.10101...
  11)101.00000...
     -11
     010 0
      -1 1
      00 100
         - 11
          00100
            - 11
             001
               ...
 101/11= 1.101010...= 1.10
(10-tabanında: 5/3= 1.66...)
```

Bu örneklerden de görüldüğü gibi bölüş işlemi; bölünenden bölenin kaydırılarak çıkartılışlarıyla gerçekleştirilmektedir.

Bölünüz ve doğrulayınız ⟨Divide and verify⟩:

```
11011 ÷ 1001= ?
101001 ÷ 110= ?
1010010 ÷ 1010= ?
```

Bitlerle Sayıların İfadesinde Değişik Yaklaşımlar: ⟨Various Approaches In Expressing Numbers By Bits⟩:

Yansıtılı Kod ⟨Reflected Code⟩⟨Gray Code⟩

Sayısal düzenler açısından önemli, fakat konumsal ifade düzenlerinden farklı olan ve "**Yansıtılı Kod**" olarak bilinen bir sayı düzeni, aşağıdaki cetvelde gösterilmektedir.

Cetvel 2: Yansıtılı Kod ⟨Reflected Code⟩

10 -luk düzen	2 -lik düzen	**Yansıtılı Kod**	Yansıtım örüntüsü⟨Reflection pattern⟩ (Görülen 4 bit için ⟨For the 4-bits shown⟩)
00	0000	**0000**	
01	0001	**0001**	
02	0010	**0011**	
03	0011	**0010**	
04	0100	**0110**	
05	0101	**0111**	
06	0110	**0101**	
07	0111	**0100**	
08	1000	**1100**	
09	1001	**1101**	
10	1010	**1111**	
11	1011	**1110**	
12	1100	**1010**	
13	1101	**1011**	
14	1110	**1001**	
15	1111	**1000**	

Bunun neden Yansıtılı Kod olarak bilindiği, aşağıda işaretlenen yansıtım örüntüsünün incelenişinden daha iyi anlaşılabilir:

10 -luk düzen	**Yansıtılı Kod**
00	**000[0]**
01	**000[1]**
02	**001[1]**
03	**001[0]**

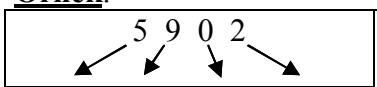
10 -luk düzen	**Yansıtılı Kod**
00	**00[00]**
01	**00[01]**
02	**00[11]**
03	**00[10]**
04	**01[10]**
05	**01[11]**
06	**01[01]**
07	**01[00]**

Sol tarafa yeni bir bit ilâve ederek kodlar bir kat artırılmak istendiğinde, mevcut kodlar aynadan yansır gibi yansıtılarak ters sırada cetvelin altına eklenir, soldaki yeni bit cetvelin üst yarısında 0, alt yarısında 1 yapılır.

Bu kodun önemli bir özelliği şudur:
Hangi **n** (**n**≥ 0) sayısı alınırsa alınsın, **n** ve **n**+ 1 sayılarının kodları arasında **sadece bir bit** fark eder. ⟨The codes of any two consecutive numbers differ by **only one bit**⟩

İkilik Kodlu Onluk (İKO) ⟨Binary Coded Decimal (BCD)⟩ Sayılar:

Onluk bir sayı, rakamlarının her biri ayrı-ayrı ikilik düzende ifade edilerek yazılabilir.
Örnek:



5902_{10}= (0101 1001 0000 0010)$_{\text{İKO⟨BCD⟩}}$

Böyle ifade edilen bir sayıya İkilik Kodlu Onluk (İKO⟨BCD⟩) sayı denir.

3-Artık Kod ⟨Excess-3 Code⟩ ⟨XS-3 Code⟩:

İKO Sayılara benzer bir kod düzeni, 3-Artık Kod olarak bilinmektedir. Bunda her bir 10 -luk rakam, aşağıdaki cetvelde görüldüğü üzere, üç fazlasının ikilik karşılığıyla temsil edilir.

10 -luk Rakam ⟨Decimal Digit⟩	İKO Karşılığı ⟨BCD form⟩	3-Artık Kod ⟨XS-3 Code⟩	(*)
0	0000	0011	3
1	0001	0100	4
2	0010	0101	5
3	0011	0110	6
4	0100	0111	7
5	0101	1000	8
6	0110	1001	9
7	0111	1010	10
8	1000	1011	11
9	1001	1100	12

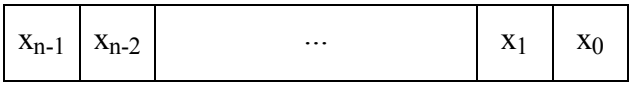
(*): 3-Artık Kod bitleri ikilik sayı olarak yorumlandığında 10 -luk karşılığı.
⟨The decimal equivalent of the binary interpretation of the bits of the XS-3 Code⟩

Dikkat: Bir rakamın İKO Karşılığı veya 3-Artık Kod karşılığında kullanılmayan birleşimler ⟨unused combinations⟩ vardır. Bir bit dizgisinin geçerli bir İKO gösterim veya 3-Artık Kod gösterimi oluşu için öncelikle 4 -er bitlik gurupların bitişiminden oluşuşu, sonra da her bir 4 bitlik gurubun, ilgili gösterime ait sütundaki birleşimlerden oluşu şarttır.

Örnekler:	**Examples**:
$(01010101)_{\text{İKO}} = 55_{10}$	$(01011000)_{\text{BCD}} = 58_{10}$
$(01010101)_{\text{3-Artık}} = 22_{10}$	$(001011000)_{\text{BCD}}$... invalid.
$(10101010)_{\text{3-Artık}} = 77_{10}$	$(0101010)_{\text{XS-3}}$... invalid.
$(00001010)_{\text{İKO}}$... Geçersiz	$(10000001)_{\text{XS-3}}$... invalid.
$(11000000)_{\text{İKO}}$... Geçersiz	$5902_{10} = (1000110000110101)_{\text{XS-3}}$

KAYITLIK ARİTMETİĞİ ⟨REGISTER ARITHMETIC⟩

Bilgisayarlardaki aritmetik, kayıtlıklarda gerçekleştirilir ⟨The arithmetic in computers are realized in registers.⟩. Bir n-bitlik kayıtlığın çizimsel gösterimi ⟨The graphical representation of an n-bit register⟩:



($n-1 \geq i \geq 0$ için her x_i, bir bit)

Şeklindedir. Burada "X", kayıtlığın ismi veya tamamını temsil eden simgedir; $n-1 \geq i \geq 0$ için her bir x_i, X -in bitlerinden birini temsil etmektedir; i indisi, ilgili bitin sağ baştan itibaren sola doğru olan konumunu belirtmektedir. Böyle bir X kayıtlığında bulunmakta olan $x_{n-1}x_{n-2}...x_1x_0$ şeklindeki bit dizgisine, bu kayıtlığın **bit-muhtevası ⟨bit-contents⟩** denir ve (X) şeklinde parantezli olarak gösterilir:

$(X)= x_{n-1}x_{n-2}...x_1x_0$ X kayıtlığının **bit-muhtevası ⟨bit-contents⟩**

Bit-muhtevası, yorumsuz**⟨uninterpreted⟩** muhtevadır. Burada ...

x_0 : En sağ bit ⟨Rightmost bit⟩

x_{n-1}: En sol bit ⟨Leftmost bit⟩

... olarak anılır. Bit-muhtevasında her bit (solda sıfır da olsa) gösterildiği için, buna bakıldığında, kayıtlığın kaç bitlik olduğu görülür. Söz gelişi, (X)= 0100 ise X -in 4-bitlik bir kayıtlık olduğu anlaşılır.

Değer muhtevası ⟨value-contents⟩, Bir X kayıtlığının bit-muhtevasına belli bir **Y yorumu⟨interpretation⟩** altında bakıldığında görülen değer **Y**(X) şeklinde gösterilir. Örneğin, (X)= 0100 ise, buna ikilik tamsayı (kısaca "Tamsayı$_2$") olarak bakıldığında Tamsayı$_2$(X)= 0100_2 yani dörttür. Aynı bit-muhtevasına Yansıtılı kod (kısaca "Yansıtılı") olarak bakıldığında ise, 0100 karşılığı 7 olduğundan dolayı, Yansıtılı(X)= 7 olur.
⟨For example, if (X)= 0100, when this is viewed as a binary integer ("Integer$_2$" in short) Integer$_2$(X)= 0100_2 namely four. In case, the same bit-contents is viewed as a Reflected Code ("Reflected" in short), Reflected(X)= 7 because 0100 corresponds to 7.⟩

Demek ki, bir kayıtlıkta hangi değerin olduğunu bilmek için bit-muhtevasını bilmek yeterli değil, "hangi yorum altında bakılacağı"nı da bilmek gereklidir.
⟨Therefore, in order to know the value in a register, knowing its bit-contents is not sufficient, knowing "the interpretation under which it is to be viewed" is also necessary.⟩

Belli bir **Y** yorumu⟨interpretation **Y**⟩ altında, bir **D** değerini tutan n-bitlik bir X kayıtlığının bit muhtevasını ifade etmek için Bit(**Y**(X)=**D**) tarzını kullanalım. Buna göre, **Bit(Yansıtılı(X)=7)= 00100** ifadesinden, 5-bitlik bir X-kayıtlığının olduğu ve Yansıtılı kodda 7 değerini ifade ettiğinden dolayı bit-muhtevasının 00100 şeklinde olduğu anlaşılmaktadır.

Genel: Toplayış ve çıkartış işlemlerinde aynı uzunlukta kayıtlıklar kullanılır. Çarpış işlemini toplayışlarla, bölüş işlemini de çıkartışlarla gerçekleştirmek mümkün olduğundan, aritmetik olarak burada daha ziyade, toplayış ve çıkartış işlemleri üzerinde durulmaktadır.

İşaretsiz İkilik Tamsayılarla Aritmetik ⟨Arithmetic Using Unsigned Binary Integers⟩:

İşaretsiz ikilik tamsayılar demekle kastettiğimiz, yukarıda anlatıldığı şekliyle ikilik tamsayılardır.

İşaretsiz İkilik Tamsayıların Kayıtlıklarda İfade Edilişi ⟨Expressing Unsigned Binary Integers in Registers⟩:

İşaretsiz ikilik tamsayıların kayıtlıkta ifadesi, aynen yazılışı gibidir.
A: k-bitlik bir işaretsiz ikilik tamsayı olsun. Bunu yazmak için en az k-bitlik bir kayıtlığa ihtiyaç vardır.
X: n-bitlik ($n \geq k$) bir kayıtlık olsun. Bu kayıtlığa A -nın yazılışı, A -nın bitlerinin X kayıtlığına, sağa yanaşık olarak yerleştirilişi ve solda artan kayıtlık hanelerinin 0 bitleriyle doldurulуşu şeklinde olur.

Örnek: 1011_2 sayısı 4-bitlik bir X-kayıtlığına yazıldığında Bit(X)= 1011 olur. Şayet X-kayıtlığı 6-bit olsaydı, Bit(X)= 001011 olurdu.

n-bitlik bir kayıtlığa sığabilecek en büyük sayı n-tane 1-bitinden oluşan sayıdır.
$(\underbrace{11\ldots1}_{\text{n-tane}})_2 = 2^n - 1$. Sonucu bunu aşan bir işlem, kayıtlığı taşırır.

"Topla" ve "Çıkart" işlemleri:

İşaretsiz ikilik tamsayıların kayıtlıklarda toplanışı ve çıkartılışı, kağıt üzerinde toplayışa ve çıkartışa benzer. Ancak,
Toplayışta en sol bit konumundan elde çıktığı takdirde, sonuç kayıtlık boyuna sığmamış olacağından, "**Taştı⟨overflow⟩**" hatası meydana gelir. (Sonuç> 2^n-1 ise.)
Çıkartışta en sol bit konumundan borçluluk bildirimi çıktığı takdirde "**İflas⟨underflow⟩**" hatası var demektir. (Sonuç< 0 ise; çünki sonuca işaret koymak mümkün değildir.)

Örnekler:

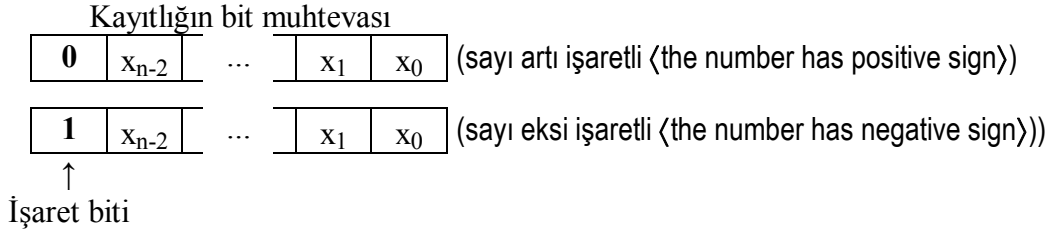
Bit(X)= 0100 + Bit(Y)= 0110 sonuç: 1010 (Doğru)	Bit(Y)= 1101 - Bit(X)= 0110 sonuç: 0111 (Doğru)	Bit(Y)= 1101 + Bit(X)= 0110 sonuç: 0011 (elde=1) (Yanlış: Taştı hatası.)	Bit(X)= 0100 - Bit(Y)= 0110 sonuç: 1110 (ödünç=1) (Yanlış: İflas hatası.)

İşaretli İkilik Tamsayılarla Aritmetik ⟨Arithmetic Using Signed Binary Integers⟩:

İşaretli ikilik tamsayılarla aritmetikte, sayının büyüklüğü yanı sıra, bir de işaretinin hesaba katılışı gerekmektedir. Bunun için mevcut olan farklı yöntemlerin başlıcaları şunlardır:

1. İşaret-Büyüklük Aritmetiği ⟨**Sign-Magnitude Arithmetic**⟩,
2. Bir-Tamlayan Aritmetiği ⟨**One's Complement Arithmetic**⟩,
3. İki-Tamlayan Aritmetiği ⟨**Two's Complement Arithmetic**⟩.

Bu aritmetiklerin üçünde de kayıtlığın en sol bitine "**işaret-biti ⟨sign-bit⟩**" denir ve sayının işaretini gösterir.



Sayının işareti artı (en sol bit 0) ise, büyüklüğünün ifadesi, işaretsiz ikilik tamsayılarda olduğu gibidir. Böyle olduğunda işaret biti, büyüklüğe tesir etmeyen bir "solda 0" şeklinde düşünülebilir. **Bu kabul her üç aritmetikte de geçerlidir.**

Ör. Bit(X)= 01011 ise Değer(X)= $+1011_2$= $+11_{10}$.

Eksi İşaretli İkilik Tamsayılar (İşaret biti 1) ⟨Binary Integers With Minus Sign (Sign bit 1)⟩: Eksi işaretli ikilik tamsayılarda büyüklüklerin ifade ediliş şekli kullanılan aritmetiğe bağlıdır. Farklılıklar, bu aritmetiklerdeki **işaret değiştiriş işlemleri**nin farklı oluşlarından kaynaklanmaktadır.

Kullanılan Aritmetik ⟨The Arithmetic Used⟩ →	**İşaret-Büyüklük Aritmetiği**	**Bir-Tamlayan Aritmetiği**	**İki-Tamlayan Aritmetiği**
İşaret Değiştirmek İçin ⟨To Change the Sign⟩ →	İşaret bitini tamla ⟨Complement the Sign bit⟩	Bütün bitleri tamla. ⟨Complement All the bits⟩	En sağdaki 1 -in solundaki bitleri tamla. ⟨Complement the bits on the left side of the rightmost 1⟩

Örnekler: Bir tane 5-bitlik X kayıtlığımız olsun.

Kullanılan Aritmetik→	**İşaret-Büyüklük Aritmetiği**	**Bir-Tamlayan Aritmetiği**	**İki-Tamlayan Aritmetiği**
Değer(X)= 12_{10}	Bit(X)= 01100	Bit(X)= 01100	Bit(X)= 01100
Değer(X)= - 12_{10}	Bit(X)= 11100	Bit(X)= 10011	Bit(X)= 10100
Bit(X) = 11100	11100= - 01100 Değer(X)= - 12_{10}	11100= - 00011 Değer(X)= - 3_{10}	11100= - 00100 Değer(X)= - 4_{10}
Değer(X)= 0_{10}	Bit(X)= 00000 veya Bit(X)= 10000 (iki farklı 0 ifadesi)	Bit(X)= 00000 veya Bit(X)= 11111 (iki farklı 0 ifadesi)	Bit(X)= 00000 (biricik 0 ifadesi)

İki-Tamlayan Aritmetiği:

İşaretli ikilik tamsayılarla aritmetik için bilgisayarlarda en çok kullanılan yöntem İki-Tamlayan Aritmetiğidir. Bu yöntemde, işaret değiştirmek için yukarıda gösterilen işleme "İki-Tamlayanını Almak" denir.

Neden İki-Tamlayan Aritmetiği ?:

İki-tamlayan aritmetiği, işaretsiz ikilik tamsayılarla aynı sayış sırasına sahiptir.

Kağıt-kalem aritmetiğine alışmışken bir kayıtlık aritmetiği olarak İki-Tamlayan Aritmetiği, başlangıçta yabancı, hatta karmaşık gelebilir. Öyleyse, "Neden iki-tamlayan aritmetiği?"

Bu iki-tamlayan aritmetiğinin üstünlüğü, algoritmik sadeliğinden ileri gelmektedir. Kağıt-kalem aritmetiğinde en basit bir a+b işlemini gerçekleştirmek için (genellikle şuurunda olmadan) izlediğimiz algoritmayı bir düşünecek olursak, oldukça karmaşık olduğunu fark ederiz. (İşaretler aynıysa, büyüklükleri topla; farklıysa büyüklüğü daha büyük olanı al; ondan diğerinin büyüklüğünü çıkart, ... -ayrıntılı bir akış çizelgesi yapmayı deneyiniz.-)

İki-Tamlayan Aritmetiğinde, X ve Y kayıtlıklarında bulunan işaretli ikilik tamsayıları aynı uzunluktaki bir Z kayıtlığına toplamak için yapılışı gereken işlem, işaretlerinin ne olduğu umursanmaksızın (X) ile (Y) nin, **işaretsiz ikilik sayılar** olarak verdiği toplamı Z -ye koymak ve bu arada en soldan elde-bir çıkıp çıkmadığına aldırmamaktan ibarettir.

Yani, İki-Tamlayan Aritmetiği, işaretli ikilik tamsayıların toplanması işlemini, en soldan elde-bir çıkmasına dahi bakılmaksızın işaretsiz ikilik sayıların toplandığı bir işleme indirgemektedir.

Örnekler: A= $+7_{10}$; B= $+5_{10}$ değerleriyle X ve Y şeklinde 5-bitlik kayıtlıklarda işlemler.
Bit[2Tam(X)= A]= 00111_2
00101_2 değerleriyle 5-bitlik kayıtlıklarda

A+B	00111 +00101 01100		00111		01011 +00011 01101		

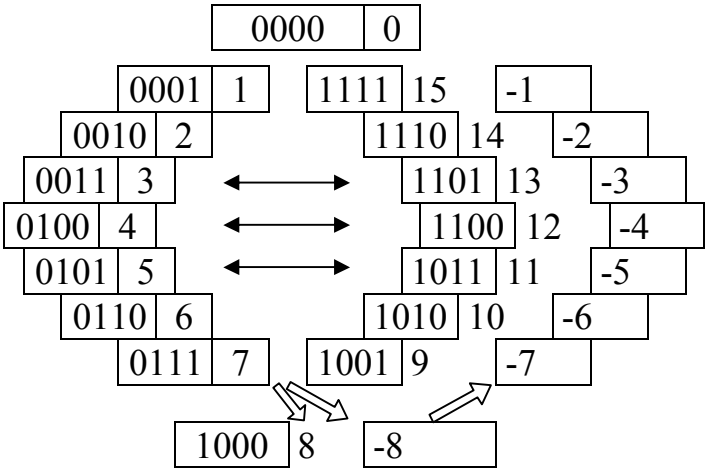
00110011
01010111

	Sütun 1	Sütun 2
Bit(x)	İşaretsiz Tamsayı	İşaretli Tamsayı
1000	8	-8
1001	9	-7
1010	10	-6
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2
1111	15	-1
0000	0	0
0001	1	1

0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7

Sütun1= Sütun2 + 2^n –2^n

Sütun2= Sütun1 + 2^n –2^n


0000 0
0001 1
1111 15
-1
0010 2
1110 14
-2
0011 3
1101 13
-3
0100 4
1100 12
-4
0101 5
1011 11
-5
0110 6
1010 10
-6
0111 7
1001 9
-7
1000 8
-8

İKİ-TAMLAYAN ARİTMETİĞİNE YAKINDAN BAKIŞ

Bilgisayarlarda işaretli ikilik tamsayılarla aritmetik yapmak için, kayıtlık ortamına uygun bir takım yöntemler vardır. Bunlardan en yaygın olanı, "İki-Tamlayan Aritmetiği"dir. Bu yöntemde,

Artı Sayılar: Kayıtlıklara bilinen ikilik düzende yazılırlar; ancak en sola en az bir adet sıfır bitinin sığışına dikkat edilir. Örneğin $+13_{10}= +1101_2$ sayısı, 01101 veya 001101 veya 0001101, vs. olarak yazılabilir; fakat 1101 olarak yazılamaz. Dolayısıyla, n-bitlik bir kayıtlığa yazılabilecek en büyük sayı, $011...1_2$ sayısıdır. Bu da $+2^{n-1}-1$ eder.

İki-Tamlayan Kavramı: İki sayıdan birisi bir kayıtlığa konup, diğeri kadar artırıldığında sonucun kayıtlıkta kalan kısmı sıfır oluyorsa (tabii, soldan bir "elde biti" çıkar) bu iki sayı birbirinin "İki-Tamlayan"ıdır.

İki-Tamlayanını Buluş: Herhangi bir ikilik sayının iki-tamlayanını bulmak için, en sağdaki 1'in solundaki bitler tamlanır.

Bu formüle göre tamlanacak bitleri olmadığından, iki-tamlayanı kendisine eşit olan iki istisna:
İstisna1: 00...0 ("En sağdaki 1" diye bir şey yok.) Dolayısıyla iki-tamlayanı yine kendisi.
İstisna2: 10...0 (En sağdaki 1'in solunda bit yok.) Dolayısıyla iki-tamlayanı yine kendisi.

Eksi Sayılar: Her artı sayının iki-tamlayanı, onun zıt işaretlisi olan eksi sayının yazılışını verir. Dolayısıyla en sol bit, artı sayılarda nasıl hep 0 ise, eksi sayılarda da hep 1'dir. Böylece, bir sayının iki-tamlayan aritmetiğindeki yazılışının en sol biti, onun işaretini belli eder.

0= "artı", 1= "eksi"

Genel: Her sayının yazılışının iki-tamlayanı, onun zıt işaretlisi olan sayının yazılışını vermektedir. Böylece, sayıların yazılışlarına uygulanan iki-tamlayanına dönüştürüş işlemi, sayılara uygulanan işaret değiştiriş işlemine tekabül etmektedir.

Örnekler:

Sayı	Yazılışı (5-bit)
+6	00110
-6	11010

Sayı	Yazılışı (5-bit)
+13	01101
-13	10011

Sayı	Yazılışı (7-bit)
+24	0011000
-24	1101000

En sol bitinin 1 olmasından "eksi" olduğu anlaşılan bir sayının mutlak büyüklüğünü gözle kolay görmek için, iki-tamlayanına dönüştürmek suretiyle işareti değiştirilmelidir.

İstisna1'e bakıldığında, sıfırın işaret değiştiriş işleminden etkilenmediği görülmektedir ki, olağan aritmetiğe uygundur.

İstisna2'ye bakıldığında, 100...0 yazılışının bir eksi sayıya ait olduğu, fakat aynı uzunluktaki bir artı sayıdan işaret değiştirmek suretiyle elde edilemediği görülmektedir. Bunun n-bit uzunluğunda olduğunu var sayalım. İki-tamlayanına dönüştürerek işaretini değiştirirsek "artı" işaret kazanacağı için 0100...0 şeklinde (n+1)-bit kullanarak yazmak zorunda kalırız. İfade ettiği değer ise 2^{n-1} olur. Demek ki, n-bit uzunluğunda 100...0 olarak yazılan sayı, -2^{n-1} değerini ifade etmektedir. Bunun artı işaretlisi ise yukarıda, n-bitlik bir kayıtlığa

yazılabilecek en büyük sayı olarak belirtilen, $+2^{n-1}-1$ değerinden daha büyük olduğu için n-bit ile ifade edilememesi doğaldır.

İki-Tamlayan Aritmetiğinde Toplayış İşlemi

Durum: X, Y ve Z, uzunlukları eşit kayıtlıklar. Yazılışları X ve Y kayıtlıklarında hazır olan sayıların toplamı Z kayıtlığına yazılacak.

İşlem: X ve Y kayıtlıklarının muhtevaları olduğu gibi, işaretsiz ikilik sayılar olarak toplanıp Z'ye yüklenir, en soldan çıkacak elde dikkate alınmaz.

"Taştı" hatasıyla karşılaşılmazsa, sonuç işaretiyle beraber doğrudur.

"Taştı" hatası: İşlem sonucunun mutlak büyüklük itibariyle, kayıtlığa sığmaması.
Ne zaman olabilir: Aynı işaretli sayılar toplandığı zaman.
Nasıl anlaşılır: Sonuç işaretinin farklı olmasından. (İki artı sayının toplamının eksi çıkması gibi)

İki-Tamlayan Aritmetiğinde Çıkartış İşlemi

A-B işlemi, A+(-B) şeklinde gerçekleştirilir.

Örnekler: A= $+7_{10}$; B= $+5_{10}$ değerleriyle 5-bitlik kayıtlıklarda işlemler.
Yazılışlar: A ➔ 00111; B ➔ 00101; -A ➔ 11001; -B ➔ 11011

A+B	A-B	-A+B	-A-B
00111	00111	11001	11001
+00101	+11011	+00101	+11011
01100 = + 12	e ← 00010 = + 2	11110	e ← 10100
		= -00010 = - 2	= -01100 = - 12

e ← : "en soldan çıkan elde" (sonuç bu eldeye bakmıyor.)

"Taştı" hatası:
Örnekler: A= $+14_{10}$; B= $+10_{10}$ değerleriyle 5-bitlik kayıtlıklarda işlemler.
Yazılışlar: A ➔ 01110; B ➔ 01010; -A ➔ 10010; -B ➔ 10110

A+B		-A-B	
01110	"artı"	10010	"eksi"
+01010	"artı"	+10110	"eksi"
11000	"eksi"	e ← 01000	"artı"

Dikkat: Bazen "en soldan çıkan elde" ile "taşış" durumu birbirine karıştırılmaktadır. Bu ikisinin tamamen başka kavramlar olduğu yukarıda açıkça görülmektedir.

İstisna2'de görüldüğü üzere, yazılışı 100...0 olan sayıya işaret değiştiriş işlemi uygulandığı halde işaretinin değişmemiş görünmesi de, bir tür "Taştı" hatası olarak yorumlanabilir.

İşaretli Tamsayıların, Kayıtlıklarda 2-Tamlayan Aritmetiğiyle Toplanışları ⟨Addition Of Signed Integers In Registers, Using 2's Complement Arithmetic⟩:

Soru Örnekleri ⟨Examples of Questions⟩:

S1. İşlemlerin 7-bitlik kayıtlıklarda, 2-tamlayan aritmetiğiyle yapılışlarını gösteriniz (bit hatası olan işlemin tamamı yanlış sayılır).
"Taştı hatası" hâlini, karşısına "Var" veya "Yok" yazarak belirtiniz; (Noksan veya yanlış belirtim diğer hata belirtimlerini götürür.)

(a)

$+21_{10}$ =	0	0	1	0	1	0	1
$+43_{10}$ =	0	1	0	1	0	1	1
Toplam=	1	0	0	0	0	0	0

Taştı hatası: Var

(b)

-21_{10} =	1	1	0	1	0	1	1
-43_{10} =	1	0	1	0	1	0	1
Toplam=	1	0	0	0	0	0	0

Taştı hatası: Yok

(c)

-21_{10} =	1	1	0	1	0	1	1
$+43_{10}$ =	0	1	0	1	0	1	1
Toplam=	0	0	1	0	1	1	0

Taştı hatası: Yok

(d)

$+21_{10}$ =	0	0	1	0	1	0	1
-43_{10} =	1	0	1	0	1	0	1
Toplam=	1	1	0	1	0	1	0

Taştı hatası: Yok

Q2. Show how 2's complement arithmetic operations in 8-bit registers are performed (a wrong bit makes the answer wrong)
Mark whether there is overflow or not, by putting a Yes or No next to "Overflow" (a wrong or missing mark cancels the others.)

(a)

$+21_{10}$ =	0	0	0	1	0	1	0	1
$+110_{10}$ =	0	1	1	0	1	1	1	0
Toplam=	1	0	0	0	0	0	1	1

Overflow: Yes

(b)

-21_{10} =	1	1	1	0	1	0	1	1
-107_{10} =	1	0	0	1	0	1	0	1
Toplam=	1	0	0	0	0	0	0	0

Overflow: No

(c)

-21_{10} =	1	1	1	0	1	0	1	1
$+110_{10}$ =	0	1	1	0	1	1	1	0
Toplam=	0	1	0	1	1	0	0	1

Overflow: No

(d)

$+21_{10}$ =	0	0	0	1	0	1	0	1
-107_{10} =	1	0	0	1	0	1	0	1
Toplam=	1	0	1	0	1	0	1	0

Overflow: No

Yüksek Seviyeli Bir Dildeki Bir İzlencenin Yürütülüşü
⟨The Execution of a Program Which is in a High Level Language⟩:

Genellikle yüksek seviyeli bir dil, makine dilinden farklıdır. Böyle olduğu zaman, o dilde yazılmış bir izlencenin (buna kullanıcı izlencesi diyelim) yürütülebilişi için öncelikle iki tarafı aynı dil seviyesinde buluşturmak gereklidir. Bunun için başlıca üç yol vardır:

a) İzlence makine diline çevrilir (uyarlanır)

b) Makine izlence diline uyarlanır.

c) Ara seviyede tanımlanan bir dil esas alınarak, bir yandan izlence o dile çevrilir, diğer yandan da, Makine o ara seviyedeki izlence diline uyarlanır.

Birinci yolda izlencenin anlatımı, tercüme yapılarak makinenin anlayıp icra edebileceği düzeye indirilir. Bu işleme "**Derleyiş⟨Compilation⟩**", bu işlemi yapan izlenceye de "**Derleyici⟨Compiler⟩**" denir.

İkinci yolda makinenin anlayış ve yorumlayış düzeyi, "**Yorumlayıcı⟨Interpreter⟩**" denen bir izlence vasıtasıyla, kullanıcı izlencesinin dil seviyesine yükseltilir.

Üçüncü yol, ilk iki yolun bir birleşiminden ibarettir.

Yorumlayıcı, belli bir dil için hazırlanır. Örneğin BASIC dili için gerçekleştirilmiş BASIC yorumlayıcıları vardır. Böyle bir yorumlayıcıya BASIC -de yazılmış bir kullanıcı izlencesi verildiğinde, yorumlayıcı o izlenceyi mantıksal sırasına göre izler ve her adımda ne yapılacağını çözerek onu makineye yaptırır. Bu arada makine yorumlayıcıyı, yorumlayıcı da BASIC izlencesini yürütmüş olur. Dolayısıyla yorumlayıcı altında işleyen bir makineye dışarıdan bakıldığında görülen, ilgili yüksek düzeyli dildeki izlenceleri doğrudan yürütebilen bir (zahiri) makine, yani dili o yüksek düzeyli dile uyarlanmış bir makinedir.

BASIC dilindeki izlencelerin yürütülüşü için yukarıda belirtilen iki yoldan birincisi de mümkündür.

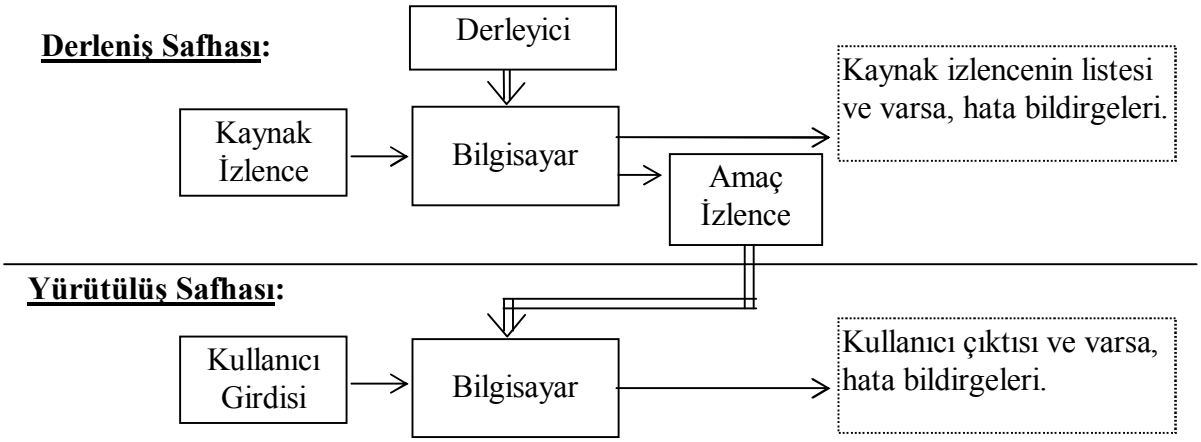
Yüksek seviyeli bir dildeki bir izlencenin yürütülüşünde iki safha vardır.

1. Derleniş safhası.
2. Yürütülüş safhası.

Bu safhalarda geçen olaylar Şekil 1 -de gösterilmektedir.

Derleyiş safhasında bilgisayarda yürütülen izlence derleyicidir. Bu izlencenin girdisi, kullanıcıya ait olan (**kaynak⟨source⟩**) izlence, çıktısı ise kaynak izlencenin karşılığı olan (**amaç⟨object⟩**) izlencedir. Aynı zamanda kaynak izlencenin bir listesi ve varsa, hata bildirgeleri de yan çıktı olarak elde edilebilir.

Yürütülüş safhasında bilgisayardaki yürütülen izlence, bir önceki safhadan çıktı olarak elde edilmiş bulunan amaç izlencedir. Bu kullanıcıya ait olan izlencenin yürütülebilir hali olduğu için girdisi, kullanıcının hazırlamış olduğu girdi verileri, çıktısı ise kullanıcının görmek istediği sonuçlardır. Yürütüm esnasında ortaya çıkan ve işletim sistemi tarafından fark edilen bazı hataların bildirgeleri de bu sonuçlar arasında bulunabilir.



⟶ : Girdi ve çıktı akışlarını belirtir.
⟹ : Yürütülecek izlencenin bilgisayara yüklenişini belirtir.

Şekil 1: Derleniş ve Yürütülüş Safhaları.

1.5. İzlenceleyişte Karşılaşılan Hatalar, Ortaya Çıkışı ve Fark Edilişi.

Bu kısımdaki ilkeler genel olmakla beraber, anlatımda izlenceleyiş için derleyicili bir dilin kullanıldığı varsayılmıştır. İzlenceleyişte hata konusu çok önemlidir. Akılcı yaklaşım, hataların olabileceğini baştan kabul ederek bunların nasıl giderilebileceği veya etkisizleştirilebileceği üzerine eğilmektir. Bu bakımdan hataların niteliklerini incelemekte yarar vardır.

İzlenceleyişte üç türlü hata ile karşılaşılır:

1. **Yazım hatası**
2. **Mantık hatası**
3. **Kesim ve yuvarlatım hatası.**

Yazım Hatası: Bir izlencenin, yazıldığı dilin yazım kurallarına uygunsuzluğu. Derleyiş safhasında ortaya çıkar. Derleyicinin çıktısındaki hata bildirgeleri bu yazım hatalarıyla ilgilidir. İzlenceci hatasını böylece fark eder. İzlencenin bilgisayara hazırlanışındaki harf hataları yazım hatalarının kaynaklarındandır. Genellikle yazım hatası bulunan bir izlence yürütülüş safhasına geçemez.

Mantık Hatası: Derleyiş safhasını geçmiş bir izlence yürütülüş safhasındayken ortaya çıkar. Bazı mantık hataları (ör. Bir ifade hesaplanırken her hangi bir değerin sıfıra bölünmesi) işletim sistemi tarafından fark edilebilir. Bu takdirde çıktıya hata bildirgesi verilir ve genellikle izlencenin yürütülüşü kesilir. Bazı mantık hataları ise (ör. 2 yerine 3 yazılmış oluşu) doğal olarak, işletim sistemince fark edilemez. Ancak, yanlış sonuçlar çıkmasına sebep olur. Bunlar izlenceci tarafından sağlama yoluyla fark edilebilir.

Yazım hataları giderilmiş bir izlencedeki mantık hatalarının ayıklanabilişi için kullanılan yaygın bir yöntem, deneyiş yöntemidir. Deneyiş yöntemi, izlencenin çeşitli girdilerle doğru çalışıp çalışmadığının denenişinden ibarettir. İzlenceleyişe yeni başlayan kişiler bu yöntemi benimseyebilirler. Ancak bu yöntemle ilgili aşağıdaki hususları bilmekte yarar vardır.

Deneyiş yöntemi, (özellikle büyükçe) bir izlencenin mantıksal doğruluğunu kanıtlamak için etkin bir yöntem değildir. Bu yöntemde hataya rastlanış hata varlığını gösterir, fakat hataya rastlanmayış, hata yokluğunu göstermez. Hata olsa bile o hatayı meydana çıkaracak girdi denenmemiş olabilir. Bununla beraber, çok sayıda girdilerle yapılan deneyişlerden sonra bir izlencede her hangi bir hatanın çıkmayışı, o izlenceye karşı bir ölçüde güven duygusu oluşumunu sağlar.

Yazılım güvenilirliğinin sağlanışı bazı ileri araştırılara konu olacak kadar önemlidir. Mantıksal hatalar bulunmayan izlencelerin elde edilebilişi için kullanılan yöntemlerde, yapısal izlenceleyiş ⟨**structured programming**⟩, doğruluk ispat teknikleri ⟨**proving the correctness techniques**⟩ ve "izlence belirtim ve geliştirim dilleri ⟨**program specification and development languages**⟩" gibi yaklaşımlardan yararlanılmaktadır.

Kesim ve Yuvarlatım Hatası :

Makinenin içinde sayı saklanan yerlerin hane kapasiteleri bellidir. Bu yerlere kapasitelerini aşan uzunlukta sayılar konmak istendiğinde fazla haneler kesilir veya yuvarlatılır. Makinedeki olayların ikilik veya onaltılık düzene göre cereyan etmesine rağmen kesim ve yuvarlatımı onluk düzende anlatmak belki anlaşılmayı kolaylaştıracaktır. Örneğin,

2/3 = 0.666...

Bu sayıyı iki kesir hanelik bir yere koymak isteyelim. Bunun için,üçüncü kesir hanesinden itibaren kesim yaparsak, elimizde kalan sayı 0.66 ve hatamız 0.00666... olur. Bir de yuvarlatım vardır. Yuvarlatımda, kesilen en büyük haneye bakılır. Eğer bu hane yarıya varmamışsa kesimdeki gibi yapılır. Yarıya varmış veya geçmişse, kalan sayının en küçük hanesi bir artırılır. Buna göre yukarıda yuvarlatım yapılmış olsaydı, kesilen en büyük hanenin 6 (yani yarıyı geçmiş) olduğu görülerek, kalan sayı 0.66 yerine 0.67 olarak alınırdı ve hata 0.00333... olurdu. Görülüyor ki, yuvarlatım işlemi kesim kadar basit olmamakla beraber, doğurduğu hata genelde kesimden daha az olmaktadır.

Örnek: Verilen sayıların kesim ve yuvarlatım ile üç kesir hanesine sığdırılışı:

Verilen Sayı	Kesim Sonucu	Yuvarlatım Sonucu
0.1234	0.123	0.123
0.1235	0.123	0.124
0.6529	0.652	0.653
0.65249	0.652	0.652

İncele-araştır: Aşağıdaki üç adet PASCAL izlencesini çalıştırarak gözlemlerinizi özetleyiniz.

```
Program HATA1;
var i: integer; Toplam, kesir: real;
begin while true do
 begin i:= 0; Toplam:= 0;
  writeln; write('Bir kesir giriniz: '); readln(kesir);
  repeat
   i:= i+ 1; Toplam:= Toplam+ kesir; writeln(i,'>>>', Toplam);
  until (Toplam<> i* kesir){olur mu hiç?} or (i> 99);
 end;
end.

Program HATA2;
```

```
var i: integer; Top, A: real;
  begin i:= 0; Top:= 0; A:= 1;
   repeat
    i:= i+ 1; A:= A/i; Top:= Top+ A; writeln(i, '>>>', Top);
   until (Top= Top+ A); {olur mu hiç?}
end.

Program HATA3;
var i: integer; X, Y: real;
  begin X:= 0; For i:= 1 to 2000 do X:= X+ 1/i;
        Y:= 0; For i:= 2000 downto 1 do Y:= Y+ 1/i;
        writeln(X,' === ', Y, ' === ', X-Y);
  end. {X'mi daha doğru, Y'mi?}
```

Genel olarak kesim ve yuvarlatım hatalarının incelenmesi ve etkisizleştirilmesi gibi konular, sayısal çözümleme alanına girerler.

1.6. Bilgisayarların etkin olduğu bazı uygulayış türleri:

Bilgisayarların en belirgin üstünlükleri:

- Süratli ve emniyetli bir şekilde aritmetik ve mantık işlemleri yapım yeteneği oluşu,
- Büyük bir bilgi saklayış kapasitesine sahip oluşu.

Buna göre, şu alanlarda etkin uygulayışları vardır:

-Çok sayıda verinin aynı işlemlere tabi tutuluşu (istatistik, vs.)

-Az sayıda veri olsa dahi, çok sayıda tekrar eden işlemlerden geçirilişi (yineleyişli çözümler.)

-Değişik parametrelerin denenerek en iyi parametrelerin bulunmaya çalışılışı.

-Bilgi biriktirilmesi ve birikmiş bilgilerin isteğe göre verilmesi (Uçakta yer ayırma sistemleri vs.)

-Fiziksel düzenlerde denetim ve karar temini (Endüstriyel uygulamalar vs.)

-Haberleşme, akitleşme, eş-güdüm (ağ uygulamaları, e-ticaret, vs.)

1.7. İzlence Hazırlayışın Safhaları:

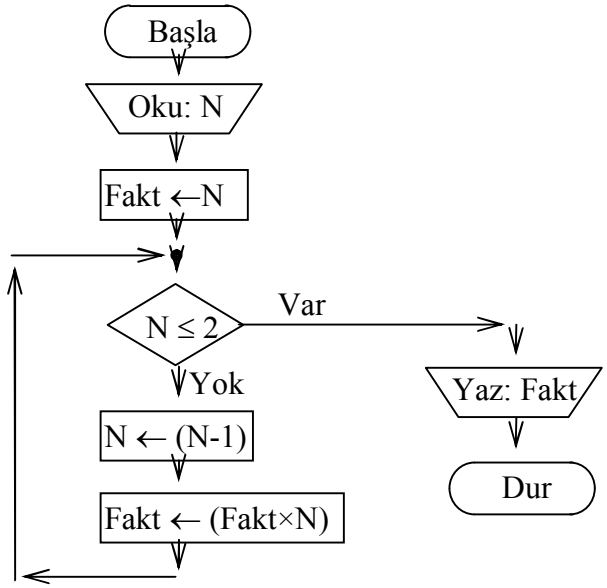
Genel olarak bir izlencenin hazırlanmasında şu adımlar izlenir:

a) Problemin açık, seçik ve tam olarak belirlenmesi.
b) Çözüm için nasıl bir yol izleneceğinin belirlenmesi.
c) İzlencenin mantıksal yapısının şekillendirilmesi. (Bunun için bir "akış çizelgesi"hazırlanır.)
d) Akış çizelgesinin izlenceye dönüştürülmesi.
e) İzlencenin bilgisayara uygun bir ortama yazılması (ör. disket veya sabit diskete bir dosyaya)
f) Yazılmış izlencenin "İş" in nasıl ele alınacağının işletim sistemine bildiren gerekli iş-denetim komutlarıyla beraber bilgi-işlem düzenine verilmesi
g) İzlence genel kullanıma sunulacaksa, bir kullanım kılavuzunun hazırlanması.

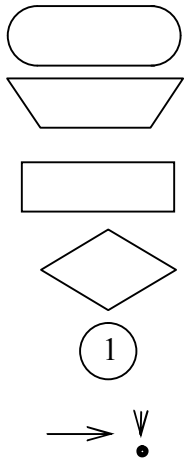
1.8. <u>Akış Çizelgeleri</u>

Bir izlencenin mantıksal yapısını şekillendirmek için akış çizelgelerinden yararlanılır. Akış çizelgeleri, izlenceler gibi katı yazım kurullarına tabi değildirler. Bu bakımdan izlencecinin, zihnini işin mantığına daha fazla verebilmesine imkan tanırlar.

Örnek: Bir akış çizelgesi



Akış çizelgeleri şu öğelerden oluşur:



: "Başla" veya "Dur" öbeği

: Giriş veya Çıkış işlemi öbeği

: İşlemsel bir öbek

: Karar öbeği

1 : Çizelgenin iki uzak noktasını bağlamak için işaret.

: Mantık akışını gösteren oklar.

Bir akış çizelgesi, seçilen izlenceleyiş dilinin düzeyinde olmalıdır. Yani akış çizelgelerindeki öbek içi işlemler, izlenceleyiş dilinde doğrudan karşılıkları bulunabilecek düzeyde olmalıdır. Yukarıda verilmiş olan örnekteki çizelge, PASCAL diline uygundur.

1.9. Bir İzlence Hazırlayış Örneği :

İstenen : Birden büyük değerler için N faktoriyeli hesaplayan bir izlence hazırlayınız.

Safhalar :

a) N faktoriyel, $N > 1$ için $N\times(N-1)\times...\times2$ şeklinde verilir. $N \leq 1$ için izlenceden ne istendiği ise belirtilmemiştir. Belirlemede eksiklik olmaması için bunu bizim belirlememiz gerekmektedir. Ancak bu belirleme, istenen kısmın dışında olacağı için serbestiyetimiz vardır. Yani $N \leq 1$ için sonucun klasik tanıma uygun olması veya anlamlı olması zorunluğu yoktur. Öyleyse $N \leq 1$ olursa N faktoriyel doğrudan N çıksın diyelim.

b) Verilen N ikiden büyükse, $N\times(N-1)\times...$ şeklinde 2 -ye kadar giden bir çarpım oluşturalım. Bu çarpım en son 2 ile de çarpılınca sonuç çıkacaktır. Eğer N ikiden büyük değilse, N faktoriyel doğrudan N olsun. ($N = 2$ ise N faktoriyel zaten 2 -dir.)

c) İzlenceye intikal ettireceğimiz bu yolun mantıksal yapısını şekillendiren akış çizelgesi zaten, bir örnek olarak yukarıda verilmiş olan akış çizelgesidir. "Başla" öbeği yolun

giriş noktasını işaret etmektedir. Yapılacak ilk iş, bir N değerini almaktır (oku:N). Sonra gelen öbekte, N değerinin (keyfi olarak) Fakt diye isimlendirilen bir değişkene aktarılacağı gösterilmektedir. Daha sonra gelinen, bir karar öbeğidir. Burada $N \leq 2$ durumu aranmaktadır. Bu durum var ise "var" çıkışı izlenerek N faktoriyelin değeri verilmekte, (yaz: fakt) ve durulmaktadır. Eğer $N \leq 2$ durumu yoksa, $N > 2$ demektir. Bu takdirde "yok" çıkışı izlenmekte ve önce N -nin değeri bir eksiltilmekte ($N \leftarrow (N-1)$) sonra da fakt×N değeri hesaplanıp, Fakt -ın yeni değeri olarak Fakt -a aktarılmaktadır. Bundan sonra tekrar karar öbeğine gidilmektedir. Böylece oluşan döngü, her seferinde N -nin değerini bir azaltmaktadır. Yukarıda (b) de sözü edilen çarpım Fakt ismi altında oluşmakta ve N -nin eksile eksile 2 -ye varıp varmadığı karar öbeğinde yoklanmaktadır.

1.10. İzlenceleyişte Dikkat Edilecek Noktalar.

İzlenceleyişte dikkat edilecek noktalar, teknolojinin ilerleyerek donanım kaynaklarının genişlemesi ve ucuzlamasıyla zamana göre değişime uğramıştır. Özellikle zamanımızda söz konusu olabilen çok büyük çapta yazılımlar, hazırlanmalarında titiz teknikler gerektirmektedirler. Bu arada dikkat edilmesi gerekli olan noktalara geçmeden önce bazı kavramları tanımlamakta yarar vardır.

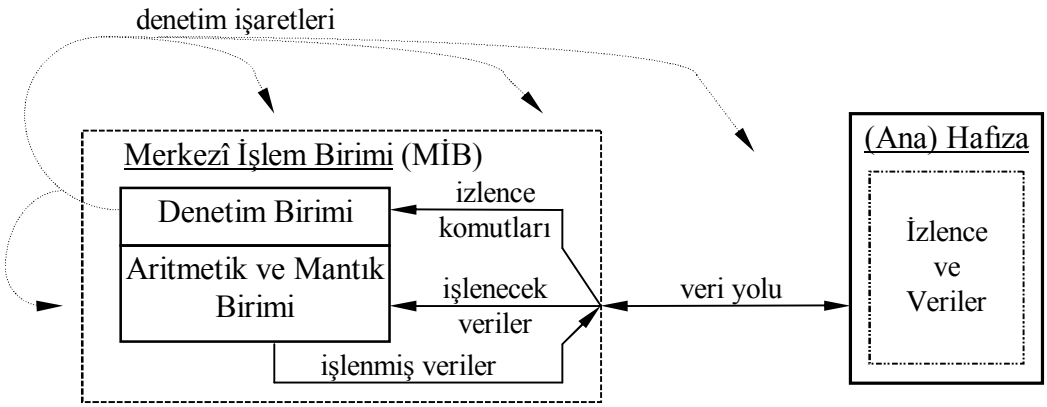
Taşınabilirliğin sağlanması konusunda yüksek düzeyli dillerin kullanımı kolaylık sağlamaktadır. Ancak, aşağı yukarı her yüksek düzeyli dil (bu arada PASCAL) için değişik bilgi-işlem düzenlerinde farklı lehçelere rastlanmaktadır. Lehçe farklılıkları, genellikle firmaların standart tanımları aşarak izlencecilere daha çok kolaylık tanıma çabalarından kaynaklanmaktadır. Taşınabilirliğe önem veren bir izlenceci, bu gibi kolaylıkları ret ederek kendisini standart anlatımlarla kısıtlamalıdır.

İzlenceleyişte en önemli sayabileceğimiz nokta, tertiptir. Tertip, izlenceleyişin belli bir disiplin altında yürütülmesiyle elde edilebilir. Bu, izlencecinin kendi kendisine uyguladığı bir disiplin olabileceği gibi, kullanılan izlenceleyiş dilinin yaptırımlarından kaynaklanan bir disiplin de olabilir. Tertipli bir izlence elde edebilmek için yapısal yaklaşım ve izlence içi açıklamalardan yararlanılabilir. Genellikle her izlenceleyiş dilinde, makineyi ilgilendirmeyip izlenceciye veya izlenceyi inceleyecek başka kişilere hitap edecek açıklamaların yapılabilmesi için imkan vardır. Bu imkandan en geniş ölçüde yararlanılarak izlencenin kendisini açıklar biçimde gerçekleştirilmesinde büyük yarar vardır. Tertip sayesinde bir izlencede yüksek güvenilirlik ve yaşatılabilirliğin sağlanması kolaylaşır.

Günümüz teknolojisi çerçevesinde insan emeğinin değeri oldukça yüksektir. Bu bakımdan her hangi bir izlencede ucuz hafıza ve zaman kazanma hesapları pahasına aşırı emek harcanmamalıdır. Bu tür emek harcamak çoğu kez izlence yapısını da çetrefilleştirerek mantık hatalarına yol açabilir. Tertipli ve dosdoğru bir mantığın ortaya konduğu izlenceler, aynı zamanda insan emeği açısından da verimli sonuçlar doğururlar.

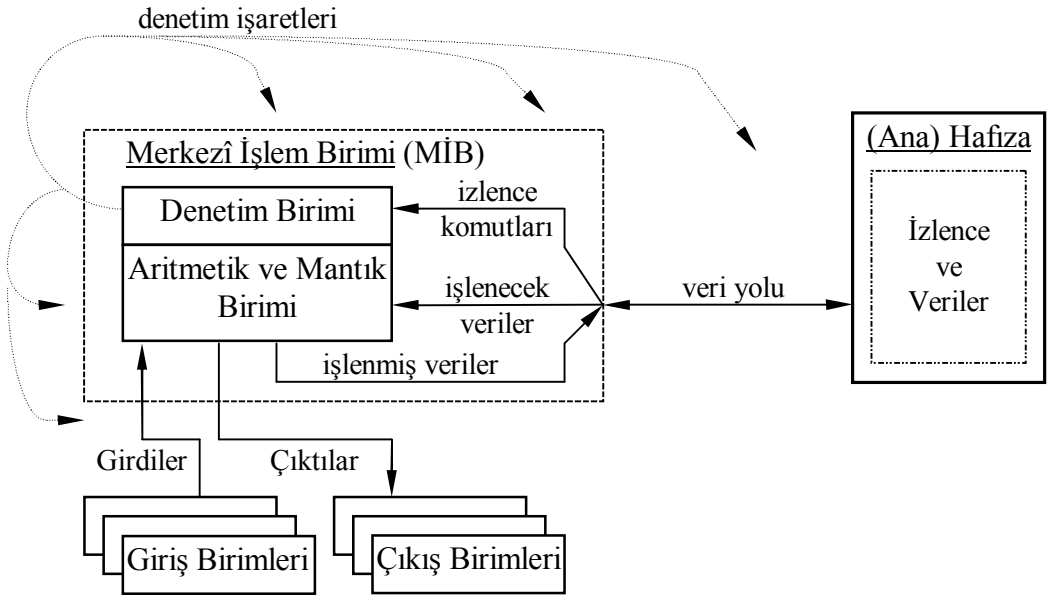
BİLGİSAYAR DONANIMINDA ESASLAR

Genel olarak basit bir bilgisayarın çekirdek yapısı, Şekilde görüldüğü gibidir.

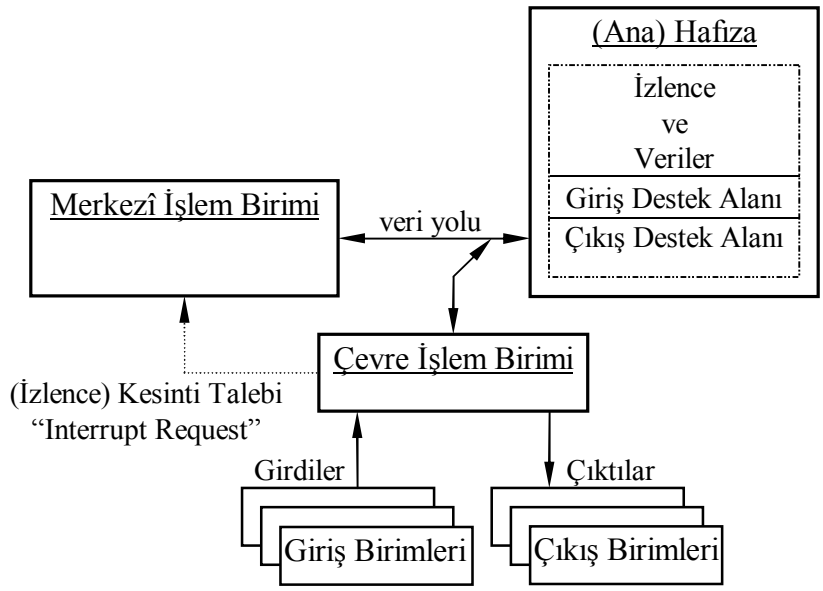


Şekil x: Bilgisayarın çekirdek yapısı

prog.:komut dizisi
fiziksel sıra genelde komutların ifa ediliş sırası (mantıksal) mantıksal/fiziksel gösterim???



Şekil x: Giriş-çıkış birimleriyle temel bağlantı.



Şekil x: Çevre İşlem Birimi ilavesiyle Merkezi İşlem Biriminin sadece hesap işlemlerine hasredilmesi.

- Özelleştirilmiş MİB: ÇİB