

MANTIK DEVRELERİ

İçindekiler

[SORU 1: Encoder \(Kodlayıcı\)](#)

[SORU 2: İleri ve geri zincirleme \(Forward and Backward Chaining\)](#)

[SORU 3: CDMA \(code division multiple access\)](#)

[SORU 4: Kuantum Kapıları \(Quantum Gates\)](#)

[SORU 5: Değil Kapısı \(not gate\)](#)

[SORU 6: Toffoli Kapısı \(Toffoli Gate\)](#)

[SORU 7: LFSR \(Linear Feedback Shift Register\)](#)

[SORU 8: Kaydırma Kayıtları \(Kaydırma Yazmaçları , Shift Registers\)](#)

[SORU 9: Turing Makinesi \(Turing Machine\)](#)

[SORU 10: Atomluluk \(Atomicity\)](#)

[SORU 11: Tehlike \(Hazard\)](#)

[SORU 12: Kayan Nokta Sayıları \(Floating Point Numbers\)](#)

[SORU 13: Çıkarıcı Devre \(Subtractor Circuit\)](#)

[SORU 14: İkillik Prensibi \(Duality Principle, İstaniyet\)](#)

[SORU 15: Kuantum İşleme \(Quantum Computing\)](#)

[SORU 16: Kubit \(Qubit\)](#)

[SORU 17: Doğrusal Ayrılabilirlik \(Linear Seperability\)](#)

[SORU 18: Yahut \(Özel Veya \(exclusive or, farklılık operatörü\)\)](#)

[SORU 19: CRC \(cyclic redundancy check, çevrimsel fazlalık sınaması\)](#)

[SORU 20: Sayıcı \(Counter\)](#)

[SORU 21: Sonlu Durum Makinası \(Finite State Machine, Finite State Automaton\)](#)

[SORU 22: flip flop \(flipflop\)](#)

[SORU 23: Salt okunur bellek \(read only memory , ROM\)](#)

[SORU 24: çoklayıcı \(multiplexer\)](#)

[SORU 25: kod çözücü \(decoder\)](#)

[SORU 26: tam toplayıcı \(full adder\)](#)

[SORU 27: yarım toplayıcı \(half adder\)](#)

[SORU 28: doğruluk çizelgesi \(truth table\)](#)

[SORU 29: karnaugh haritası \(karnaugh map\)](#)

[SORU 30: de morgan kuralı \(de morgan rule\)](#)

[SORU 31: veya kapısı \(or gate\)](#)

[SORU 32: Ve kapısı \(and gate\)](#)

[SORU 33: Önergeler \(kaziye\) Mantığı \(Propositional Logic\)](#)

[SORU 34: Bir tümleyeni](#)

[SORU 35: İki tümleyeni](#)

SORU 1: Encoder (Kodlayıcı)

Bu yazının amacı, bir mantıksal devre elemanı olan kolayıcının (encoder) çalışma mantığını ve tasarımını açıklamaktır.

Basit bir kodlayıcı, [kod çözücünün \(decoder\)](#) tersine üssel işlemi geri alır. Örneğin bir kod çözücünde, yapılan işlem 2^n şeklinde gelen girdinin (input) üstünü almaktır. 3×8 bir kod çözücünde, gelen 3 bitlik girdinin (input) değeri n olarak kabul edilirse, kod çözücü bu değere göre 8 farklı çıktıdan (output) bir tanesini seçer.

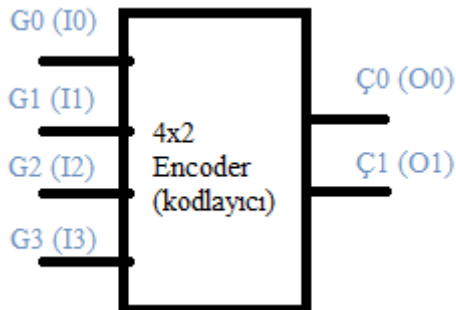
Kodlayıcı ise bu işlemin tam tersi yönde 8 farklı girdiden birisinden sinyal gelmesi halinde 3 çıktıdan (output) ilgili ihtimalleri işaretleyerek üst alma işleminin tersini (logaritma) yapar.

Örneğin aşağıda bir 4×2 kodlayıcının (encoder) [doğruluk tablosu \(truth table\)](#) verilmiştir:

I_3	I_2	I_1	I_0	O_1	O_0	V
0	0	0	0	x	x	0
0	0	0	1	0	0	1
0	0	1	0	0	1	1
0	1	0	0	1	0	1
1	0	0	0	1	1	1

Tabloda I ile ifade edilen kolonlar girdi (input) ve O ile ifade edilen kolonlar ise çıktı (output) değerlerdir. Örneğin 0100 değerinin 10'luk tabanda karşılığı 4 olarak yazılabilir. Bu değer tablodaki çıktı değeri (output) 10 olarak okunacaktır. 10 değeri ise 10'luk tabanda 2 olarak yazılabilir. Gerçekten de $\log_2 4 = 2$ olmaktadır ve kodlayıcının bir logaritma işlemi olduğu görülebilir.

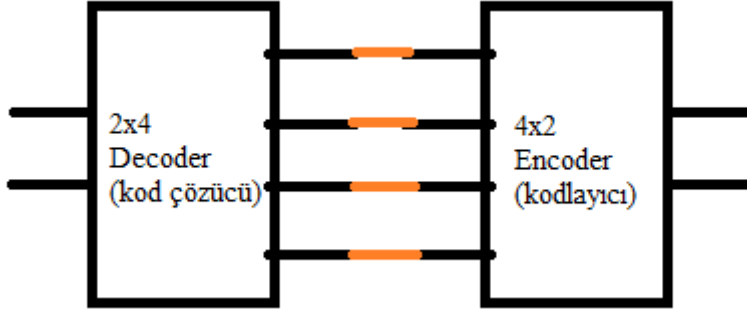
Yukarıdaki tabloyu gerçekleyen kodlayıcının genelde kullanılan blok çizimi aşağıdaki şekildedir:



Ayrıca doğruluk tablosunda görülebileceği üzere, V biti (valid bit, kabul edilebilir) kullanılarak tanımsız durumlar ortadan kaldırılabilir. Örneğin logaritmanın tanımından bilindiği üzere 0'ın logaritması tanımsızdır. Bu durumda bütün girdi (input) bitlerinin 0 olması

durumunda çıktı belirsiz olacaktır. İşte bu belirsizlik durumunda çıktının kabul edilemez (invalid) olduğunu ifade için V biti 0 değerinde verilebilir.

Şayet bir [kod çözücü \(decoder\)](#) ile bir kodlayıcı (encoder) arka arkaya bağlanırsa, sistemin girdi değeri, çıktı değeri olarak okunur.



Yukarıdaki devrede, soldan verilen girdi sağdan değişmeden okunurken devre tam tersine çevrilerek, sağdan bir girdi verilmesi halinde de soldan okunacaktır.

Kodlayıcı devresini, kapılar kullanarak yapmak da mümkündür. Örnek bir tasarım aşağıda verilmiştir:

I_3	I_2	I_1	I_0	O_1	O_0	V
0	0	0	0	x	x	0
0	0	0	1	0	0	1
0	0	1	0	0	1	1
0	1	0	0	1	0	1
1	0	0	0	1	1	1

Doğruluk tablosunun [karnaugh haritasını \(karnaugh map\)](#) çizerek:

O_0 için

	$I_1I_0=00$	$I_1I_0=01$	$I_1I_0=11$	$I_1I_0=10$
$I_3I_2=00$	X	0	X	1
$I_3I_2=01$	0	X	X	X
$I_3I_2=11$	X	X	X	X
$I_3I_2=10$	1	X	X	X

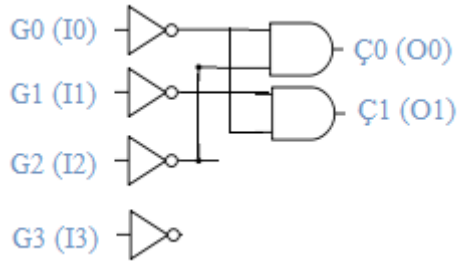
Yukarıdaki haritada, sonucu etkilemeyen (kodlayıcının çalışması belirsiz ve hiçbir şekilde girdi olarak gelemeyecek değerler) X ile ifade edilmiştir. Bu tip kodlayıcılara (encoder) özel olarak öncelik kodlayıcısı (priority encoder) ismi de verilmektedir. Bu haritada X değerleri 1 veya 0 olarak kabul edilebilir. O halde yukarıdaki tabloda mavi ile işaretlenmiş olan 4 ihtimal tek başına alınarak O_0 için $I_0'I_2'$ sonucuna varılabilir.

O_1 için

	$I_1I_0 = 00$	$I_1I_0 = 01$	$I_1I_0 = 11$	$I_1I_0 = 10$
$I_3I_2 = 00$	X	0	X	0
$I_3I_2 = 01$	1	X	X	X
$I_3I_2 = 11$	X	X	X	X
$I_3I_2 = 10$	1	X	X	X

Yukarıdaki tabloda da benzer şekilde O_1 için $I_1'I_0'$ sonucuna varılabilir.

Yukarıdaki sonuçlara göre bir kodlayıcıyı (encoder) aşağıdaki şekilde çizebiliriz:



SORU 2: İleri ve geri zincirleme (Forward and Backward Chaining)

Bu yazının amacı, bilgisayar bilimlerinde, özellikle de mantıksal sistemlerin ispatında kullanılan ileri zincirleme ve geri zincirleme yöntemlerini açıklamaktır.

Yöntemin çalışması oldukça basittir. Öncelikle problem, mantık düzleminde modellenir. Buradaki mantık sistemi sonlu ispatı olan herhangi bir system olabilir. Örneğin birinci dereceden mantık (first order logic) veya daha özel olarak boole cebiri kullanılabilir.

Modelleme aşamasının ardından problemin çözümüne geçilir. İşte tam bu noktada ileri zincirleme (forward chaining) veya geri zincirleme (backward chaining) yöntemlerinden birisi seçilebilir.

Örneğin aşağıdaki mantıksal sistemi ve şekli ele alalım:

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

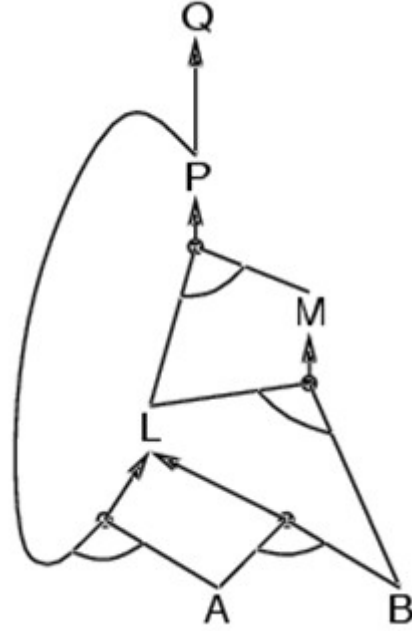
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

A

B

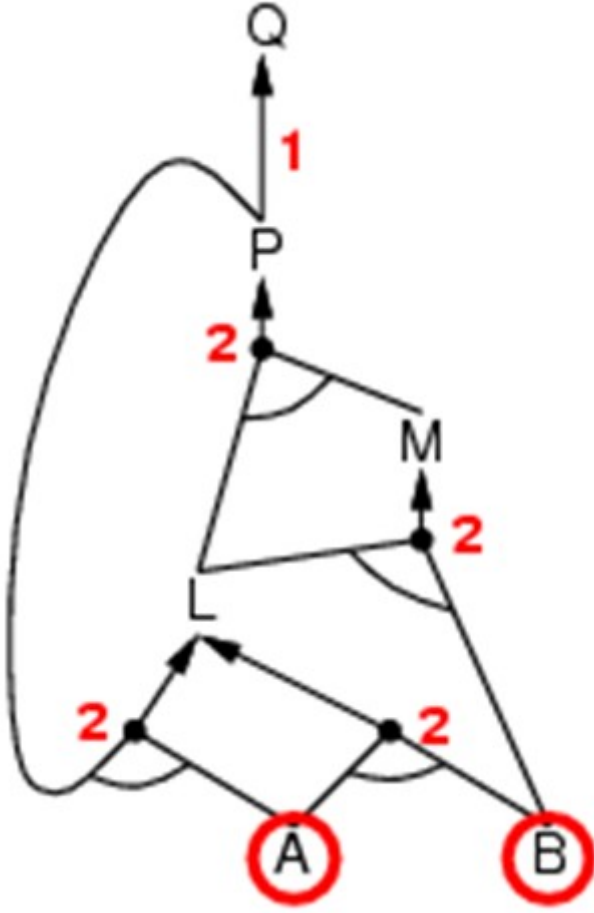


Sistemde görüldüğü üzere bazı mantıksal dizilimler verilmiş ve son iki satırda A ve B önermelerinin (kaziye) doğru olduğu belirtilmiştir.

Buna göre sağdaki çizim, hangi durumlarda, hangi diğer durumların doğru olacağını bu mantıksal sistemden çıkarır. Örneğin $p \Rightarrow q$ ifadesi, çizimin en tepesinde gösterilmiş ve p önermesinin (predicate, kaziyesinin) doğruluğu halinde q önermesinin de (kaziyesinin de) doğru olacağını ifade etmektedir.

Benzer şekilde, L önermesinin (kaziyesinin) doğruluğu A ve B önermelerine bağlı olduğu gibi, A ve P önermelerinin doğruluğuna da bağlanmıştır. Bu iki sistemden birisinin doğru olması sonucun doğruluğunu sağlar.

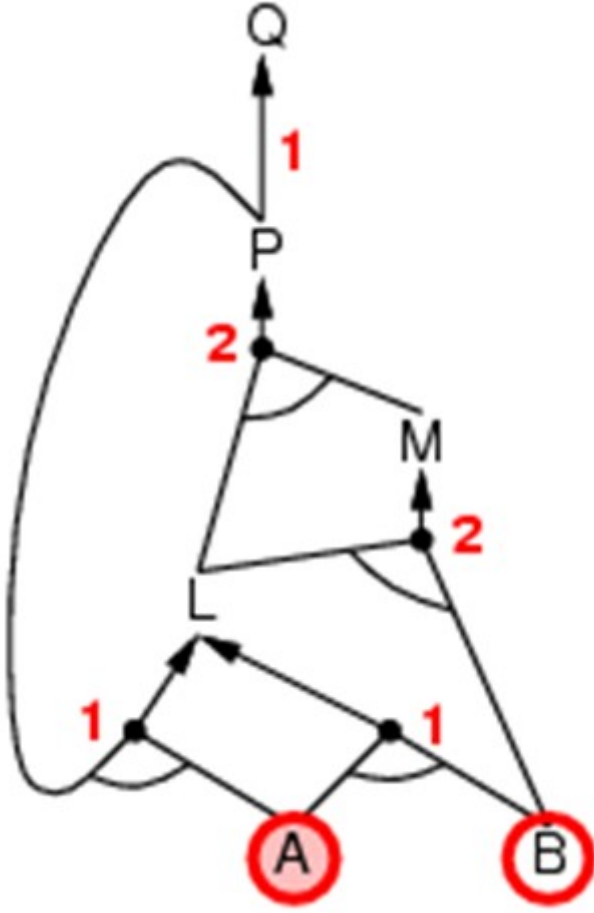
Şimdi şekilde gösterilen sistemi ileri zincirleme (forward chaining) yöntemi ile çözelim. Öncelikle sistemdeki bütün doğruluk şartlarını sayısal olarak ifade ediyoruz:



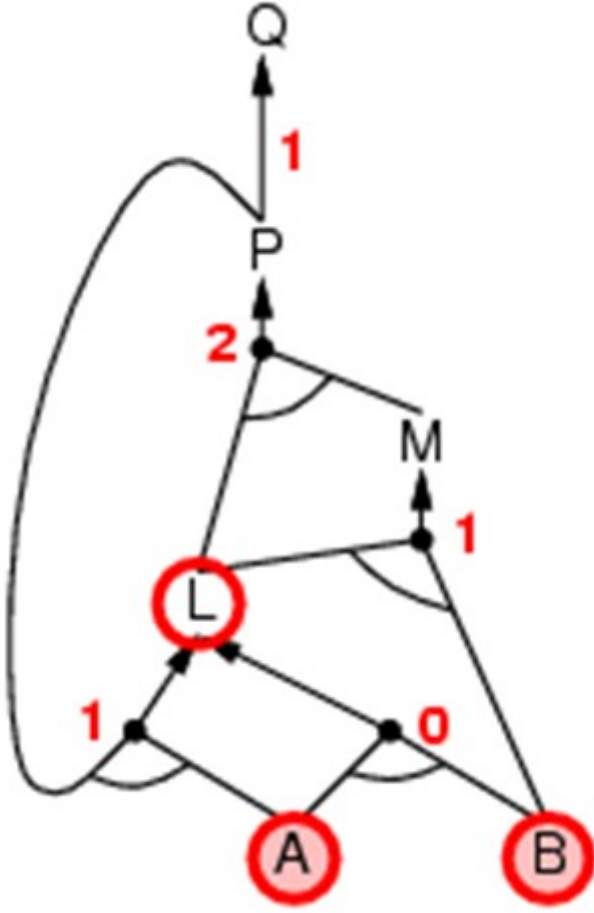
Şekilde görüldüğü üzere bütün doğruluk şartları birer sayı ile ifade edilmiştir. Söz gelimi, M önermesinin doğruluğu L ve B önermesi gibi 2 önermenin doğruluğunu gerektirir. Bu yüzden M birleşiminde 2 sayısı bulunur. Benzer şekilde Q önermesinde bulunan 1 sayısı, sadece P önermesinin doğruluğunun yeterli olduğunu ifade etmektedir.

Şimdi ileri zincirleme yöntemini kullanarak sistemin doğru olduğu verilen A ve B önermelerinden itibaren çözümünü izleyelim.

Öncelikle A ve B'ye komşu olan düğümlerdeki değerleri 1'er azaltıyoruz:



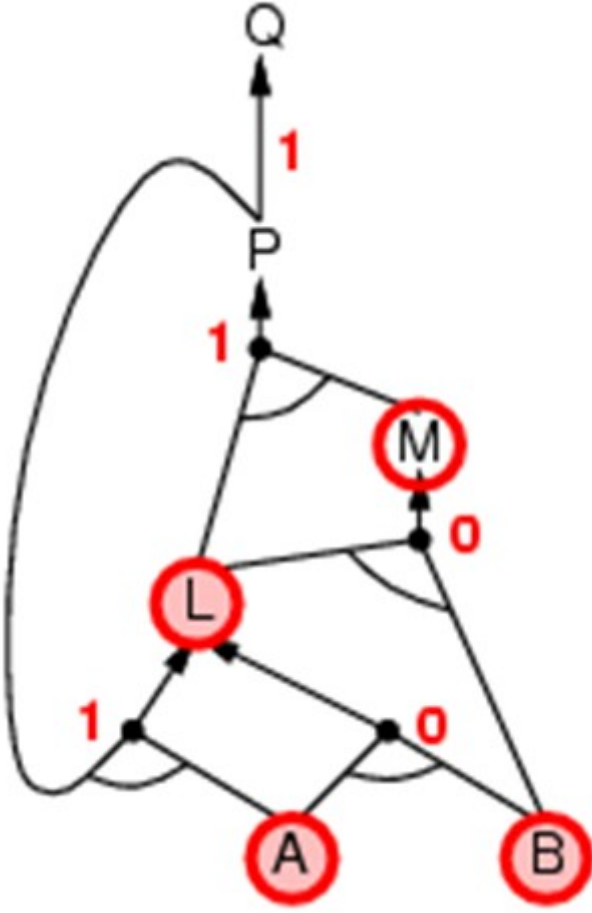
Yukarıdaki şekilde ileri zincirleme işlemi (forward chaining) A önermesi için çalıştırılmış olup A'nın komşularını 1 azaltmıştır. Sırada B önermesi var ve onu da çalıştıralım:



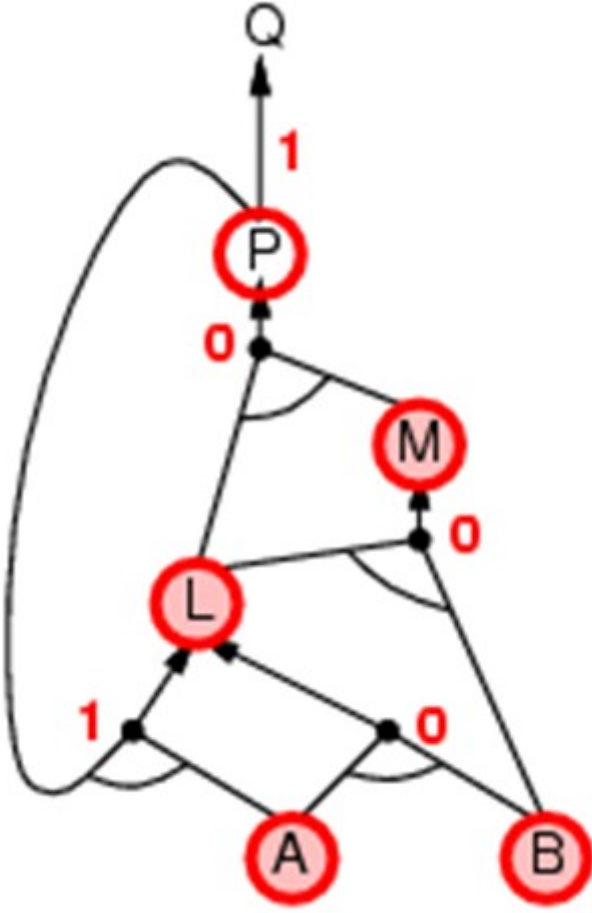
Görüldüğü üzere B'nin komşuları da 1 azaldığında 0 değerine sahip bir düğüm elde ettik. Bu durumda L'nin doğru olduğunu söyleyebiliriz çünkü L'nin doğru olması için gereken 2 değer de sağlandı. Yani mantıksal sistemimizde bulunan

$$A \blacktriangleleft B \blacktriangleright L$$

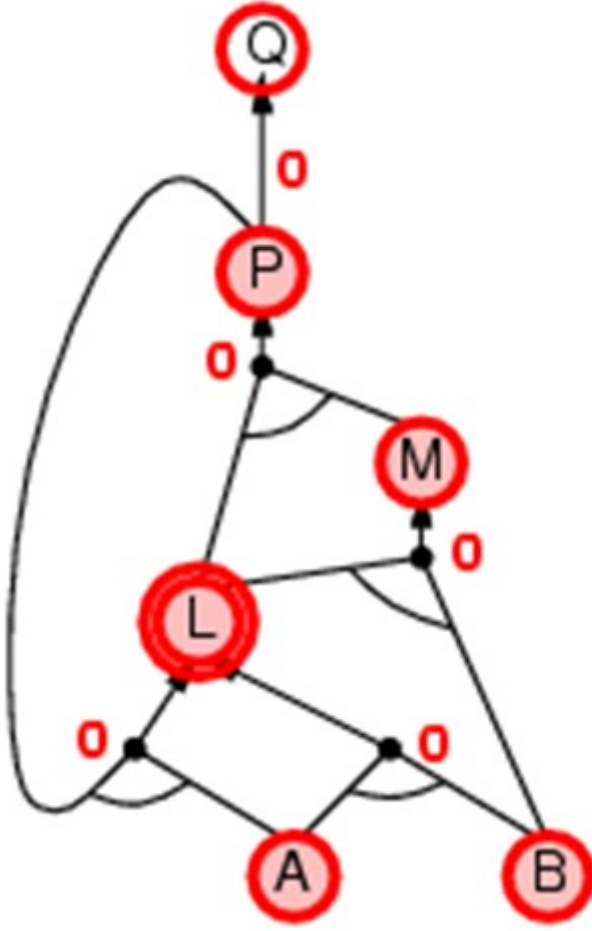
Satırını sağlamış olduk. Buradan doğruluğunu bulduğumuz L önermesinin komşularını 1 azaltıyoruz:



L'nin komşularının 1 azalması sonucunda M'nin değeri 0'a inmiş oluyor ve artık M için de doğru diyebiliyoruz. Şimdi M'nin komşularını 1 azaltalım:



Artık P için doğru sonucuna ulaştık ve P'nin iki komşusunda değerini 1 azaltarak sonucu buluyoruz:

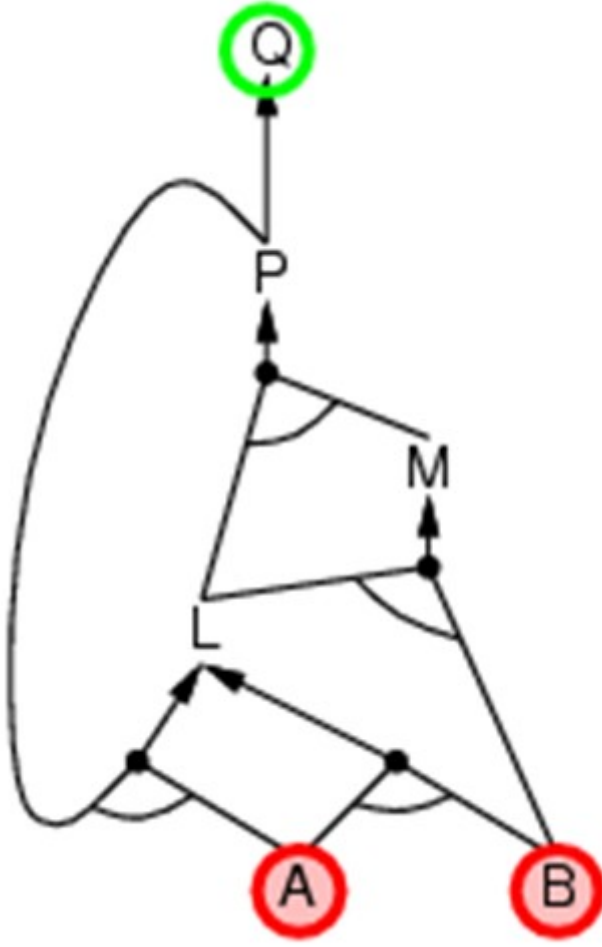


Görüldüğü üzere zaten doğru olduğunu bildiğimiz L için tekrar doğru sonucunu bulduk ve ilave olarak Q için de doğru sonucunu bulduk.

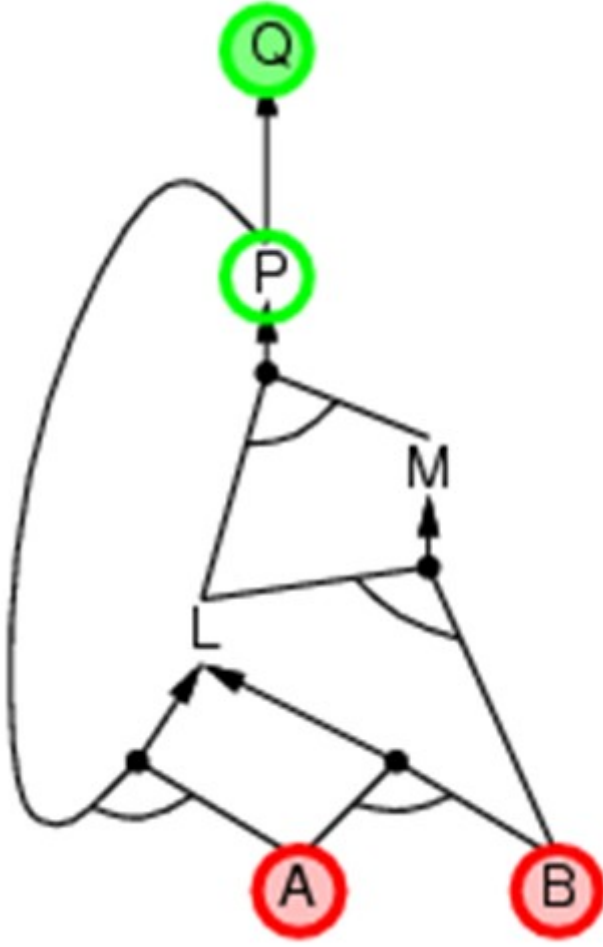
Demek ki ilk sistem bize verildiğinde, Q'nun değeri sorulsaydı, doğru olduğunu söyleyebilirdik, ancak bunu bilgisayarın bulması için yukarıda adım adım anlatılan aşamaların tamamlanması gerekmektedir.

Geri Zincirleme (Backward Chaining)

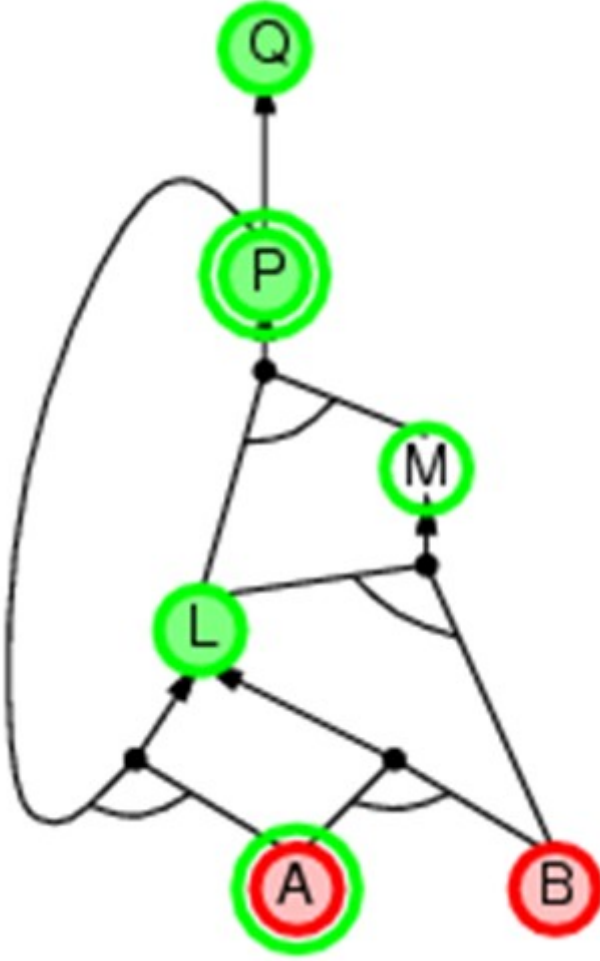
Gelelim aynı amaç için kullanılan, yani bir mantıksal sistemi çözmek için kullanılan geri zincirleme yöntemine. İleri zincirleme yöntemine çok benzer olarak yine bir mantıksal sistem, bir şekil üzerinde gösterilebilir:



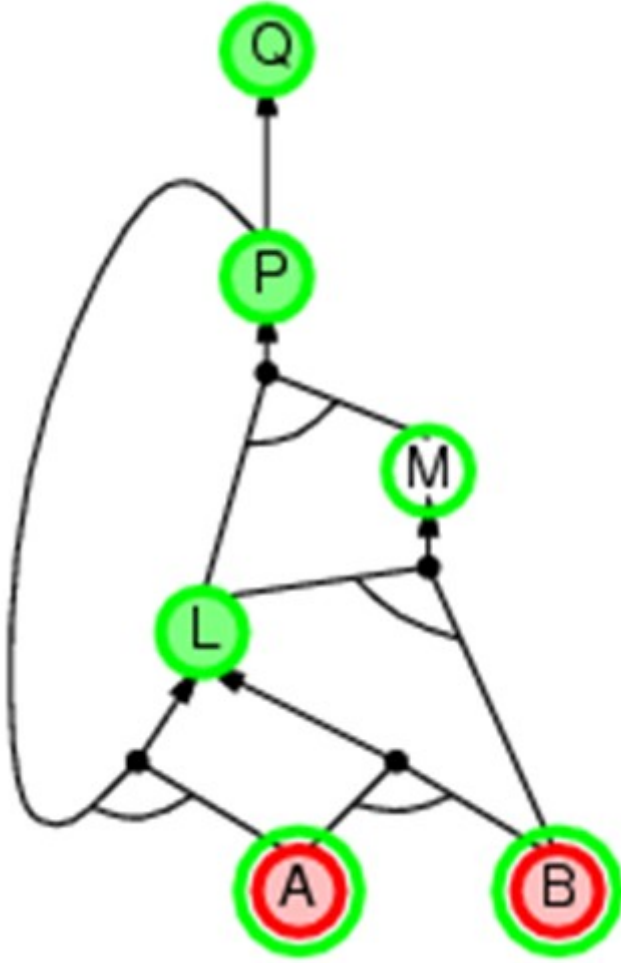
Sistemde diyelim ki Q'nun deęerini merak ediyor olalım. Bilgisayar algoritması, bu defa Q'nun deęerinin doęruluęunun P'nin deęerinin doęruluęuna baęlı olduęunu çözererek işe başlayacaktır. Aslında geri zincirlemede kullanılan yaklaşım, tam olarak ileri zincirlemenin tersidir. İleri zincirlemede, doęruluęunu bildiğimiz önermelerden başlanırken, geri zincirlemede, doęruluęunu aradığımız önermelerden başlıyoruz. Burada doęruluęunu aradığımız önerme Q olduęuna göre, Q'dan başlayarak sistemi dolaşacağız. İlk adımda Q'nun doęruluęu, P'nin doęruluęuna baęlıdır, o halde Q yerine P doęru mudur diye sistemi çözmeye çalışırız:



P'nin doğruluğu, şekilde de görüldüğü üzere, L ve M'ye bağlıdır ve artık L ve M doğru mudur diye sorarız. Bunlardan birisinin yanlış olması halinde sonuç yanlış veya ikisinin de doğru olması halinde sonuç doğru olacaktır. Burada sonuç ile kastedilen P ve dolayısıyla Q'dur. Dikkat edilirse artık L ve M'ye bakarak Q'nun değerini tahmin edebiliyoruz. Devam edelim:



L'nin doğruluğuna bakıldığında P ve A bulunmakta, aslında bu sorunun cevabını P'nin değerini bilmediğimiz için veremeyiz. Ancak burada bir tehlike bizi bekliyor, şayet doğruluğunu araştırmak için DFS (depth first search, derin öncelikli arama) benzeri bir algoritma ile ağacı (veya şekli (graph)) dolaşıyorsak, bu durumda bir sonsuz döngüye (fasit daire) girme ihtimalimiz bulunuyor. Bunu engellemek için diyelim ki derinliği sabitledik ve L'nin doğruluğu için A ve B ikilisine bakmaya karar verdik:



Sonuçta A ve B doğru ise L doğru demektir. O halde L doğru mu sorusunu sormayı bırakıyor ve M doğru mu A ve B doğru mu sorularını arayarak sistemi çözmeye devam ediyoruz. M'nin doğruluğu ise L ve B'ye dayanmakta, o halde bir kere daha L'nin doğruluğunu sorguluyor ve yukarıda anlatıldığı üzere bir kere daha A ve B'nin doğruluğunu sorguluyoruz. Neticede sorumuz basitçe A ve B doğru mudur şeklinde oluyor.

Verilen mantıksal sistemden de bildiğimiz üzere A ve B doğrudur, o halde Q da doğrudur diyebiliriz, çünkü sistemi buraya kadar adım adım çözdük ve neticede Q'nun doğruluğunu sorgulamanın A ve B'nin doğruluğunu sorgulamak olduğunu gördük.

Geri zincirleme (backward chaining) yaklaşımında istenirse buradan geriye dönülerek bütün sistemdeki önermelerin durumları doğru veya yanlış olarak işaretlenebilir. Ancak geri zincirleme algoritması, bu aşamada aranan Q önermesinin sonucunu bularak durabilir de. Bu iki yaklaşım arasındaki fark aslında CPS (call by passing style) ile birikimsel tarz (accumulation style) arasındaki fark gibidir.

İki yöntemde de sonuç doğru bir şekilde bulunur. Belki ufak bir fark olarak dikkat edilmesi gereken, geri zincirlemede, özel olarak aranan bir önermenin sonucuna konsantre olmamız, buna bağlı olarak da bazı büyük sistemlerde, sistemin sadece belirli bir kısmını çözüyor olmamız görülebilir. Buna mukabil, ileri zincirleme yaklaşımında, sistemin tamamı çözülmektedir.

SORU 3: CDMA (code division multiple access)

Bilgisayar bilimlerinde, özellikle ağ (network) konusunda geçen ve bir ortamı, birden fazla veri kanalının iletişimi için kullanılan yöntemlerden birisidir. Literatürde sıkça geçen diğer çok kanallı veri iletişim yöntemleri, [TDMA \(time division multiple access, zaman paylaşımli çoklu erişim\)](#) ve [FDMA \(frequency division multiple access, frekans paylaşımli çoklu erişim\)](#) yöntemleridir.

CDMA yöntemini bu diğer meşhur iki yöntem ile karşılaştırmak için genelde şu şekilde bir örnek verilir. Örneğin bir odada birden çok kişinin konuşarak haberleştiğini düşünelim. TDM yaklaşımında, kişiler sırayla ve teker teker konuşmakta, ilgili alıcı konuşan kişinin mesajını almaktadır. FDM yaklaşımında, kişiler farklı ses tonları ile konuşmakta ve dolayısıyla alıcı olan kişi, ilgili ses tonuna dikkatini vererek iletilen mesajı almaktadır. CDMA yaklaşımında ise, kişiler farklı lisanlarda konuşmakta, dolayısıyla o lisanı bilen kişiler tarafından algılanmakta, diğer kişiler tarafından iletilen veri gürültü olarak algılanıp dikkate alınmamaktadır.

Örnek

Konuyu bir örnek üzerinden açıklamaya çalışalım. Örneğin 4 farklı veri kanalı üzerinden veri akmakta olsun ve bunları CDMA yöntemi ile tek bir kanaldan taşımak isteyelim.

- V1: 1101
- V2: 0010
- V3: 1010
- V4: 0011

Yukarıdaki şekilde verilen 4 farklı verinin CDMA ile nasıl taşındığını anlatalım. Verileri ilk adımda farklı frekans değerine sahip işaretler ile kodluyoruz (code). Örneğimizde kullanacağımız 4 farklı kodumuz aşağıdaki şekilde olsun:

- K1: 1111
- K2: 1010
- K3: 1100
- K4: 1001

Verilerin, kodlar tarafından işlenebilmesi için ve 4 farklı verimiz olduğu için, verilerin genliğini 4 misli şeklinde düünebiliriz. Buna göre örnek olarak son veri için kodlamayı anlatalım:

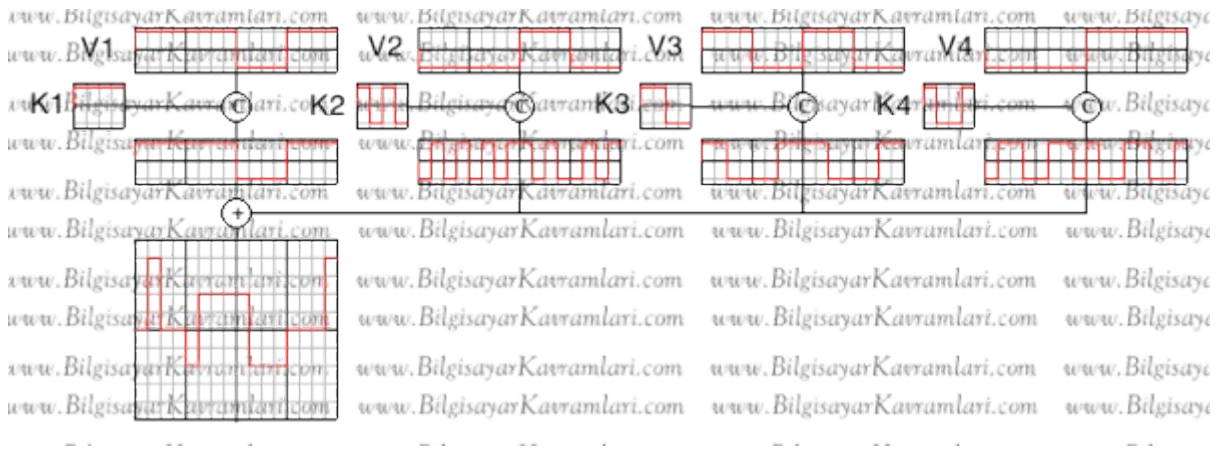
V4 : 0000 0000 1111 1111 (Gösterim için 0011 verisinin her elemanını 4 kere tekrarladım.)

K(V4,K4) : 0110 0110 1001 1001 (V4'ün, K4 ile kodlanması sonucunda, V4 üzerindeki 1 değerleri için K4'ün kendisi, V4 üzerindeki 0 değerleri için ise K4'ün tersi gelmektedir. Daha basit anlamda her V4 dörtlüsü (uzun şekilde yazılmış halini düşünün) ile K4 değerlerinin [özel veyasının \(XOR\)](#) tersi alınır !((0000 0000 1111 1111) XOR (1001 1001 1001 1001)) şeklinde)

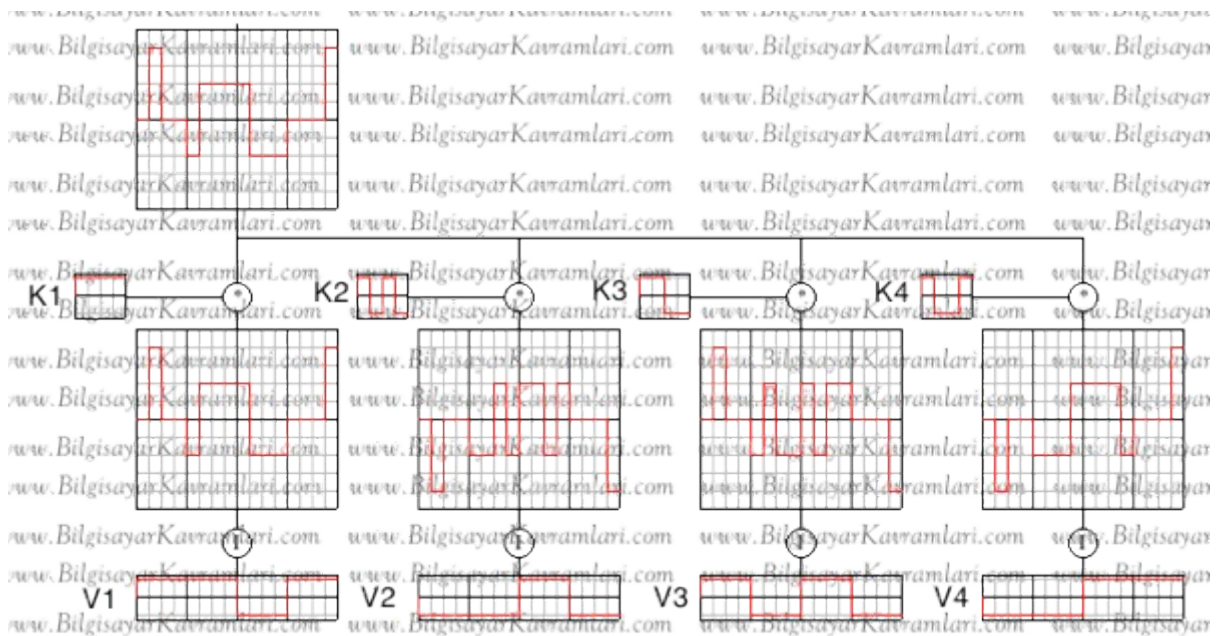
Sonuçta 4 farklı veri ve 4 farklı kodlama için aşağıdaki sonuçlara ulaşılır:

- $K(V1, K1) : 1111 \ 1111 \ 0000 \ 1111$
- $K(V2, K2) : 0101 \ 0101 \ 1010 \ 0101$
- $K(V3, K3) : 1100 \ 0011 \ 1100 \ 0011$
- $K(V4, K4) : 0110 \ 0110 \ 1001 \ 1001$

CDMA algoritmamızda, son adım olarak yukarıdaki değerleri topluyoruz. Toplamın ve yukarıdaki işlemlerin görsel olarak ifadesi aşağıdaki şekildedir:



Yukarıdaki toplama işlemi sonucunda elde edilen verilerin, her birisinin farklı alıcılar tarafından alınmak istediğini düşünelim. Bu durumda her alıcı, almak istediği göndericinin kodlama değerini kendisinde ayarlayacak ve yukarıda elde edilen sonuç verisini kendisinde işleyecektir. Bu durum aşağıdaki şekilde gösterilmektedir:



Yukarıda görüldüğü üzere her kodlama değeri sonucunda açılan veri, orjinal olarak kodlanan ilgili veridir. Örneğin K1 kodlamasından açılan veri V1 olarak bulunmuştur. Bu işlem diğer kodlamaları engellememektedir.

CDMA yöntemi, günümüzde de kullanılan UMTS teknolojisinin temelini oluşturur. UMTS (universal mobile telecommunication system, evrensel hareketli telekomünikasyon sistemi) teknolojisi, CDMA2000 teknolojisinden sonra (IMT Multi Carrier, inter mobile telecommunications, hareketli telekomünikasyonlar arası çoklu taşıyıcı olarak da bilinir) geliştirilen ve CDMA 2000 teknolojisi ile rekabeti amaçlayan bir teknolojidir. CDMA2000 de, UMTS'in temeli olan W-CDMA de birer 3G teknolojisidir ve cep telefonlarının aynı anda iletişimi için kullanılmaktadır.

SORU 4: Kuantum Kapıları (Quantum Gates)

Kuantum kapıları, mantıksal devre tasarımında bulunan klasik kapılara alternatiftir. Amaç, elektronik devrelerin karar mekanizmasında quantum teknolojisini kullanmaktır.

Klasik kapılarda bulunan ve bitlere göre karar vermeye yarayan mekanizmadan farklı olarak kuantum kapılarında, [kubitler \(qubits\)](#) üzerinden karar verilir. Kuantum kapılarının bir özelliği, geri döndürülebilir olmalarıdır (reversible), yani bir girdi için elde edilen sonuç, sonuçtan girdi olarak verildiğinde, girdi geri elde edilebilir.

Bir mantıksal kapının geri döndürülebilir olması, kapının girdisinden elde edilen çıktının tekrar girdi olması halinde, ilk girdinin geri elde edilebilmesidir. Bu karmaşık cümle ile anlatılmak istenen örneğin L kapısı için $L(x) = y$ gibi bir sonuç alınıyorsa, bu kapının tersi olan L' için $L'(y) = x$ sonucunun alınması beklenir. Veya kapının kendisinin ters olması halinde de $L(x) = y$ ve $L(y) = x$ şartlarının aynı anda sağlanması beklenir.

Örneğin klasik değil kapısı (not gate) geri döndürülebilir kapıdır (reversible). Bunu [doğruluk çizelgesine \(truth table\)](#) bakarak kolayca görebiliriz.

Girdi	Çıktı
1	0
0	1

Görüldüğü üzere $L(1) = 0$ ve $L(0) = 1$ olmakta, dolayısıyla tersi alınabilir bir kapı olmaktadır.

Buna karşılık, geri döndürülebilirlik (reversible) konusunun daha iyi anlaşılabilmesi için, geri döndürülemez bir kapı olan veya kapısını inceleyelim.

Girdi	Çıktı
00	0
01	1
10	1
11	1

Yukarıdaki [doğruluk çizelgesinde \(truth table\)](#) görüldüğü üzere, herhangi bir çıktının, girdiye verilmesi durumunda, girdinin geri elde edilmesi mümkün değildir. Örneğin $L(10) = 1$ olmakta ama $L(1) = 10$ olmamaktadır.

Aynı zamanda herhangi bir L' devresi de yukarıdaki tablonun tersini üretemez. Bunun sebebi, 1 çıktısının 01, 10 veya 11 şeklinde geri döndürülme ihtimali olduğu ve 1 çıktısı alındıktan sonra, orijinal girdinin ne olduğunun tahmininin imkânsız olduğudur.

Ve kapısı örneğini ele alarak, bir kapının geri döndürülebilir olması için giriş ve çıkış bitlerinin sayısının aynı olması gerektiğini tahmin edebilirsiniz. Aslında bu durum basitçe [güvercin yuvası kaidesi \(pigeonhole principle\)](#) ile açıklanabilir ve evet bir kapının geri döndürülebilir olması için giriş biti sayısı ile çıkış biti sayısı eşit olmalıdır.

Şayet giriş bitlerinin sayısı ile çıkış bitlerinin sayısı eşit ise, kapının karakterini, yukarıdaki örneklerde olduğu gibi doğruluk çizelgesi (truth table) şeklinde klasik gösterimden farklı olarak gösterebiliriz. Aslında kuantum kapıları (quantum Gates) için vaz geçilmez olan bu gösterim matris gösterimidir.

Örneğin [değil kapısını \(not gate\)](#) ele alalım ve matriste göstermeye çalışalım.

	0	1
0	0	1
1	1	0

Yukarıdaki matris, okunması kolay olsun diye bir satır (en üstteki) ve bir sütun (en soldaki) eklenerek verilmiştir. Bu matriste, satırlar, girdiyi, sütunlar ise çıktıyı tutmaktadır. Yani tablomuzu aşağıdaki şekilde yorumlayabiliriz

	0	1
0	0 girdisi için, 0 çıktısı alınabilir mi?	0 girdisi için, 1 çıktısı alınabilir mi?
1	1 girdisi için 0 çıktısı alınabilir mi?	1 girdisi için 1 çıktısı alınabilir mi?

Yukarıdaki bu sorulara evet veya hayır cevaplarını vererek evet için 1 ve hayır için 0 yerleştiriyoruz. Örneğin değil kapısı (not gate) 0 için 1 sonucu verir ve 0 için 0 sonucu vermez. Dolayısıyla yukarıdaki doğruluk çizelgesinin matris gösterimini aşağıdaki şekilde yapmak yeterlidir.

0	1
1	0

Yukarıdaki bu matrise bakıldığında zaman, bu matrisin [doğruluk çizelgesi \(truth table\)](#) kolaylıkla anlaşılabilir.

Matris gösteriminin kuantum kapıları için kullanılması durumunda, aslında qubit değerlerinin matrise yerleştirilmesinden bahsediliyor demektir.

Örneğin, $\alpha|0\rangle + \beta|1\rangle$ şeklinde yazılan bir kubit gösterimini vektör olarak modellemek istersek

α
β

Şeklinde bir vektör elde edebiliriz. Bu vektörü değil kapısı (not gate) için girdi ve çıktı olarak modellediğimizde, bir qubit için durum aşağıdaki şekilde olur:

$$X \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \alpha \end{bmatrix}$$

Görüldüğü üzere, kubitin tersi alınmıştır. Burada dikkat edilecek bir husus, matriste kullanılan α ve β değerlerinin karmaşık sayılar (complex numbers) olduğudur.

Kuantum Kapılarının bir özelliği, bu kapılarda kullanılan matrisin, [vahid masfuf \(uniter matrix\)](#) olmasıdır.

Çok Kullanılan Kuantum Kapıları

Bu bölümde, kuantum kapılarından çok kullanılanlarını anlatacağız. Teorik olarak sonsuz sayıda kuantum kapısı üretilebilir. Ancak buradaki amaç özellikleri bakımından önemli görülen ve literatürde sıkça rastlananları açıklamaktır.

Hadamard Kapısı

Hadamard kapıları, tek kubitli bir sistemde, aşağıdaki dönüşümleri yaparlar.

$$|0\rangle \text{ deęerini } \frac{|0\rangle + |1\rangle}{\sqrt{2}} \text{ olarak}$$

$$|1\rangle \text{ deęerini ise } \frac{|0\rangle - |1\rangle}{\sqrt{2}} \text{ olarak dntrr.}$$

Bu durumda, hadamard kapısının matrisi aşağıdaki şekilde olacaktır:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Hadamard kapılarının ismi, bu kapılar için kullanılan matrisin bir [hadamard matrisi \(hadamard matrix\)](#) olmasından gelmektedir.

Aslında hadamard matrislerini, [deęil kapısının \(not gate\)](#) karekk olarak dnmek de mmkndr. Grldę zere, elde edilen sonu bir [vahid masfuf \(uniter matrix\)](#)

Pauli X kapısı

Pauli X kapıları, kalsik deęil kapısının (not gate), kuantum için uyarlanmış halidir. Yani yazının bařında anlatılan ve giriř tersine dndrmeye yarayan kapılar olarak dnlebilir. Bu durumda matrisi aşağıdaki şekilde olacaktır.

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Aslında bu kapının özelliği [Bloch Küresini \(Bloch Sphere\)](#) X eksenini etrafında pi radyan kadar döndürmesi ve $|0\rangle$ değerini $|1\rangle$ ve $|1\rangle$ değerini $|0\rangle$ yapmasıdır.

Pauli Y kapısı

Pauli X kapısına benzer olarak bu kapı da [Bloch Küresi \(Bloch Sphere\)](#) üzerinde döndürme işlemi yapmaktadır. Ancak bir önceki kapıdan farklı olarak bu defa Y eksenini üzerinde döndürme işlemi yapılır.

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$

Pauli Z kapısı

Pauli X ve Y kapılarına benzer şekilde [Bloch Küresi](#) üzerinde döndürme işlemi yapılır. Bu defa isminden de anlaşılacağı üzere döndürme işlemi Z eksenini üzerinde olur.

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Faz kaydırma kapısı (Phase shift gate)

Bu kapının özelliği, 00, 01 ve 10 için değişiklik yapmamak ama 11 durumu için $|1\rangle$ girdisinin $e^{i\theta}|1\rangle$ girdisine dönüştürmesidir. Yani $|1\rangle$ için, θ derece döndürme işlemi yapılmaktadır.

$$R(\theta) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix}$$

SORU 5: Değil Kapısı (not gate)

Mantıksal devre tasarımında kullanılan bir kapı örneğidir. Basitçe bir değerini tersini almaya yarar.

Değil kapısının [doğruluk çizelgesine \(truth table\)](#) aşağıdaki şekildedir

Girdi	Çıktı
1	0
0	1

Görüldüğü üzere $L(1) = 0$ ve $L(0)=1$ olmakta, dolayısıyla giren değerin tersi döndürülmektedir.

Ayrıca yukarıdaki doğruluk çizelgesini matriste gösterebiliriz:

	0	1
0	0	1
1	1	0

Yukarıdaki matris, okunması kolay olsun diye bir satır (en üstteki) ve bir sütun (en soldaki) eklenerek verilmiştir. Bu matriste, satırlar, girdiyi, sütunlar ise çıktıyı tutmaktadır. Yani tablomuzu aşağıdaki şekilde yorumlayabiliriz

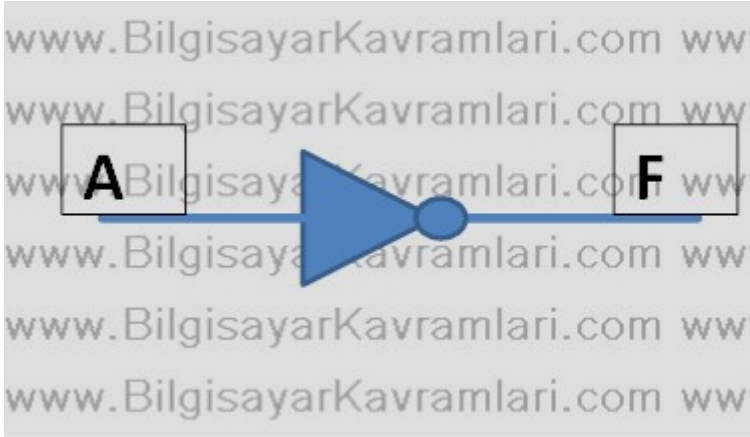
	0	1
0	0 girdisi için, 0 çıktısı alınabilir mi?	0 girdisi için, 1 çıktısı alınabilir mi?
1	1 girdisi için 0 çıktısı alınabilir mi?	1 girdisi için 1 çıktısı alınabilir mi?

Yukarıdaki bu sorulara evet veya hayır cevaplarını vererek evet için 1 ve hayır için 0 yerleştiriyoruz. Örneğin değil kapısı (not gate) 0 için 1 sonucu verir ve 0 için 0 sonucu vermez. Dolayısıyla yukarıdaki doğruluk çizelgesinin matris gösterimini aşağıdaki şekilde yapmak yeterlidir.

0	1
1	0

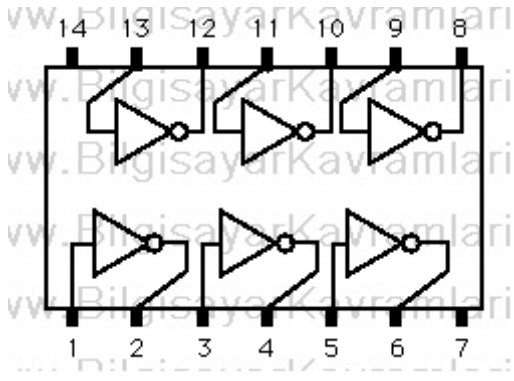
Yukarıda görüldüğü üzere 0 ve 1'lerden sanki bir çarpı veya X harfi üretilmiş gibidir. Bu özelliğinden dolayı, bazı kaynaklarda, değil kapılarına, X kapısı (Xgate) ismi de verilir.

Değil kapısının mantıksal devre tasarımıdaki gösterimi bir üçgen ve daireden oluşmaktadır.



Yukarıdaki şekilde görüldüğü gibi bir devre yerleştirilerek ifade edilir ve burada $F = A'$ şeklinde A'nın tersi alınmaktadır.

Devre tasarımı sırasında IC 7404 (Integrated Circuit, Entegre devre Standardı) kullanılabilir. Bu devreye altılı ters çevirici anlamında Hex Inverter ismi verilmektedir ve devrenin içerisinde 6 adet değil kapısı bulunur.



Devrenin, yukarıda görüldüğü üzere 14 adet bacağı bulunur ve yukarıdaki şemada gösterildiği üzere, giriş ve çıkış ayakları işaretlenmiştir.

Devrenin, 14. bacağı, pozitif besleme alırken, 7. bacak topraklamadır.

SORU 6: Toffoli Kapısı (Toffoli Gate)

Bilgisayar mühendisliğinin de bir çalışma alanı olan mantıksal devre tasarımı konusunda geçen, ve mucidinin adı ile anılan bir kapı örneğidir. Bu kapının en büyük özelliği evrensel olarak geri döndürülebilir olmasıdır (universally reversible). Literatürde bu kapı için CCNOT (control control not) kapısı ismi de verilmektedir.

Bir mantıksal kapının geri döndürülebilir olması, kapının girdisinden elde edilen çıktının tekrar girdi olması halinde, ilk girdinin geri elde edilebilmesidir. Bu karmaşık cümle ile anlatılmak istenen örneğin L kapısı için $L(x) = y$ gibi bir sonuç alınıyorsa, bu kapının tersi olan L' için $L'(y) = x$ sonucunun alınması beklenir. Veya kapının kendisinin ters olması halinde de $L(x) = y$ ve $L(y) = x$ şartlarının aynı anda sağlanması beklenir.

Örneğin klasik [değil kapısı \(not gate\)](#) geri döndürülebilir kapıdır (reversible). Bunu [doğruluk çizelgesine \(truth table\)](#) bakarak kolayca görebiliriz.

Girdi	Çıktı
1	0
0	1

Görüldüğü üzere $L(1) = 0$ ve $L(0) = 1$ olmakta, dolayısıyla tersi alınabilir bir kapı olmaktadır.

Buna karşılık, geri döndürülebilirlik (reversible) konusunun daha iyi anlaşılabilmesi için, geri döndürülemez bir kapı olan veya kapısını inceleyelim.

Girdi	Çıktı
00	0
01	1
10	1
11	1

Yukarıdaki [doğruluk çizelgesinde \(truth table\)](#) görüldüğü üzere, herhangi bir çıktının, girdiye verilmesi durumunda, girdinin geri elde edilmesi mümkün değildir. Örneğin $L(10) = 1$ olmakta ama $L(1) = 10$ olmamaktadır.

Aynı zamanda herhangi bir L' devresi de yukarıdaki tablonun tersini üretemez. Bunun sebebi, 1 çıktısının 01, 10 veya 11 şeklinde geri döndürülme ihtimali olduğu ve 1 çıktısı alındıktan sonra, orijinal girdinin ne olduğunun tahmininin imkânsız olduğudur.

Ve kapısı örneğini ele alarak, bir kapının geri döndürülebilir olması için giriş ve çıkış bitlerinin sayısının aynı olması gerektiğini tahmin edebilirsiniz. Aslında bu durum basitçe [güvercin yuvası kaidesi \(pigeonhole principle\)](#) ile açıklanabilir ve evet bir kapının geri döndürülebilir olması için giriş biti sayısı ile çıkış biti sayısı eşit olmalıdır.

Şayet giriş bitlerinin sayısı ile çıkış bitlerinin sayısı eşit ise, kapının karakterini, yukarıdaki örneklerde olduğu gibi doğruluk çizelgesi (truth table) şeklinde klasik gösterimden farklı olarak gösterebiliriz. Aslında kuantum kapıları (quantum Gates) için vaz geçilmez olan bu gösterim matris gösterimidir.

Örneğin değil kapısını (not gate) ele alalım ve matriste göstermeye çalışalım.

	0	1
0	0	1
1	1	0

Yukarıdaki matris, okunması kolay olsun diye bir satır (en üstteki) ve bir sütun (en soldaki) eklenerek verilmiştir. Bu matriste, satırlar, girdiyi, sütunlar ise çıktıyı tutmaktadır. Yani tablomuzu aşağıdaki şekilde yorumlayabiliriz

	0	1
0	0 girdisi için, 0 çıktısı alınabilir mi?	0 girdisi için, 1 çıktısı alınabilir mi?
1	1 girdisi için 0 çıktısı alınabilir mi?	1 girdisi için 1 çıktısı alınabilir mi?

Yukarıdaki bu sorulara evet veya hayır cevaplarını vererek evet için 1 ve hayır için 0 yerleştiriyoruz. Örneğin değil kapısı (not gate) 0 için 1 sonucu verir ve 0 için 0 sonucu vermez. Dolayısıyla yukarıdaki doğruluk çizelgesinin matris gösterimini aşağıdaki şekilde yapmak yeterlidir.

0	1
1	0

Yukarıdaki bu matrise bakıldığı zaman, bu matrisin [doğruluk çizelgesi \(truth table\)](#) kolaylıkla anlaşılabilir.

Toffoli kapısına gelince, bu kapının doğruluk tablosu ve matrisi aşağıda verilmiştir.

Girdi	Çıktı
0 0	0 0
0 1	0 1

$$\begin{array}{cc|cc} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{array}$$

Yukarıdaki doğruluk çizelgesinin matris hali de aşağıda verilmiştir.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Görüldüğü üzere, toffoli kapısı, [özel veya \(XOR\)](#) şeklinde çalışmaktadır ve ilk biti kontrol bitidir. Yani çıktının ikinci biti (doğruluk çizelgesindeki sağdaki bit), XOR sonucu iken, ilk bit Girdinin ilk biti ile aynıdır.

İki girdi için yukarıda verilen doğruluk çizelgesi ve matris gösterimlerinin, 3 giriş için olanı da aşağıdadır.

Giriş	Çıkış
0 0 0	0 0 0
0 0 1	0 0 1
0 1 0	0 1 0
0 1 1	0 1 1
1 0 0	1 0 0
1 0 1	1 0 1
1 1 0	1 1 1
1 1 1	1 1 0

Böylelikle, yukarıdaki 3 bit girişin aslında tek bit olan çıkışının başında iki bitlik kontrol bulunmakta ve yazının başında bahsettiğimiz CCNOT yani kontrol kontrol değil (not) kapısı olmaktadır. Bu tablonun matris gösterimi aşağıdaki şekildedir.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

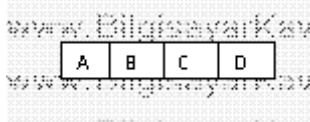
SORU 7: LFSR (Linear Feedback Shift Register)

Bilgisayar bilimlerinde, özellikle şifreleme ve veri güvenliği konularında [doğrusal ahenk sınıfı \(linear congruence \)](#) üretmek için kullanılan yöntemin ismidir. İngilizcedeki linear feedback shift register terimini Türkçede doğrusal geri beslemeli kaydırma yazmacı olarak

tercüme etmek mümkündür. Bu yöntem genellikle ikili tabandaki sayılar üzerinden çalışır ve sistem iki adımdan oluşur :

1. Mevcut sayılar üzerinden yeni bir sayı üreten fonksiyonun çalışması
2. Mevcut sayıların kaydırılması (shift) ve açılan boşluğa bir önceki adımda elde edilen fonksiyon sonucunun yerleştirilmesi.
3. 1. Adımdan tekrar devam edilerek bir sonraki sayının üretilmesi.

Yukarıdaki bu adımları bir örnek üzerinden anlamaya çalışalım. Örneğin sayılarımız 4 bitten oluşsun.



Kullanacağımız fonksiyonu $F = C \oplus D$ olarak tanımlı olsun.

Bu durumda her adımda aşağıdaki şekilde kaydırma işlemi yapılacaktır:

$$F = C \oplus D$$

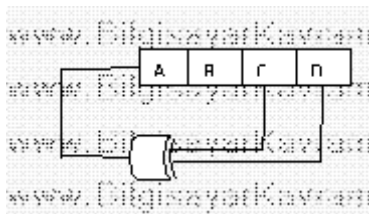
$$D = C$$

$$C = B$$

$$B = A$$

$$A = F$$

Görüldüğü üzere her adımda bütün bitler birer kere sağa kaydırılmış açılan boşluğa, bir önceki adımda olan C ve D bitlerinin değerleri yerleştirilmiştir.



Yukarıdaki bu çizimden de anlaşılacağı üzere 4 bitlik bilginin son 2 biti fonksiyona sokulmuş (buradaki fonksiyon [yahut fonksiyonu \(özel veya \(exclusive or\)\)](#) olarak tanımlıdır) ve çıkan sonuç, kaydırma işleminden doğan boşluğa konulmuştur.

Yukarıdaki bu örneği aşağıdaki örnek sayılar için çalıştıralım. Örneğin LFSR'yi 1111 bilgisi ile besleyerek başlayalım:

$$1111 \text{ (son iki bit } 1 \oplus 1 = 0 \text{)}$$

$$0111 \text{ (baş 0 eklendi yine son iki bit } 1 \oplus 1 = 0 \text{)}$$

0011 (başa yine 0 eklendi yine son iki bit $1 \oplus 1 = 0$)

0001 (başa yine 0 eklendi son iki bit $0 \oplus 1 = 1$)

1000 (başa 1 eklendi son iki bit $0 \oplus 0 = 0$)

0100

0010

1001

1100

0110

1011

0101

1010

1101

1110

1111

Yukarıdaki son adımda, başlangıçta yazmacı (register) beslerken kullandığımız ilk değeri geri elde ettik. Bu anlamda yukarıdaki sayılardan da anlaşılacağı üzere elde edilen sayılar [dairesel bir grup oluşturur \(cyclic group\)](#) ve şifreleme algoritmalarından bu tip bir gruba ihtiyaç duyan sistemlerde kullanılabilir.

Yukarıdaki bu sayıların üzerinde tanımlı olan f fonksiyonun tasarımına göre güvenlik artırımı da söz konusudur.

Örneğin yukarıda verilen F fonksiyonu tasarımının bir eksik yanı bu fonksiyon sayesinde üretilen sayılarda ileri ve geri hareketin kolay olmasıdır.

F fonksiyonunda yapılacak değişikliklerle bu durum düzeltilebilir.

Öncelikle yukarıdaki F fonksiyonundaki sayılarda geri gidilebileceğini görelim:

Yukarıdaki sayılardan rast gele seçilen 1011 sayısını ele alalım.

Bu sayıdan bir önceki sayı için 011X gibi bir sayı olduğunu biliyoruz. Buradaki X değerinin hemen yanında buluna 1 değeri ile yahut işlemine sokulduğunu ve çıkan sonucun da 1 olduğunu biliyoruz. O halde denklemimiz :

$$1 \oplus X = 1$$

Halini alır ki bu durumu veren tek X değeri 0'dır.

Gerçekten de yukarıdaki örnekte 1011 sayısından önce 0110 sayısı gelmektedir.

Görüldüğü üzere f fonksiyonun tasarımına bağlı olarak ileri gitmek kolay olurken geri gitmenin de mümkün olduğu durumlar ortaya çıkabilir.

Örneğimizi biraz değiştirerek 7 bitlik bir mesajdaki son iki bitin yahut sonucunun tekrarlı olarak baştaki 2 bit'e ekleneceğini varsayalım.

Bu durumda sayımızdaki kaydırma işlemini aşağıdaki şekilde tasarlayalım:

A	B	C	D	P	Q	R
---	---	---	---	---	---	---

Yukarıdaki 7 bit için aşağıdaki kaydırma işlemini tasarlıyoruz:

$$F = Q \oplus R$$

$$R = P$$

$$Q = D$$

$$P = C$$

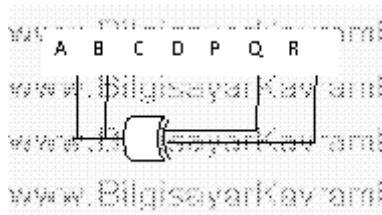
$$D = B$$

$$C = A$$

$$B = F$$

$$A = A$$

Yukarıdaki bu LFSR tasarımını mantıksal bir devre olarak çizecek olursak aşağıdaki sonucu elde ederiz:



Örnek sayılar üzerinden çalışmasını görelim. Bunun için yine başlangıç olarak 1111111 gibi bir girdi ile besleyelim.

1111111 (son iki bit 11 olduğu için $Q \otimes R = 1 \oplus 1 = 0$)

0011111 (sayılar iki kaydırıldı ve ilk iki bite bir önceki adımda bulunan 0 konuldu)

0000111 (sayılar iki kaydırıldı ve ilk iki bite bir önceki adımda bulunan 0 konuldu)

0000001 (sayılar iki kaydırıldı ve ilk iki bite bir önceki adımda bulunan 0 konuldu)

1100000 (bir önceki adımdaki son iki bit 0 ve 1 olduğu için $0 \oplus 1 = 1$ sonucu bulunup ilk iki bite bunlar yerleştirildi)

0011000

0000110

1100001

1111000

0011110

1100111

0011001

1100110

1111001

1111110

1111111

Görüldüğü üzere son adımda yine başlangıç değeri elde edildi ve böylelikle daire tamamlanmış oldu.

Buradaki yeni örneğin bir önceki örnekten en büyük farkı sayılar arasında iler hareket etmek mümkünken geri gidilmesinin güçlüğüdür.

Örneğin yukarıdaki sayılardan birisini rast gele seçelim ve bu iddiayı inceleyelim. Sayımı 0011110 olsun. Bu sayının serideki bir sonraki değerini hesaplamak basittir. Son iki bit yahut işlemine sokulup sayılar iki kaydırılacak ve açılan boşluğa yahut (xor) sonucu yazılacaktır. Bu durumda değer 1100111 olacaktır.

Ancak bu sayıdan (yani 0011110 sayısından) bir önceki sayının, seride hangi sayı olduğunu nasıl bulabiliriz?

Bunun için ilk iki bitin aslında bir önceki adımdaki yahut(xor) işleminde çıkan sonuç olduğunu söyleyerek bir önceki adımda bulunan sayının 11110XY gibi bir sayı olduğunu ve burada

$$X \oplus Y = 0$$

Olduğunu söyleyebiliriz.

Şimdi sorumuz acaba X ve Y değerleri nedir? Sonuç 0 olduğu için ya X=1, Y=1 yada X=0 ve Y=0 doğru olacaktır.

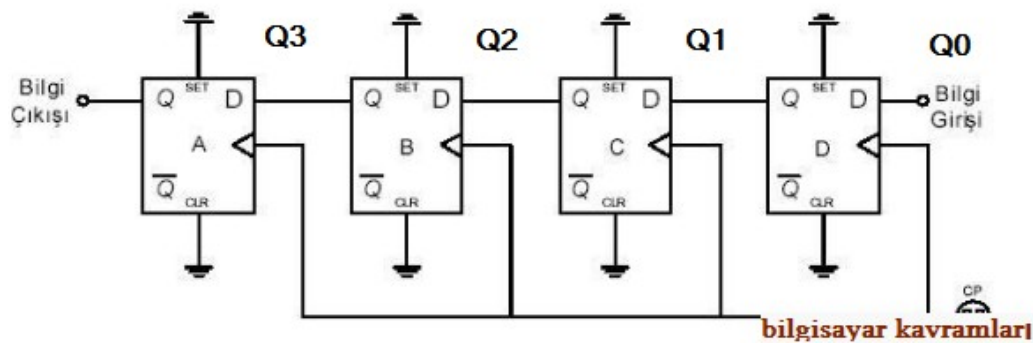
Ne yazık ki hangisi olduğu söylenemez.

Görüldüğü üzere LFSR kullanılarak elde edilen serilere özel F fonksiyonları ile amaca yönelik olarak ek özellikler kazandırılabilir.

SORU 8: Kaydırma Kayıtları (Kaydırma Yazmaçları , Shift Registers)

Bilgisayar bilimlerinin özellikle mantık devreleri (logic circuits) alanında kullanılan bir devre tasarımı şeklidir. Alt yapı olarak [flip-floplardan](#) istifade eden tasarımda amaç girilen ikilik tabandaki [bitlerin \(ikil\)](#) her saat tikiyle bir kaydırılmasıdır.

Bir kaydırma yazmacının mantık devreleri ile tasarımı aşağıdaki şekilde görülebilir:



Yukarıdaki şekilde Q0'dan Q3'e kadar olan girdiler [D-Flipflop](#)lar üzerine yüklendikten sonra her saat vuruşuyla (clock pulse) değerler kaydırılır. Basitçe Q0'daki veri Q1'e ve Q1'deki Q2'ye vs.

Yani örneğin yukarıdaki kaydırma yazmacı (shift-register) devresinde ilk besleme (t_0 zamanı için) değeri olarak Q3, Q2, Q1, Q0 değerlerine 1010 verilmiş olsun. Saat tikiyle t_1 için 0101 - t_2 için 1011 - t_3 için 0111 - t_4 için 1111 şeklinde değerler okunacaktır. Bu örnekte t_1 den sonraki Q0 değerlerinin 1 olarak girdiğini kabul ediyoruz.

Yukarıdaki bu kaydırma yazmacı [D flip floplar](#) ile yapılmış ve 4 ikilik (bit) bir örnektir. Yukarıdaki devre tasarımı için bir [doğruluk çizelgesi \(truth table\)](#) çizilecek olursa:

t_0 için	t_1 için
0000	0000
0001	0011
0010	0100
0011	0111
0100	1000
0101	1011
0110	1100
0111	1111
1000	0000

1001	0011
1010	0100
1011	0111
1100	1000
1101	1011
1110	1100
1111	1111

Şeklinde sonuç elde edilir. Yukarıdaki örnekte Q0 girişinin t0 ve t1 anları için aynı kaldığı kabul edilmiştir.

Bu yazı şadi evren şeker tarafından yazılmış ve bilgisayar kavramları.com sitesinde yayınlanmıştır. Bu içeriğin kopyalanması veya farklı bir sitede yayınlanması hırsızlıktır ve telif hakları yasası gereği suçtur.

Yukarıdaki doğruluk çizelgesine dikkat edilirse kaydırma işleminden de anlaşılacağı üzere ilk [bitin \(ikil\)](#) sonuçta bir etkisi yoktur.

Kaydırma yazmaçları giriş ve çıkışlarına göre 4 farklı grupta incelenebilir.

- SISO (Serial input serial output , seri giriş seri çıkış)
- PIPO (Parallel input parallel output, paralel giriş seri çıkış)
- PISO (Parallel input serial output, paralel giriş seri çıkış)
- SIPO (Serial input parallel output , seri giriş paralel çıkış)

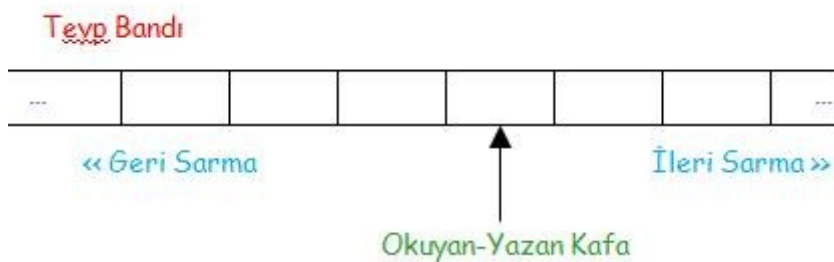
Yukarıdaki çizim ve [doğruluk tablosu](#) verilen kaydırma yazmacı seri giriş ve paralel çıkışa bir örnektir. Burada dikkat edilirse giriş tek bir noktadan seri olarak (arka arkaya) yapılmakta ve çıkış her Q noktasından paralel olarak okunmaktadır.

SORU 9: Turing Makinesi (Turing Machine)

Bilgisayar bilimlerinin önemli bir kısmını oluşturan [otomatlar \(Automata\)](#) ve [Algoritma Analizi \(Algorithm analysis\)](#) çalışmalarının altındaki dil bilimin en temel taşlarından birisidir. 1936 yılında Alan Turing tarafından ortaya atılan makine tasarımı günümüzde pek çok teori ve standardın belirlenmesinde önemli rol oynar.

Turing Makinesinin Tanımı

Basitçe bir kafadan (head) ve bir de teyp bandından (tape) oluşan bir makinedir.



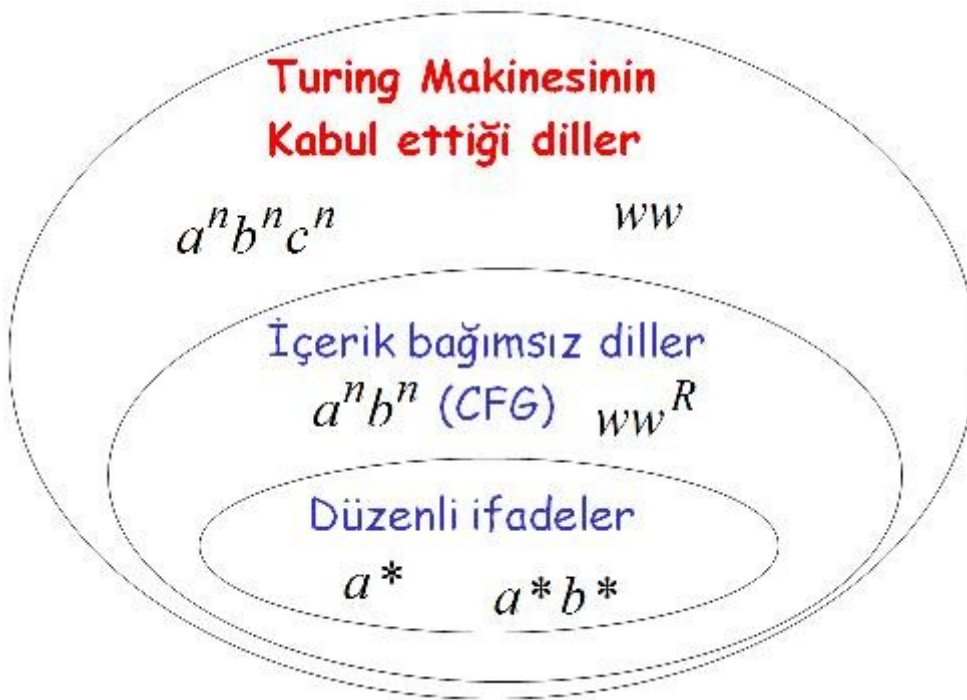
Makinede yapılabilecek işlemler

- Yazmak
- Okumak
- Bandı ileri sarmak
- Bandı geri sarmak

şeklinde sıralanabilir.

Chomsky hiyerarşisi ve Turing Makinesi

Bütün teori bu basit dört işlem üzerine kurulmuştur ve sadece yukarıdaki bu işlemleri kullanarak bir işin yapıp yapılamayacağı veya bir dilin bu basit 4 işleme indirgenip indirgenemeyeceğine göre diller ve işlemler tasnif edilmiştir.



Bu sınıflandırma yukarıdaki venn şeması ile gösterilmiştir. Aynı zamanda [chomsky hiyerarşisi \(chomsky hierarchy\)](#) için 1. seviye (type-1) olan ve Turing makinesi ile kabul edilebilen diller bütün tip-2 ve tip-3 dilleri yani içerik bağımsız dilleri ve düzenli dilleri kapsamaktadır. Ayrıca ilave olarak içerik bağımsız dillerin işleyemediği (üretmediği veya parçalayamadığı (parse)) $a^n b^n c^n$ şeklindeki kelimeleri de işleyebilmektedir. Düzenli ifadelerin işleyememesi konusunda bilgi için [düzenli ifadelerde pompalama savı \(pumping lemma in regular expressions\)](#) ve [içerik bağımsız dillerin işlemeyemesi için de içerik bağımsız dillerde pompalama savı \(pumping lemma for CFG\)](#) başlıklı yazıları okuyabilirsiniz.

Turing Makinesinin Akademik Tanımı

Turing makineleri literatürde akademik olarak aşağıdaki şekilde tanımlanır:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, \diamond, F)$$

Burada M ile gösterilen makinenin parçaları aşağıda listelenmiştir:

Q sembolü sonlu sayıdaki durumların [kümesidir](#). Yani makinenin işleme sırasında aldığı durumardır.

Γ sembolü dilde bulunan bütün harfleri içeren alfabeyi gösterir. Örneğin ikilik tabandaki sayılar ile işlem yapılıyorsa $\{0,1\}$ şeklinde kabul edilir.

Σ sembolü ile makineye verilecek girdiler (input) [kümesi](#) gösterilir. Girdi [kümesi](#) dildeki harfler dışında bir sembol taşıyamayacağı için $\Sigma \subseteq \Gamma$ demek doğru olur.

δ sembolü dilde bulunan ve makinenin çalışması sırasında kullanacağı geçişleri (transitions) tutmaktadır.

\diamond sembolü teyp bandı üzerindeki boşlukları ifade etmektedir. Yani teyp üzerinde hiçbir bilgi yokken bu sembol okunur.

q_0 sembolü makinenin başlangıç durumunu (state) tutmaktadır ve dolayısıyla $q_0 \subseteq Q$ olmak zorundadır.

F sembolü makinenin bitiş durumunu (state) tutmaktadır ve yine $F \subseteq Q$ olmak zorundadır.

Örnek Turing Makinesi

Yukarıdaki sembolleri kullanarak örnek bir Turing makinesini aşağıdaki şekilde inşa edebiliriz.

Örneğin basit bir kelime olan a^* [düzenli ifadesini \(regular expression\)](#) Turing makinesi ile gösterelim ve bize verilen aaa şeklindeki 3 a'ya makinemizin kabul edip etmediğine bakalım.

Tanım itibarıyla makinemizi aşağıdaki şekilde tanımlayalım:

$$M = \{ \{q_0, q_1\}, \{a\}, \{a, x\}, \{q_0 a \rightarrow a R q_0, q_0 x \rightarrow x L q_1\}, q_0, x, q_1 \}$$

Yukarıdaki bu makineyi yorumlayacak olursak:

Q değeri olarak $\{q_0, q_1\}$ verilmiştir. Yani makinemizin iki durumu olacaktır.

Γ değeri olarak $\{a, x\}$ verilmiştir. Yani makinemizdeki kullanılan semboller a ve x'ten ibarettir.

Σ değeri olarak $\{a\}$ verilmiştir. Yani makinemize sadece a girdisi kabul edilmektedir.

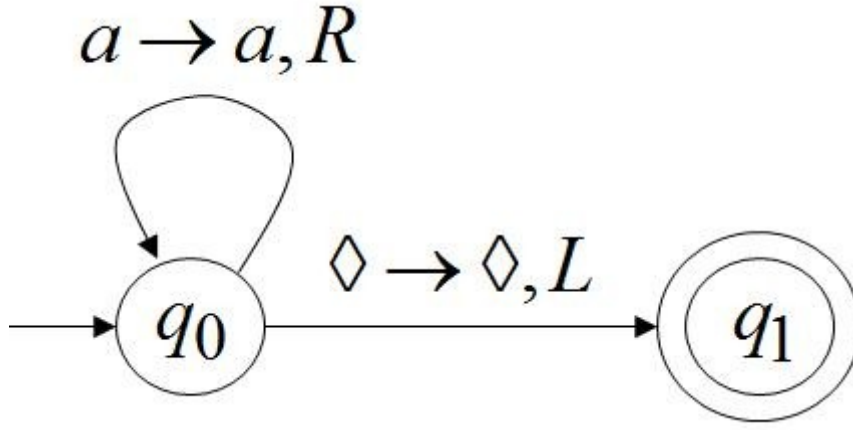
δ değeri olarak iki geçiş verilmiştir $\{q_0 a \rightarrow a R q_0, q_0 x \rightarrow x L q_1\}$ buradaki R sağa sarma L ise sola sarmadır ve görüleceği üzere Q değerindeki durumlar arasındaki geçişleri tutmaktadır.

\diamond değeri olarak x sembolü verilmiştir. Buradan x sembolünün aslında boş sembolü olduğu ve bantta hiçbir değer yokken okunan değer olduğu anlaşılmaktadır.

q_0 ile makinenin başlangıç durumundaki hali belirtilmiştir.

F değeri olarak q_1 değeri verilmiştir. Demek ki makinemiz q_1 durumuna geldiğinde bitmektedir (halt) ve bu duruma gelmesi halinde bu duruma kadar olan girdileri kabul etmiş olur.

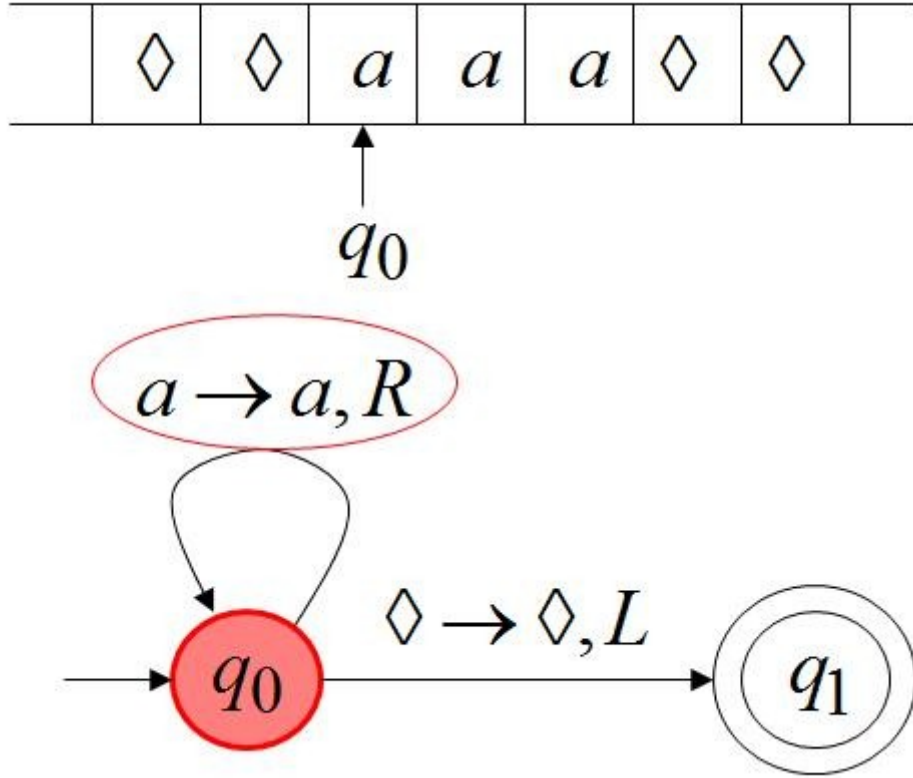
Yukarıdaki bu tanımlı görsel olarak göstermek de mümkündür:



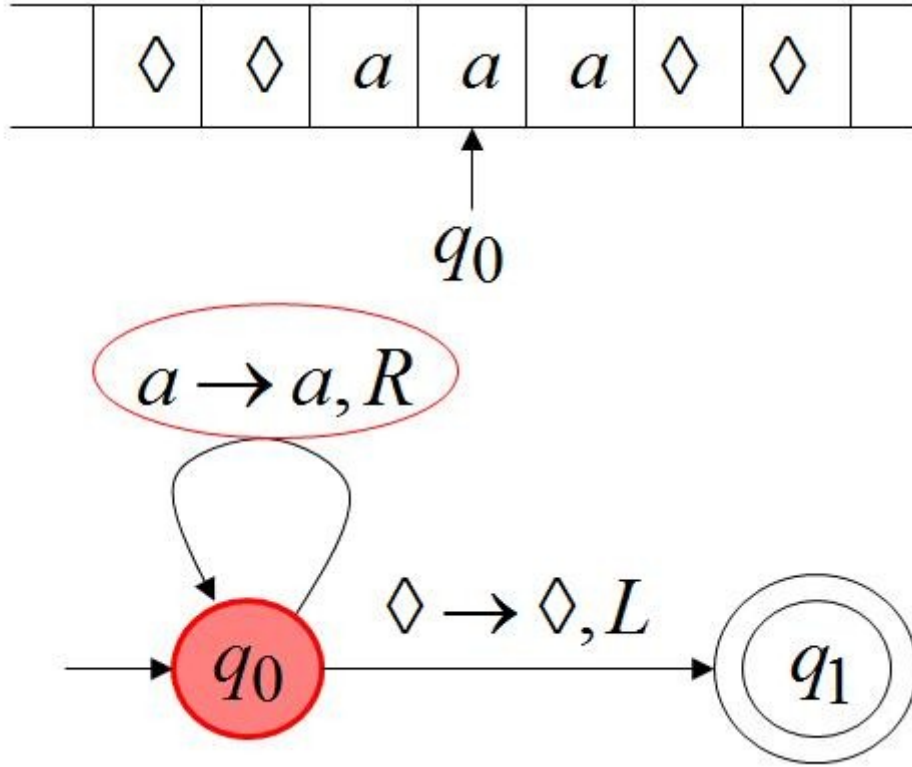
Yukarıdaki bu temsili resimde verilen turing makinesi çizilmiştir.

Makinemizin örnek çalışmasını ve bant durumunu adım adım inceleyelim.

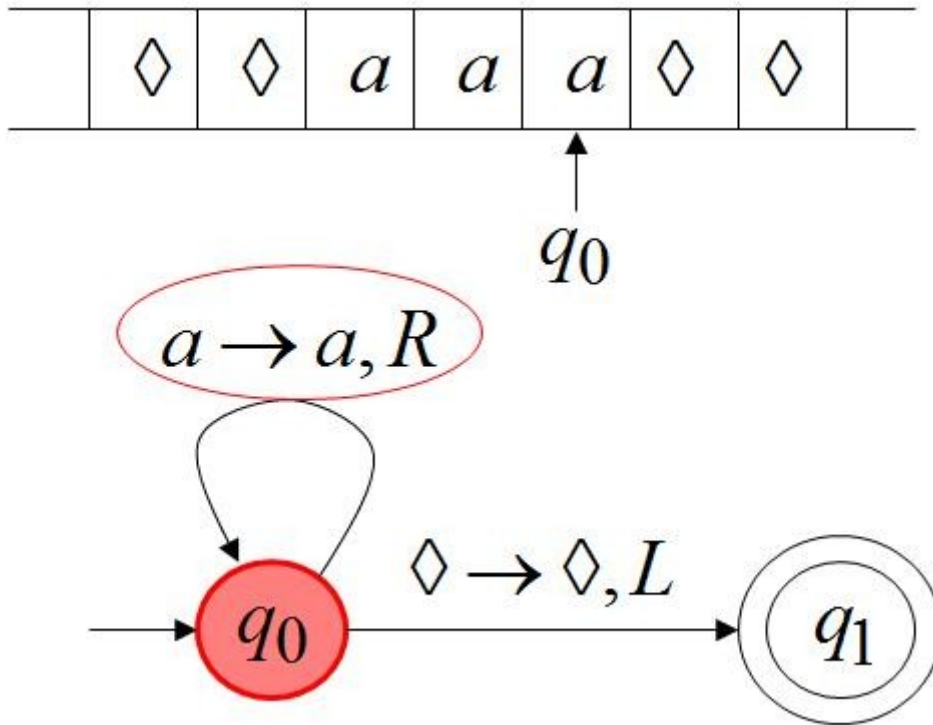
Birinci adımda bandımızda aaa (3 adet a) yazılı olduğunu kabul edelim ve makinemizin bu aaa değerini kabul edip etmeyeceğini adım adım görelim. Zaten istediğimiz de aaa değerini kabul eden bir makine yapabilmektir.



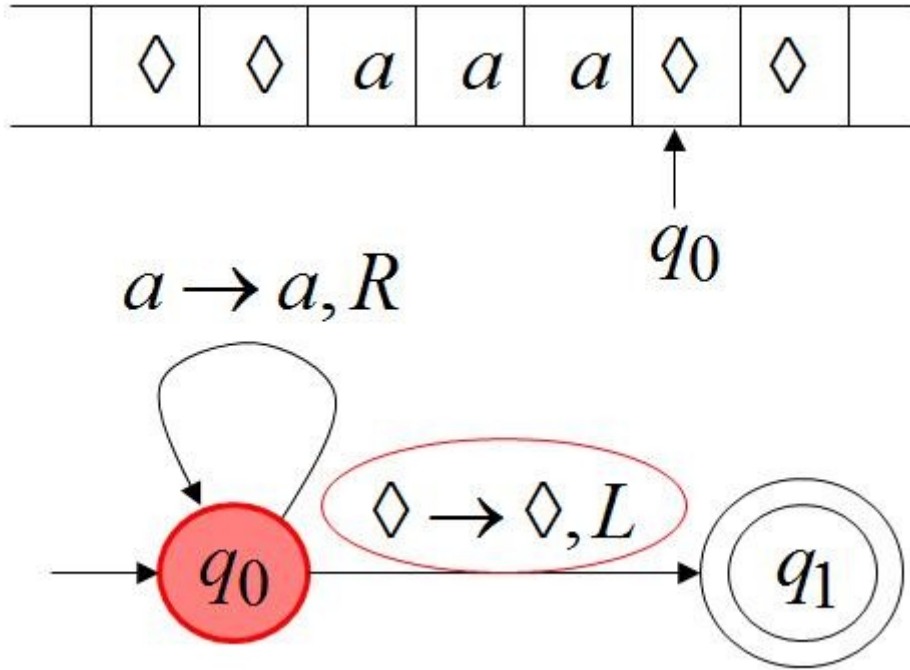
Yukarıdaki ilk durumda bant üzerinde beklenen ve kabul edilip edilmeyeceği merak edilen değerimiz bulunuyor. Makinemizin kafasının okuduğu değer a sembolü. Makinemizin geçiş tasarımına göre q_0 halinde başlıyoruz ve a geldiğinde teybi sağa sarıp yine q_0 durumunda kalmamız gerekiyor.



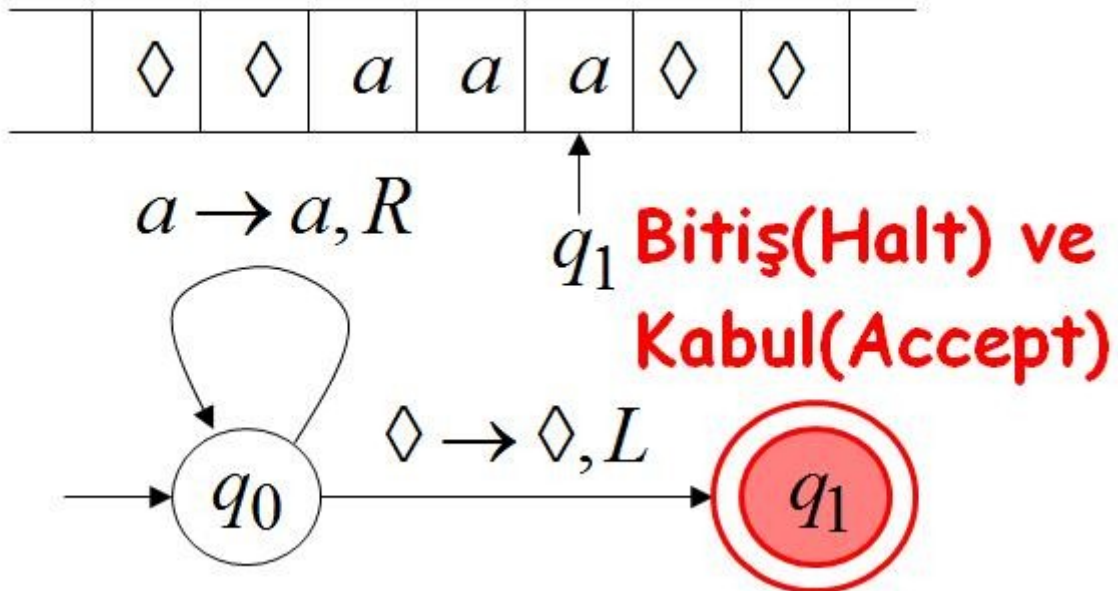
Yeni durumda kafamızın okuduğu değer banttaki 2. a harfi ve bu durumda yine q_0 durumundayken teybi sağa sarıp yine q_0 durumunda kalmamız tasarlanmıştır



3. durumda kafamızın okuduğu değer yine a sembolü olmakta ve daha önceki 2 duruma benzer şekilde q_0 durumundayken a sembolü okumanın sonucu olarak teybi sağa sarıp q_0 durumunda sabit kalıyoruz.



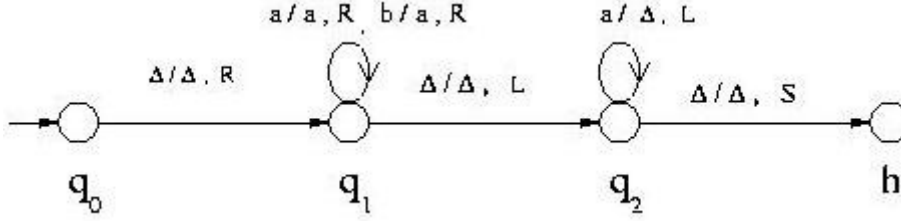
4. adımda teypten okuduğumuz değer boşluk sembolü x oluyor. Bu değer makinemizin tasarımında q_1 durumuna gitmemiz olarak tasarlanmış ve teybe sola sarma emri veriyoruz.



Makinenin son durumunda q_1 durumu makinenin kabul ve bitiş durumu olarak tasarlanmıştı (makinenin tasarımındaki F kümesi) dolayısıyla çalışmamız burada sonlanmış ve giriş olarak aaa girdisini kabul etmiş oluyoruz.

2. Örnek

Hasan Bey'in sorusu üzerine bir örnek makine daha ekleme ihtiyacı zuhur etti. Makinemiz $\{a,b\}$ sembolleri için çalışsın ve ilk durum olarak bandın en solunda başlayarak bandta bulunan sembolleri silmek için tasarlansın. Bu tasarımı aşağıdaki temsili resimde görülen otomat ile yapabiliriz:



Görüldüğü üzere makinemizde 4 durum bulunuyor, bunlardan en sağda olan h durumu bitişi (halt) temsil ediyor. Şimdi bu makinenin bir misal olarak “aabb” yazılı bir bandta silme işlemini nasıl yaptığını adım adım izah etmeye çalışalım.

Aşağıda, makinenin her adımda nasıl davranacağı bant üzerinde gösterilmiş ve altında açıklanmıştır. Sarı renge boyalı olan kutular, kafanın o anda üzerinde durduğu bant konumunu temsil etmektedir.

◇	◇	a	a	b	b	◇
---	---	---	---	---	---	---

q0 durumunda başlıyoruz. Ve boşluk ile bandı sağa sarıyoruz:

◇	◇	a	a	b	b	◇
---	---	---	---	---	---	---

a veya b değeri okundukça bant sağa sarılmaya devam ediyor ve q1 durumunda kalıyoruz.

◇	◇	a	a	b	b	◇
---	---	---	---	---	---	---

◇	◇	a	a	b	b	◇
---	---	---	---	---	---	---

Okunan değer b ise banda geri a değeri yazılıyor

◇	◇	a	a	a	b	◇
---	---	---	---	---	---	---

www.bilgisayarkavramlari.com

◇	◇	a	a	a	a	◇
---	---	---	---	---	---	---

En sağda boşluk değerini okuyunca (◇) Sağa sarma işlemi bitiyor ve geri dönüyoruz

◇	◇	a	a	a	a	◇
---	---	---	---	---	---	---

◇	◇	a	a	a	◇	◇
---	---	---	---	---	---	---

◇	◇	a	a	◇	◇	◇
---	---	---	---	---	---	---

◇	◇	a	◇	◇	◇	◇
---	---	---	---	---	---	---

◇	◇	◇	◇	◇	◇	◇
---	---	---	---	---	---	---

Tekrar boşluk (◇) görülünce makine bitiyor.

Geri sarma işlemi sırasında a değerleri silinmiş oluyor

Netice olarak Hasan Bey'in sorusuna temel teşkil eden ve örneğin q1 üzerindeki döngülerden birisi olan b/a,R geçişi, banttan b okunduğunda banta a değerini yaz manasındadır.

SORU 10: Atomluluk (Atomicity)

Latince bölünemez anlamına gelen atom kökünden üretilen bu kelime, bilgisayar bilimlerinde çeşitli alanlarda bir bilginin veya bir varlığın bölünemediğini ifade eder.

Örneğin programlama dillerinde bir dilin atomic (bölünemez) en küçük üyesi bu anlama gelmektedir. Mesela C dilinde her satır (statement) atomic (bölünemez) bir varlıktır.

Benzer şekilde bir verinin bölünemezliğini ifade etmek için de veri tabanı, veri güvenliği veya veri iletimi konularında kullanılabilir.

Örneğin veri tabanında bir işlemin (transaction) tamamlanmasının bölünemez olması gerekir. Yani basit bir örnekle bir para transferi bir hesabın değerinin artması ve diğer hesabın değerinin azalmasıdır (havale yapılan kaynak hesaptan havale yapılan hedef hesaba doğru

paranın yer deęiřtirmesi) bu sıradaki işlemlerin bölünmeden tamamlanması (atomic olması) gerekir ve bir hesaptan para eksildikten sonra, dięer hesaba para eklenmeden araya başka işlem giremez.

Benzer şekilde işletim sistemi tasarımı, paralel programlama gibi konularda da bir işlemin atomic olması araya başka işlemlerin girmemesi anlamına gelir.

Örneğin sistem tasarımında kullanılan check and set fonksiyonu önce bir deęişkeni kontrol edip sonra deęerini deęiřtirmektedir. Bir deęişkenin deęeri kontrol edildikten sonra içerisine deęer atanmadan farklı işlemler araya girerse bu sırada problem yaşanması mümkündür. Pekçok işlemci tasarımında buna benzer fonksiyonlar sunulmaktadır.

Genel olarak bölünmezlik (atomicity) geliştirilen ortamda daha düşük seviyeli kontroller ile sağlanır. Örneğin işletim sistemlerinde kullanılan [semafor'lar \(semaphores\)](#), kilitler (locks), koşullu deęişkenler (conditional variables) ve monitörler (monitors) bunlar örnektir ve işletim sisteminde bir işlemin yapılması öncesinde bölünmezlik sağlayabilirler.

Kullanılan ortama göre farklı yöntemlerle benzer bölünmezlikler geliştirilebilir. Örneğin veritabanı programlama sırasında koşul (condition) veya kilit (lock) kullanımı bölünmezliği sağlayabilir.

SORU 11: Tehlike (Hazard)

Bilgisayar bilimlerinde özellikle de mantıksal devre tasarımı sırasında karşılaşılan bir durumdur. Basitçe sistemde oluşan veya oluşabilecek tehlikeleri ifade eder. Yani örneğin sistemdeki kapıların (ve, veya, yahut kapıları) yanlış çalışması sonucunda oluşan tehlikelerdir. Temel olarak 3 ayrı grupta toplamak mümkündür:

- Sabit Tehlikeler (Static Hazards)
- Müteharrik Tehlikeler (Dinamik Tehlikeler, Dynamic Hazards)
- Fonksiyonel Tehlikeler (Functional Hazards)

Sabit tehlikeler basitçe, girdinin (input) deęişmesi halinde sonucun (output) deęişmemesi gerekirken deęişmesi durumudur.

Bu tehlike durumu için iki ayrı çözüm olabilir. Birincisinde devreye geciktirmek (Delay) için ilave devrelerin eklenmesi, ikincisinde ise devrenin hatasının düzeltilmesi için ilave devrelerin eklenmesi söz konusu olabilir. Sabit tehlikeler, müdaha edilmesi nispeten basit tehlikelerdir.

Dinamik tehlikelerde ise sorun genelde bir girdi için farklı zamanlarda farklı hatalı sonuçların alınması şeklinde tanımlanabilir. Yani sabit tehlikede olduğu gibi sürekli aynı hata deęil, ya farklı hatalar ya da bazan doğru sonuçların alınması durumudur.

Basitçe bir devreden dinamik tehlikelerin kaldırılması için bütün sabit tehlikelerin kaldırılması yeterlidir. Çünkü genelde dinamik hatalar büyük ve karmaşık devrelerde, alt parçaların sabit tehlikelerinden ortaya çıkmaktadır.

Fonksiyonel Tehlikeler çözülmesi imkansız olan tehlikelerdir. Tanım olarak birden fazla girdinin (input) aynı anda deęişmesi sırasında oluşan tehlikelerdir. Bu tehlikelerin ortadan

kaldırılmasının tek yolu, tek girdi (input) ile tehlikenin oluşturulması veya tespit edilmesidir. Bu sayede tehlike dinamik veya sabit bir tehlike haline dönüşerek çözülebilir.

SORU 12: Kayan Nokta Sayıları (Floating Point Numbers)

Bilgisayar yapılarında ondalıklı sayıları (floatingpoint numbers) iki farklı bilginin tutulması ile gösterilebilir:

mantis x kök ^{üst}

yukarıda verilen formüle göre bir ondalıklı sayıyı önce bir ondalık çarpan sonra da bir kök'ün verilen üstü ile çarpımı olarak göstermek mümkündür.

Örneğin [ikilik tabanda](#) 1101.11 küsurlu sayısını ele alalım (“.” işaretinden sonraki kısım küsurudu). Bu sayıyı göstermek için öncelikle bütün sayıyı önce 4 hane ilerleterek (kaydırarak, float) sayının tamamını küsurlu hale getirelim:

.110111

Yukarıdaki bu sayının orjinal değerini taşıyan gösterimi

.110111 x 2 ⁴

olmalıdır çünkü orjinal sayının 4 hane kaydırılmış halidir. Dolayısıyla sayımızı aşağıdaki iki tamsayıyı tutarak göstermek mümkündür:

mantisa (mantissa) : 110111

üst (exponent) : 0100 (onluk tabandaki 4'ün karşılığı)

Merve Hanımın soruları üzerine aşağıdaki kısmı eklemenin gerekli olduğu anlaşılmıştır:

Kayan noktaların gösterimi

Konuyu örnek bir sayı üzerinden anlamaya çalışalım. Örneğin sayımız 123.321 olsun.

Sayımızın tam kısmı 123'tür küsurat kısmı aşağıdaki şekilde yazılabilir:

$$3*10^{-1} + 2*10^{-2} + 1*10^{-3}$$

Bu düşünce, bizim alıştığımız onluk sistemdeki yaklaşımdır. Bu sayının ikilik tabana nasıl çevrileceği ile ilgili olarak [“ Ondalık sayıların taban dönüşümü ”](#) başlıklı yazıyı okuyabilirsiniz.

Ayrıca yukarıdaki sayımızı bilimsel gösterimde aşağıdaki şekilde yazabiliriz:

$$1.23321 * 10^2$$

Tek hassasiyetli (single precision) kayan nokta sayısı, 32 bitlik bir paket olarak düşünülebilir. Bu yapı aşağıdaki tabloda temsil edilmiştir:

X	XXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX
Yön	Üst	Kök
Sign	Exponent	Mantissa
1 bit	8 bit	23 bit

Yön biti (sign bit)

Bu bit 1 olduğu zaman, sayımız – değerdedir. Şayet 0 veya artı değerde bir sayı ise, bu bit 0 olur.

Üst Biti (Exponent Bit)

Yukarıdaki örnekte gösterilen bilimsel yazıma göre, sayının üst değeri bu alanda tutulur. Buna göre bir sayının 2'nin kaçınıcı kuvveti ile çarpılabileceğini gösterir.

Burada dikkat edilecek bir husus, üst değerinin de eksi olabileceğidir. Örneğin sayımız 0.000123 ise, bu sayının bilimsel gösterimi : $1.23 * 10^{-4}$ olacaktır.

Bu durumda 8 bitlik alanın ilk biti yön bilgisi tutmaktadır. O halde geriye kalan 7 bit, üst bilgisi tutabilir. Buradan çıkan sonuç, sayımızın en fazla 127. üstüne kadar olan değerin tutulabileceğidir ($2^7 = 128$ olduğunu ve bu bilginin ilk değerinin 0 için ayrıldığını dolayısıyla, sayımızın 0 – 127 arasında olacağını hatırlayınız).

Kök biti (mantissa bit)

23 bit alan kaplayan kısımdır. Bu alana bazı durumlarda belirgin anlamına gelen (significand) ismi de verilir. Bu kısım bilimsel gösterimin kök kısmıdır. Örneğin $1.23 * 10^{-4}$ sayısı için kök kısmı 123 olacaktır.

Örnek sayı

Yukarıdaki alanların, bir örnek sayı için nasıl gösterildiğine bakalım:

21,25 sayısını çevirmeye çalışalım. Öncelikle bu sayının ikilik tabandaki karşılığını bulalım:

10101.01

Bu sayıyı, bilimsel gösterime çevirelim:

$$21,25 = 2,125 * 10^{-1} = 2125 * 10^{-3}$$

$$\text{Ayrıca son bir bilgi : } (2125)_{10} = (1000010001101)_2$$

Şimdi bu sayının hafızadaki tutuluşuna bakalım:

Yön bitimiz, sayımız pozitif olduğu için 0 olacak.

Üst bitlerimiz, sayının üstü -3 olduğu için 100000011 olacak. Buradaki son iki bit 3 değerini, baştaki bit ise yönün eksi olduğunu göstermektedir.

Geriye kalan alan ise kök değerini tutacaktır.

$(21)_{10} = (10101)_2$ olduğuna göre, sayımız aşağıdaki şekilde gösterilebilir:

0	10000011	010101000000000000000000
Yön	Üst	Kök
Sign	Exponent	Mantissa
1 bit	8 bit	23 bit

Sonuç olarak sayımız:

01000001101010100000000000000000

şeklinde bulunmuş olur.

Çift hassasiyet ve tek hassasiyet (double precision , single precision)

Yukarıdaki yazı, tek hassasiyet dikkate alınarak hazırlanmıştır. Bunun anlamı sistemin 32 bit olmasıdır. Sistem 64 bit olabilir. Bu durumda çift hassasiyetten bahsedilmelidir. İki sistem arasındaki fark, aşağıdaki şekildedir:

1	bit	yön	biti	(sign	bit)
8	bit	üst	değeridir	(exponent)	
23 bit kök değeridir (mantissa)					
64bitlik	bir	sistemde	ise	durum	aşağıdaki
1	bit	yön	bitidir	(sign	bit)
11	bit	üst	değeridir	(exponent)	
52 bit kök değeridir (mantissa)					

Kayan nokta gösteriminden Onluk sisteme çevirim

Bu çevirimi anlamak için bir örnek üzerinden konuyu anlatalım.

Çevirmek istediğimiz sayı, 0xC0B40000 olsun. Bu sayı onaltılık tabandadır (hexadecimal) ve aşağıdaki şekilde ikilik tabana (binary) çevirilebilir:

Onaltılık	C	0	B	4	0	0	0	0
İkilik	1100	0000	1011	0100	0000	0000	0000	0000

Yukarıdaki çevirimden çıkan ve tek hassasiyetli (single precision) gösterim aşağıda verilmiştir:

1 10000001 011010000000000000000000

Demek ki sayımız eksi bir değermiş (en baştaki yön biti 1 olduğu için) ve ayrıca üstü de eksi imiş (yön bitini tutan grup 1 ile başladığı için).

Gelelim kök kısmının (mantissa) dönüşümüne.

Kök kısmımızdaki değer, 01101 olarak bulunuyor. Bu değeri yukarıdaki bilgiler ile birleştirirsek aslında sayımız aşağıdaki şekildedir:

$$-1.01101 * 2^2$$

Artık problemimiz bu sayıyı onluk tabanda göstermektir. Sayının her basamağının üst değerini yazalım:

$$-(2^0 + 2^{-2} + 2^{-3} + 2^{-5}) * 2^2 = -(2^2 + 2^0 + 2^{-1} + 2^{-3})$$

değerleri hesaplayalım:

$$-(4 + 1 + .5 + 0.125) = -5.625$$

SORU 13: Çıkarıcı Devre (Subtractor Circuit)

Mantıksal tasarım (logic design) kullanılarak bir çıkarma devresi yapmak mümkündür. Devre tasarımına başlanmadan önce ikilik tabandaki çıkarma işlemini hatırlayalım. Bu işlem [1 tümleyeni \(1's complement\)](#) veya [2 tümleyeni \(2's complement\)](#) alınarak sayının eksi halde gösterilmesi ve ardından toplama işleminin yapılması ile tamamlanmaktaydı. Aşağıdaki sayısal örneği inceleyelim:

Örneğin

11011001

sayısının bir tümleyeni aşağıda verilmiştir:

00100110

bu sayıya 1 eklenerek, iki tümleyeni elde edilir:

00100111

Bu sayı aynı zamanda orjinal sayı olan 11011001 sayısının da negatif gösterimi olarak kullanılabilir.

Bunu bir örnek ile göstermek gerekirse, aşağıdaki çıkarma işlemini ele alalım:

$$\begin{array}{r} 11001001 \\ 10110101 \\ - \\ \hline 00010100 \end{array}$$

bilindiği üzere aslında çıkarma işlemini, çıkarılan sayının negatifini alıp toplama olarak da yorumlayabiliriz. Dolayısıyla aslında her toplama devresi bir çıkarma devresi olarak kullanılabilir. Yapılması gereken tek şey çıkarılmak istenen değer negatifini 2 tümleyeni ile almaktır.

Şimdi çıkarıcı bir devrenin 1 [bit \(ikil\)](#) için tasarımını yapabiliriz. Bu tasarım yukarıdaki sebeplerden dolayı tamamen mantık devrelerinin bir egzersizi şeklinde olup gerçek bir uygulama için anlamlı değildir.

A	B	C
0	0	00
0	1	11
1	0	01
1	1	00

Yukarıdaki tasarıma göre $C = A - B$ işlemi gösterilmiştir. C değerinin başındaki hane sign bit(yön ikili) olarak düşünülebilir yani sayının eksi olması durumunda 1, artı olması durumunda ise 0 değeri almaktadır.

Bu devredeki iki haneli C değerini C1 ve C2 bitleri (ikilleri) olarak ifade edece olursak:

C1 için karnaugh hartiası:

0 1

0 0

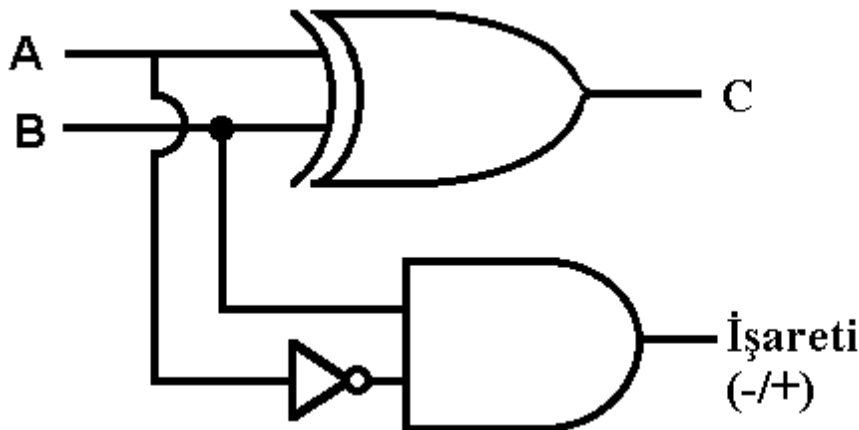
C2 için karnaugh haritası:

0 1

1 0

olarak bulunur. C2 değerinin [yahut \(XOR \(özel veya\)\)](#) olduğu açıktır. Buna göre $C2 = A \oplus B$ denilebilir.

C1 için ise $B \wedge A'$ sonucu çıkmaktadır. Sonuçta elde ettiğimiz çıkarıcı devreyi aşağıdaki çizimde görebiliriz:



SORU 14: İkillik Prensibi (Duality Principle, İstaniyet)

Din ve felsefede benzer anlamlara gelmesine karşılık bu yazının amacı bilgisayar bilimleri için önemli olan matematikteki ikilik prensibini açıklamaktır.

Bir matematikçi bu kavramı basitçe şöyle açıklayabilir “boyalı elimle bir cama ellesem ve elimin izi camda çıksa, camın her iki yönünden gördüğüm görüntü birbirinin ikilidir (dual)”.

Bu açıklama aslında kavramın ne olduğunu göstermektedir. Örneğin iktisatta kârlılık arttırmak için yapılan bir çalışmanın maliyet azaltmak olması ve arttırkam (maximization) ve azaltmak (minimization) kavramlarının birbirinin ikili (dual) olması gibi.

Örneğin bilgisayar bilimlerinde mantık işlemlerinde (Boole algebra) kullanılan aşağıdaki eşitlikleri inceleyelim:

Yer Değiştirme (Commutative Law)

$$(a) \quad A \quad V \quad B \quad = \quad B \quad V \quad A$$
$$(b) \quad A \wedge B = B \wedge A$$

Birleştirme (Associate Law)

$$(a) \quad (A \quad V \quad B) \quad V \quad C \quad = \quad A \quad V \quad (B \quad V \quad C)$$
$$(b) \quad (A \wedge B) \wedge C = A \wedge (B \wedge C)$$

Dağılma (Distributive Law)

$$(a) \quad A \quad \wedge \quad (B \quad V \quad C) \quad = \quad (A \quad \wedge \quad B) \quad V \quad (A \quad \wedge \quad C)$$
$$(b) \quad A \quad V \quad (B \quad \wedge \quad C) \quad = \quad (A \quad V \quad B) \quad \wedge \quad (A \quad V \quad C)$$

Kendisi Kuralı (Identity Law)

$$(a) \quad A \quad V \quad A \quad = \quad A$$
$$(b) \quad A \wedge A = A$$

Yukarıdaki örneklerde görüldüğü üzere her eşitlikte ikil bir karşılık gösterilmiştir (a eşitliğinin ikili b, b eşitliğinin ikili de a eşitliğidir).

Örneğin mantık işlemlerinde yukarıdaki kurallar çerçevesinde [DeMorgan kuralı](#) geliştirilmiş ve bu kurala göre bütün [V\(veya, or\)](#) işlemleri [A \(ve, and\)](#) işlemi olarak yazılabilir. Tabi buradaki kuralımız her [kaziyeinin \(önerme\)](#) tersinin alınmasıdır.

Not : Bu yazıya gelen sorulara cevap olarak. Dinde ikillik için örneğin yahudilikteki yetzer ha-ra (kötü olanı yapmak) ve yetzer ha-tov (iyi olanı yapmak) veya kurandaki zariyat 41/49 “herşeyi çift yarattık ki düşünüp ders alasınız”, Rahman 55/17 “iki doğunun ve iki batının rabbi” ayetlerine dayanan tefsirlerde ve şeytan ve melek ayırımında (ki bu ikillik hemen bütün semavi dinlerde vardır), veya Manihaizm ve Mecusilikteki “Nur” ve “Karanlık” kavramlarındaki (Yani iyilik tanrısı olarak aydınlık (ateşe) tapınmak ve kötülük tanrısı olarak karanlık) kavramlarında veya Monistik (tekçi) yaklaşımı benimsemiş özellikle katolik hristiyanlıktaki yine özellikle descartes’tan sonra açığa çıkan ruh ve beden ayırımında ikillik kavramları görülür. Elbette bu yazdıkların sadece basit birer örnek olarak alınmıştır ve heps ile ilgili çok geniş ve detaylı kaynaklar mevcuttur.

SORU 15: Kuantum İşleme (Quantum Computing)

Bu yazının amacı kuantum bilgisayarları ve kuantum işleme (Quantum Computing) konusunda fikir vermek ve yapılan çalışmaların arkasındaki felsefeyi aktarmaktır.

Kuantum bilgisayarları basitçe veriyi işlemek için çok küçük parçacıklar kullanır. Örneğin her gün yolda görebileceğimiz basit bir çakıl taşı aslında bir kuantun işlemi olarak kabul edilebilir. Temelde çakıl taşının yaptığı iş uzayda (kainatta) çok küçük parçacıkların bir arada durmasını sağlaması ve neticede bir konumlandırma işlemi yapmasıdır.

Günümüz bilgisayar teknolojilerinin üzerine inşa edilmiş olan [Von Neumann bilgisayarlarında](#) en düşük veri ünitesi ikildir (bit). Benzer şekilde kuantum bilgisayarları içinde kubit (qubit = quantum bit) kullanılmaktadır. Normal ikilde (bit) sadece 1 ve 0 değerleri depolanabilirken bir kubit içinde 0, 1 veya her ikisi birden bulunabilmektedir. Bu konuyu daha iyi anlayabilmek için [kubit kavramını](#) daha detaylı okuyabilirsiniz.

Kuantum hesaplamalarının en büyük farklılığı Kuantum paralellığıdır. Kuantum paralellığı (Quantum parallelism) adı da verilen bir işlemde kubitlerin heri ki durumu da göz önünde bulundurulmaktadır. Yani kubit 0 veya 1 durumunda olduğunda sonucun alacağı iki farklı değer ayrı ayrı hesaplanmış gibi tek bir işlemde hesaplanmaktadır.

Kuantum işleme sırasında performans avantajı sağlandığı bir gerçektir. Bu avantajın nasıl sağlandığını anlamak için 200 haneli bir sayıyı çarpanlarına ayıracağımızı düşünelim. Bu işlem günümüz teknolojisindeki 1500 kadar bilgisayarın paralel çalışması ile yaklaşık 700bin yıl sürmektedir. Kuantum bilgisayarları kullanılarak bu işlem ise yaklaşık bir kaç milyon işlem ile sonuca ulaşmaktadır. Buradaki temel fark aynı anda birden fazla durumun kuantum bilgisayarları ile paralel olarak işlenebilmesidir (Kuantum paralellığı).

SORU 16: Kubit (Qubit)

Günümüz bilgisayar teknolojilerinin üzerine inşa edilmiş olan [Von Neumann bilgisayarlarında](#) en düşük veri ünitesi [ikildir \(bit\)](#). Benzer şekilde [kuantum bilgisayarları](#) içinde kubit (qubit = quantum bit) kullanılmaktadır. Normal [ikilde \(bit\)](#) sadece 1 ve 0 değerleri depolanabilirken bir kubit içinde 0, 1 veya her ikisi birden bulunabilmektedir. Bu konuyu daha iyi anlayabilmek için kubit kavramını daha detaylı okuyabilirsiniz.

Bir kubit, bazı elementlerin atomları üzerine inşa edilmiştir. Bu konuda hidrojen güzel bir örnek olabilir. Bilindiği üzere hidrojen atomu basit bir elementtir ve bir elektron ve bir çekirdekten oluşmaktadır. Elektronlar ise çeşitli enerji seviyesinde bulunabilirler. Bu enerji seviyeleri 0 veya 1 değerlerini göstermek için kullanılacak olsun. Basitçe elektronun en düşük yörüngede bulunması 0 en yüksek yörüngede bulunması ise 1 olarak ifade edilecek olsun.

Bu konuda ilave bir bilgi de elektronların LASER marifetiyle yörüngelerinin değiştirilebildiği ve yüksek ve düşük yörüngeler arasında hareket ettirilebildiğidir. Burada LASER sisteme foton katmakta ve kısaca değeri 0 ile 1 arasında değiştirmektedir. Bu işlemi basit bir olumsuz (not) operatörüne benzetebiliriz. Yani mantıksal olarak giriş değerinin olumsuzunu almaktadır.

Kubitlerin normal ikillerden (bit) ayrıldığı nokta ise sisteme yörünge değiştirmek için gereken LASER etkisinin yarısının yapılması durumudur. İşte normal bir bitten kubitin ayrıldığı noktada burasıdır. Bu durumda elektron her iki yörüngede de bulunmaktadır. Bu duruma süper konum (Super position) adı verilmektedir.

Super konumun hesaplamalarda da sağladığı müspet etkiler bulunmaktadır. Kuantum paralellığı (Quantum parallelism) adı da verilen bu işlemde kubitlerin heri ki durumu da göz

önünde bulundurulmaktadır. Yani kubit 0 veya 1 durumunda olduğunda sonucun alacağı iki farklı değer ayrı ayrı hesaplanmış gibi tek bir işlemde hesaplanmaktadır.

SORU 17: Doğrusal Ayrılabilirlik (Linear Seperability)

Yapay sinir ağlarının en basit anlamda incelenebilmesi için problemi iki adet [ikil haneleri olan \(binary digists\)](#) bir girdiye bir de tek ikil (binary) çıktıya sahip bir örnek üzerinden inceleyelim.

Aşağıda iki farklı fonksiyonun [gerçeklik çizelgesi \(doğruluk tablosu, truth table\)](#) verilmiştir. A ve B değerleri girişi C ise çıkışı ifade etsin:

F fonksiyonu için :

A B C

0 0 1

0 1 0

1 0 1

1 1 1

G fonksiyonu için

A B C

0 0 0

0 1 1

1 0 1

1 1 0

Olarak tanımlanmış olan bu fonksiyonların ilki (F fonksiyonu) doğrusal olarak ayrılabilirken ikincisi (G fonksiyonu) doğrusal olarak ayrılabilir değildir. Bu durumu aşağıdaki [karnaugh haritaları \(karnaugh map\)](#) üzerinde göstermeye çalışalım:

	0	1		0	1
1	1	1		0	1
0	1	0		1	0

Görüldüğü üzere ilk fonksiyonun (F) [karnaugh haritası](#) üzerinde bir doğru ile sonuçları iki gruba bölmek doğrusal ayırım yapmak mümkün iken (linearly seperable) ikinci fonksiyon (G) için aynı doğrusal ayırım işlemi yapılamaz. Aynı zamanda ikinci fonksiyon sonuçları için karnaugh haritası dışında farklı bir harita çizmek ve yine doğrusal olarak ayıran bir çizgi elde etmek de imkansızdır. Yani kolon veya satırları yer değiştirdiğinizde bütün alternatiflerde doğrusal olarak ayrılamayan bir sonuç elde edersiniz.

Bir problemin yada fonksiyonun doğrusal olarak ayrılabilir olup olmaması problemin yapay sinir ağları tarafından çözülebilirliğinin kolaylığını da belirlemektedir. Ne yazık ki doğrusal olarak ayrılamayan problemleri yapay sinir ağı ile çözmek çok daha zordur.

Dikkat edilirse yukarıdaki doğrusal olarak ayrılamayan fonksiyon bir [yahut \(özel veya exclusive or\) fonksiyonudur](#). Literatüre de bu şekilde girmiş olan doğrusal olarak ayrılamayan fonksiyonlar (veya kısaca [“XOR problem \(yahut problemi\)”](#)) programlama dünyasındaki “Hello World” yazdırmak kadar meşhurdur.

SORU 18: Yahut (Özel Veya (exclusive or, farklılık operatörü))

İki kaziye (önerme) arasındaki farklılık durumuna göre çalışan operatördür. Yani sonuçların aynılığı durumunda yanlış, farklılığı durumunda doğru döndüren operatördür. Basitçe ikili tabanda iki sayının [\(bit\)](#) farklı olup olmadığını kontrol için de kullanılabilir.

Dilimizde bu işlemi karşılayan kelime “yahut” kelimesidir. Yani a yahut b doğrusanın anlamı a veya b’den birisi doğru diğeri yanlış olacak demektir (ikisi de doğru veya ikisi de yanlış olamaz)

Giren sayılar aynı ise 0, farklı ise 1 sonucu üretir. Aşağıda [doğruluk tablosu](#) verilmiştir:

A	B	Xor
0	0	0
0	1	1
1	0	1
1 1 0		

SORU 19: CRC (cyclic redundancy check, çevrimsel fazlalık sınaması)

Hata algılama yaklaşımlarından birisidir. Bu yöntemde işlenmekte olan verinin dışında ilave bir kontrol verisi daha bulunur. Bu ilave bilgi ile bütün bilgi kontrol edilerek bilgide bir

bozulma olup olmadığı kontrol edilir. Örneğin ağ iletişiminde gidip gelen bilginin kontrol edilmesinde veya CD gibi kayıt ortamlarında verinin bozulup bozulmadığının kontrol edilmesinde kullanılır.

Çalışması:

Her iki tarafın da bildiği bir sayı bölen olarak kullanılır. Örneğin bölen sayımız 13 olsun ve bunu her iki tarafta başlangıçta biliyor olsun. (Bu sayı standarta bağlıdır lütfen standartların anlatıldığı kısıma bakınız.)

Gidecek olan verilerimiz aşağıda yazılmış
19 54 89 22 03 44 19
CRC uygularken bu verilerin toplamı alınır : 250
CRC hesaplanırken daha önceden bildiğimiz sayıya bu sonuç bölünür ve kalan alınır: $250 \% 13 = 3$

Dolayısıyla yukarıdaki veriler yollanırken CRC olarak 3 sayısı yollanmaktadır. Veri alındıktan sonra kontrol edilmesi:

yukarıdaki veriler alındıktan sonra alan taraf da verileri toplar, bu toplamdan CRC bilgisini çıkarır ve $250 - 3 = 247$ sayısını bulur. Daha önceden bildiği 13 sayısına böler ve 0 sonucunu bulursa iletim hatasızdır yargısına varır $247 \% 13 = 0$ şayet hata olsaydı sonuç 0'dan farklı çıkardır. Burada görüldüğü üzere CRC'nin de hata yapma ihtimali vardır örneğin veri bozulması verinin ilk kısmının 1 fazla olması şeklindeyse: 20 54 89 22 03 44 19

verisi alındığında bu hata algılanır : $251 - 3 = 248 \% 13 = 1$ ve sonuç 0 olmadığı için hata kararına varılacaktı ancak hata miktarı daha önceden bilinen sayının (ki bu örnekte 13) katı şeklinde olursa hatanın yakalanması imkansızdır. Örneğin veri bozulması sonucu: 32 54 89 22 03 44 19 sayıları çıkmış olsun. Bu sayıların toplamı 263 olacak ve CRC kısmı olan 3 değeri çıktıktan sonra $260 \% 13 = 0$ olacaktır. Görüldüğü üzere orjinal verimizden farklı olmasına rağmen hatasız olarak kabul edilmiştir.

SORU 20: Sayıcı (Counter)

Bilgisayar devrelerinde sayıcı (counter) kavramı çok farklı alanlarda kullanılmaktadır. Bu alanlardan birisi de mantıksal devre tasarımıdır. Buna göre mantıksal devremiz ikilik tabandaki sayıları 1'er arttırarak yeni sayılar üretmeli ve bu işlemi bir döngü halinde yapmalıdır. Yani aşağıda verilen durum geçiş diyagramı (state transition diagram) olduğu gibi her durumdan diğer durumlara geçişi başarılı bir şekilde yapmalıdır.

Yukarıdaki şekilde basitçe her durumdan bir sonraki duruma geçiş bir sayıcı devre için verilmiştir. Buna göre örneğin bir sayıcının 100 durumundan sonra alması istenen durum 101 olmalıdır. Bu sayıcının ikili tabanda çalıştığına dikkat edilmelidir.

Yukarıda verilen bu sayıcının geçiş tablosu aşağıdaki şekilde doldurulabilir. Burada her durumdan bir sonraki duruma geçiş verilmiştir.

Şimdiki	Sonraki
Durum	Durum

A2	A1	A0	A2	A1	A0
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

Yukarıdaki diyagramda üç [bit](#) için çalışan bir sayıcının önceki ve sonraki durumları verilmiştir. Bu sayıcının [T Flip Flop](#) ile tasarlanmış hali için T [flip flop](#)'un karakteristik tablosuna bakılarak aşağıdaki değerler yazılabilir:

Şimdiki			Sonraki			FlipFlop		
Durum			Durum			Girişleri		
A2	A1	A0	A2	A1	A0	TA2	TA1	TA0
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	1	1
1	1	1	0	0	0	1	1	1

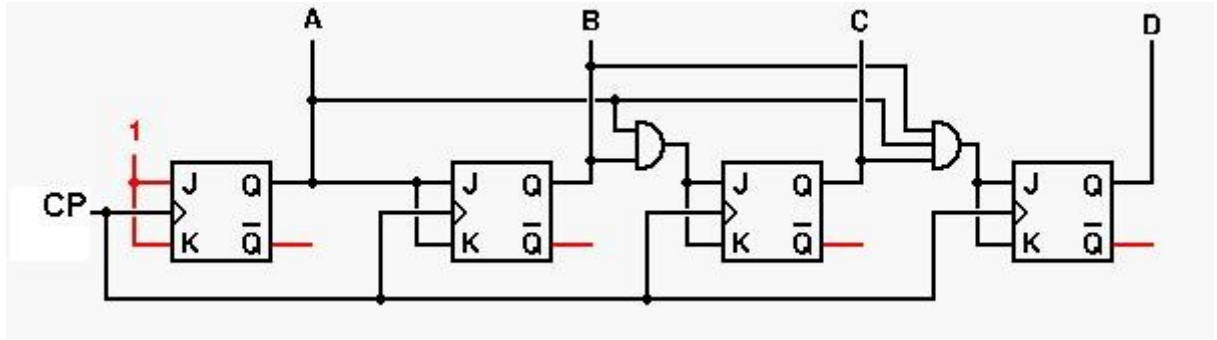
Yukarıdaki tabloda her durum geçişi için 3 farklı [flip flop](#)'un (ki her flip flop ayrı bir [bit](#) için kullanılmıştır) giriş değerlerinin alması gereken değerler verilmiştir. Bu devrenin çizimi aşağıdadır:

Benzer bir sayıcıyı bu defa 4 [bit](#) için ve [JK flip flop](#) kullanarak ve onluk tabanda yapmayı deneyelim [Flip Flopun doğruluk çizelgesi](#) aşağıda verilmiştir:

Durumlar				10luk Tabanda Karşılığı
D	C	B	A	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2

0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

Bu tabloyu veren [JK flip flop](#) çizimi aşağıdadır:



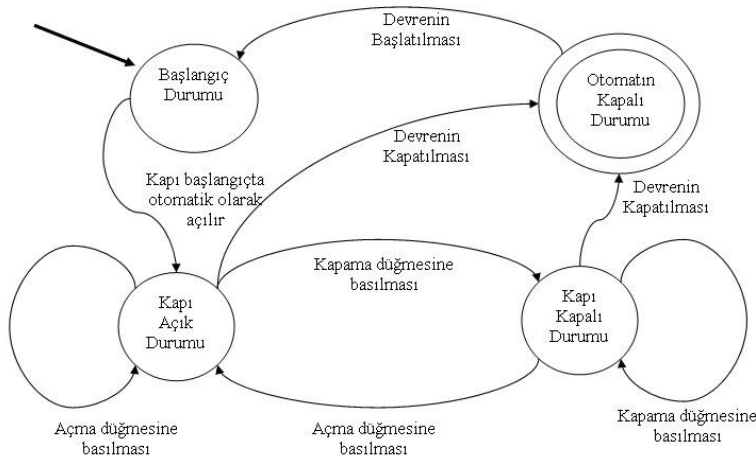
SORU 21: Sonlu Durum Makinası (Finite State Machine, Finite State Automaton)

Sonlu durum makinaları bir çizim şeklidir. Bu çizim şeklinde çeşitli durumlar ve bu durumlar arası geçiş şekilleri gösterilir. Örneğin aşağıda basit bir kapı açma ve kapama makinesi verilmiştir:



Yukarıdaki şekilde makine, açık durumdan kapalıya geçmek için kapama düğmesine basılmasını bekler. Ters durumda da kapalıdan açığa geçmek için kapama düğmesinin basılmasını bekler. Durum makinemizde kapalı durumdayken kapama düğmesine ve açık durumdayken açma düğmesine basılması bir durum değişikliği doğurmaz ve istenildiği kadar basılabilir.

Yukarıdaki şekilde bir başlangıç veya bitiş durumu belirtilmemiştir. Yani makine sonsuza kadar çalışmaktadır ve herhangi bir başlangıç koşulu yoktur. Yukarıdaki bu makinenin bir de açma ve kapama tuşları olduğunu düşünelim. Bu durumda makinenin başlangıcı açma düğmesi ve bitışı de kapama düğmesi ile olacaktır. Aşağıdaki şekilde yeni makinemizin çizimi gösterilmiştir:



Yukarıdaki şekilde sonlu durum makinemize ilave olarak başlangıç ve bitiş durumları da eklenmiştir. Buna göre kalın gidi oku başlangıç durumunu gösterir ve makinemiz buradan başlar. Çift çember içindeki durum ise bitiş durumudur ve istenirse makine burada sona erdirilir, veya çalışmaya devam edebilir. Görüldüğü üzere yukarıdaki şekilde kapı otomatı başlatıldıktan sonra istenildiği kadar açma kapama işlemi yapılmakta, gelen bu komutlara göre kapının durumu değişmektedir. Kapı otomatı kapatıldıktan sonra gelen açma ve kapama emirleri doğal olarak icra edilemez.

SORU 22: flip flop (flipflop)

Flip Flop kavramı temel olarak 1 [bitlik](#) bilginin tutulduğu ünedir. Bu devre elamanında her zaman iki çıkış olur (Q ve tersi olan Q'). Bu çıkışların değerleri kalıcıdır. Yani bir sonraki giriş değerine kadar geçici süre ile sabitlenmekte ve yeni giriş değerleri alınana kadar sabit olarak kalmaktadır. Bu durum geçici bir hafıza olarak kabul edilebilir. Aşağıda en çok kullanılan Flip Flop tipleri verilmiştir. Bu Flip Flop tipleri temel olup bunların üzerinde değişiklikler yapılarak veya harmanlanarak daha farklı amaçlar için Flip Floplar da üretilebilmektedir.

FlipFlop İsmi	Sembolü	Doğruluk Çizelgesi	Mantıksal Denklem	İç Diyagramı
---------------	---------	------------------------------------	-----------------------------------	--------------

SR		<table><tr><th>S</th><th>R</th><th>Q(sonra)</th></tr><tr><td>0</td><td>0</td><td>Q</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>?</td></tr></table>	S	R	Q(sonra)	0	0	Q	0	1	0	1	0	1	1	1	?	$Q(\text{sonra}) = S + R'QSR = 0$	
S	R	Q(sonra)																	
0	0	Q																	
0	1	0																	
1	0	1																	
1	1	?																	
JK		<table><tr><th>J</th><th>K</th><th>Q(sonra)</th></tr><tr><td>0</td><td>0</td><td>Q</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>Q'</td></tr></table>	J	K	Q(sonra)	0	0	Q	0	1	0	1	0	1	1	1	Q'	$Q(\text{sonra}) = JQ' + K'Q$	
J	K	Q(sonra)																	
0	0	Q																	
0	1	0																	
1	0	1																	
1	1	Q'																	
D		<table><tr><th>D</th><th>Q(sonra)</th></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	D	Q(sonra)	0	0	1	1	$Q(\text{sonra}) = D$										
D	Q(sonra)																		
0	0																		
1	1																		
T		<table><tr><th>T</th><th>Q(sonra)</th></tr><tr><td>0</td><td>Q</td></tr><tr><td>1</td><td>Q'</td></tr></table>	T	Q(sonra)	0	Q	1	Q'	$Q(\text{sonra}) = TQ' + T'Q$										
T	Q(sonra)																		
0	Q																		
1	Q'																		

Yukarıdaki tablo tahlil edilirken göz önünde bulundurulması gereken önemli bir nokta doğruluk çizelgeleridir. Bu çizelgeler okunurken giriş değeri ve sonucunda alınan çıkış değeri (Q) verilmiştir.

Devrelerin iç diyagramları verilirken dikkat edilirse bütün Flip Floplar için temel olan RS Flip Flop türüdür. Gerçekten de RS Flip Flop en basit Flip Flop örneği olup diğer bütün karmaşık Flip Floplara temel teşkil etmektedir.

Flip Flop devreleri hafızalara ve sayıcılara temel teşkil etmektedir.

SORU 23: Salt okunur bellek (read only memory , ROM)

üzerine sadece bir kereye mahsus yazılabilen ve daha sonra istenildiği kadar okuma işlemi yapılabilen hafıza tipidir. Buna göre bellek üretim sırasında üzerine yazılan bilgiyi saklar ve bu bilgiyi değiştirmenin bir yolu yoktur. Daha sonraları çıkan teknolojik ilerlemeler ile aşağıdaki salt okunur bellek türleri kullanıcılara farklı alternatifler sunmaktadırlar: PROM (programlanabilir salt okunur bellek, programmable read only memory): Bu hafıza tipi programının müdahalesi sayesinde üzerine bilgi yazılabilen bir yapıya sahiptir. Buna göre programcı hafıza üzerindeki sigortaları program vasıtasıyla atırarak verileri yazabilmektedir. Bu hafıza tipi de sadece bir defaya mahsus programcı tarafından yazılabilmektedir. salt okunur bellekten farkı üzerinde veri yazılı olmadan üretilmekte ve üzerine veriyi programcı yazabilmektedir.

EPROM (silinebilir programlanabilir salt okunur bellek, erasable programmable read only memory): Bu hafıza tipi bir önceki PROM'dan farklı olarak ultraviyole (mor ötesi) ışıktaki kendi

üzerine yazılmış bilgileri fabrika çıkışı haline getirmektedir. Yani programcı bu hafıza üzerine veriyi yazabilmekte ve istediği zaman bu bilgileri silerek tekrar programlayabilmektedir.

EEPROM (elektronik olarak silinebilir ve programlanabilir bellek, electrically erasable programmable read only memory): Bu hafıza tipi bir önceki EPROM'dan farklı olarak elektrik akımıyla silinebilmektedir. Bu sayede devreyi kesin ve hızlı bir şekilde sıfırlamak ve daha sonra üzerine yeni bilgileri programlamak mümkün olmaktadır.

Bir salt okunur belleğin tasarım şeması aşağıda verilmiştir.

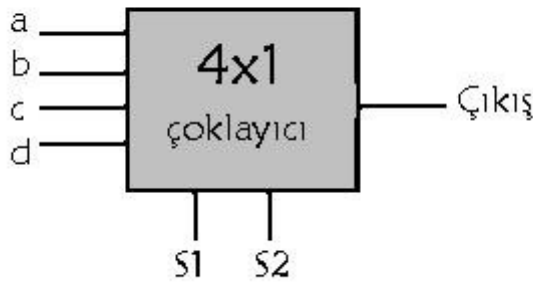
yukarıdaki resim morris mano'nun kitabından bir alıntıdır. Bu resimde bir kod çözücü devreye bağlanmış veya kapıları görülmektedir. Buna göre kod çözücü giriş sayısına göre bütün olası sonuçları üretmekte ve bu sonuçları veya kapıları ile birleştirmektedir. Şekildeki kabloların bağlantısı bir 2 boyutlu diziyi andırmaktadır. Buna göre hangi satırın seçileceği kod çözücünün seçilen bacağına göre belirlenmektedir. Bu satırdaki kısa devre seçenekleri ise aşağıdaki veya kapılarının sonucu olarak okunmaktadır.

SORU 24: çoklayıcı (multiplexer)

Çok sayıdaki girişin tek bir giriş üzerinden taşınmasıdır. Amaç çok sayıdaki girişin (örneğin 4 giriş) tek bir çıkışa düşürülmesidir. Çalışma mantığı, anlık olarak tek bir girişi çıkışa vermek şeklindedir. Yani 4 girişten sadece bir tanesi çıkış ile kısa devre halindedir, diğer girişler ise ihmal edilir. Hangi girişin çıkışa verileceğini belirlemek için bir seçme işlemi yapılması gerekmektedir. Bu seçme işlemi yapan bitlere seçici bit(select bit) adı verilmektedir. Örneğin 4 girişi olan bir devre için (Girişler a,b,c ve d isminde olsun) 2 adet seçici bit gerekmektedir çünkü 4 bit ancak 2 bit ile adreslenebilir (2 üzeri 2) bu durumu gösteren temsili tablo aşağıda verilmiştir:

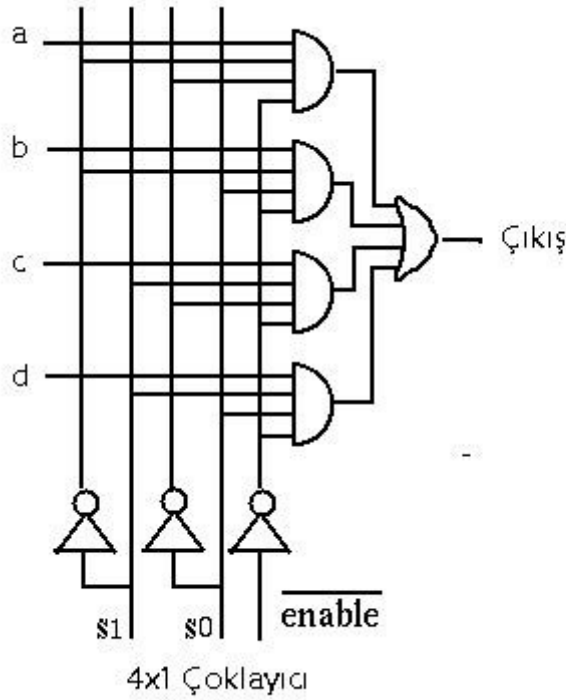
S1	S2	Ç
--	--	-
0	0	a
0	1	b
1	0	c
1	1	d

yukarıdaki tablonun çalıştığı devrenin görüntüsü aşağıda verilmiştir.



Yukarıdaki şeması verilen devrenin girişlerinden hangisini çıkış ile bağlanacağına seçici bitler

karar verirler. Bu devrenin tasarımı aşağıda verilmiştir:



Yukarıda tanımı verilen özellik kullanılarak devre tasarımında kısaltmaya gidilebilir. Buna göre karnaugh haritasında verilen her hücre tasarımın sonucunda bir girişin bağlanması ile yapılmaktadır. Bu özellik kullanılarak bir tam toplayıcıyı, kod çözücü devre yardımı ile tasarlayalım:

Öncelikle [tam toplayıcı devrenin](#) çalışmasını hatırlayalım:

[tam toplayıcı devrenin doğruluk çizelgesini](#) hatırlayalım (3 bitlik giriş için 2 bit çıkışı olan ve topladığı devre idi):

girişlerin	sayısal	değerlerinin	toplandığı	devre	idi):
A	B	C	E	T	
-	-	-	-	-	-
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	1	1

dolayısıyla yukarıda A B ve C girişlerinin toplam değerleri T ve E bitlerinde verilmiştir.

bu doğruluk çizelgesinin karnaugh haritası aşağıda verilmiştir:

T	BC				
A		00	01	11	10
0			1		1
1	1			1	

E	BC				
A		00	01	11	10
0				1	
1			1	1	1

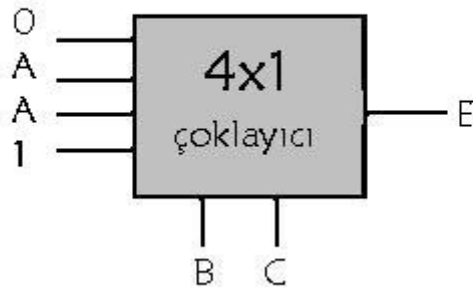
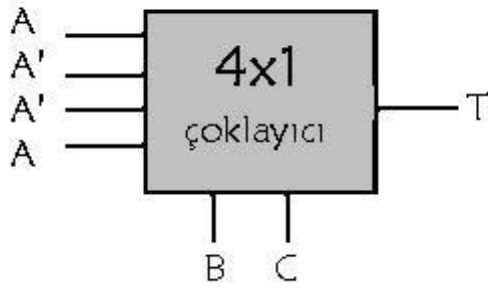
Şimdi bu karnaugh haritasının her sütunu için geçerli olan terimleri bulalım:

T	BC				
A		00	01	11	10
0			1		1
1	1			1	
		A	A'	A	A'

E	BC				
A		00	01	11	10
0				1	
1			1	1	1
		0	A	1	A

yukarıdaki tabloda, bir tam toplayıcının karnaugh haritasının üzerinde sütun bazlı olarak işlem yapılmıştır. Buna göre her sütunda (yain BC ikili ihtimali için) çıkan sonuç en alt satırda gösterilmiştir. Örneğin E biti için, BC ikilisi 11 olduğunda sonuç her zaman 1 çıkmaktadır (A bitinin sonuca bir etkisi yoktur) benzer şekilde yine E biti için BC ikilisi 0 olduğunda sonuç her zaman 0 olmaktadır (yine A bitinin sonuca bir etkisi yoktur) örneğin E biti için BC ikilis 01 olduğunda sonuç A bitine bağlıdır. Şayet A biti 1 ise sonuç 1 , A biti 0 ise sonuç 0 çıkmaktadır. Bu durumda da BC ikilisinin 01 olduğu durum için sonuç A'dır denilebilir. O halde yukarıda verilen bu özelliklerden faydalanarak bir çoklayıcı (multiplexer) devresi ile tam toplayıcı tasarlayalım.

Bu işlemden önce dikkat edilmesi gereken bir husu, karnaugh haritalarında sütun numaraları yazılırken 00, 01, 11, 10 sıralaması ile gitmesidir. Oysaki çoklayıcı devrenin doğruluk çizelgesine dikkat edilirse sıralama 00, 01, 10, 11 şeklinde gitmektedir. Bu yüzden karnough haritasındaki son iki sütun çoklayıcı devrede yer değiştirmektedir. Okuyucu buna dikkat etmelidir.



Yukarıdaki iki ayrı devrede iki ayrı çıkış değeri için çözüm yapılmıştır. Buna göre [kod çözücü](#) devre veya [“ve” ve “veya” kapıları ile tasarlanan bir tam toplayıcı](#) ile aynı işi yapan yukarıdaki devrede çoklayıcının bize sağlamış olduğu avantaj kullanılarak daha az elemanla tasarım yapılmıştır.

Çoklayıcı devreler günümüzde kullanılan pekçok devrenin temelinde bulunmaktadır. Örneğin ağ iletişiminde kullanılan HUB cihazının tasarımı basit bir multiplexerdir.

SORU 25: kod çözücü (decoder)

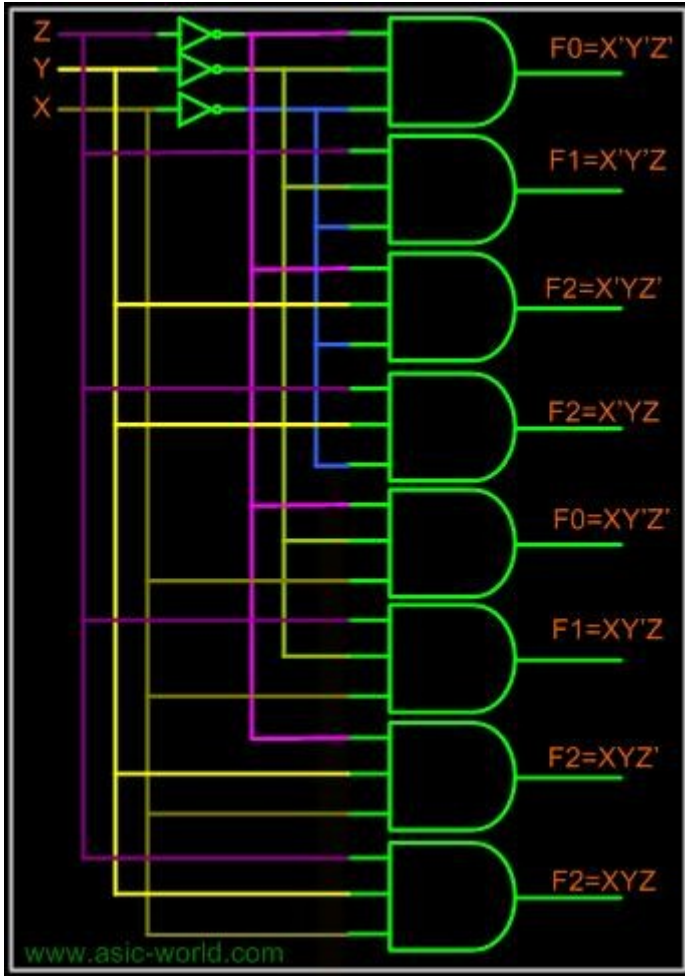
mantıksal devre tasarımının önemli parçalarından birisi olan kod çözücülerin çalışma mantığı giriş bitlerinin farklı bacaklara dağılmasıdır. Yani örneğin 2 girişli bir kod çözücünün 4 farklı çıkışı olur (2 üzeri 2) ve her çıkış sadece bir giriş ihtimali için çalışır. Daha basit anlatmak gerekirse aşağıdaki [doğruluk çizelgesini](#) inceleyelim:

A	B	a	b	c	d
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

yukarıdaki [doğruluk çizelgesinde](#) A ve B girişleri için 4 farklı çıkış değeri (a b c ve d) verilmiştir. Dikkat edilirse giriş değerlerinin alabileceği her ihtimal için farklı bir çıkış değeri 1 olmuş geri kalanlar 0 olmuştur. Bu durumda kod çözücülerin ana amacı farklı girişlerin mantıksal “ve” işlemini almaktır. Yani yukarıdaki çizelgede her satırda farklı bir ihtimalin “ve” sonucu görülmektedir. Bu durum aşağıdaki tabloda gösterilmiştir:

A	B	a	b	c	d
-	-	-	-	-	-
0	0	$A'B'$	-	->	a
0	1	$A'B$	-	->	b
1	0	AB'	-	->	c
1	1	AB	-	->	d

yukarıdaki tabloda her satırda mantıksal olarak farklı bir ihtimal gösterilmiş ve sonucun hangi çıkış bacağına görüldüğü verilmiştir. Buna göre 2x4lük bir kod çözücünün a bacağı her zaman $A'B'$ önermesini verir. Bir kod çözücünün kapılar ile tasarımı aşağıdaki şekildedir.



Bu özellik kullanılarak devre tasarımında kısaltmaya gidilebilir. Çünkü klasik yaklaşımda [karnaugh haritası](#) ile çizilen devrelerde bir “ve” kapısı grubu bir de “veya” kapısı grubu bulunmaktadır (önce ve kapısında eşleşen girişler daha sonra veya kapısı ile toplanmaktadır bkz. tam toplayıcı veya yarı toplayıcı) Bu yaklaşımındaki “ve” kapı grubunu kod çözücü ile kaldırarak sadece “veya” kapı grupları ile kod çözücünün çıkış bacaklarını birbirine bağlamak mümkündür. Çünkü yukarıdaki örnekte de görüleceği üzere girişlerin tamamının bütün muhtemel “ve” kapısı sonuçları kod çözücü ile bulunabilmektedir.

Örnek olarak bir [tam toplayıcı devrenin](#) kod çözücü ile tasarımı aşağıda verilmiştir:

[tam toplayıcı devrenin doğruluk çizelgesini](#) hatırlayalım (3 bitlik giriş için 2 bit çıkışı olan ve girişlerin sayısal değerlerinin toplandığı devre idi):

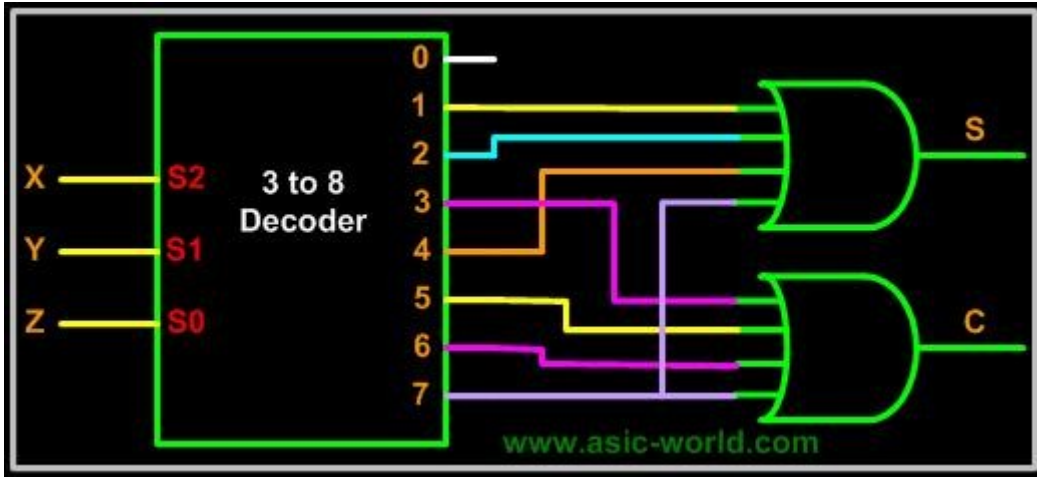
A	B	C	E	T
-	-	-	-	-
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

dolayısıyla yukarıda A B ve C girişlerinin toplam değerleri T ve E bitlerinde verilmiştir. bu doğruluk çizelgesinin karnaugh haritası aşağıda verilmiştir:

T	BC				
A		00	01	11	10
0			1		1
1	1	1		1	

E	BC				
A		00	01	11	10
0				1	
1		1	1	1	1

şimdi yukarıdaki haritaya ve doğruluk çizelgesine bakıldığında görülür ki aslında doğruluk çizelgesinin her satırı kod çözücünün farklı bir bacağıdır. Bu durumda tam toplayıcı devre aşağıdaki şekilde de tasarlanabilir:



Yukarıdaki devrede S, T bitine ve C ise E bitine karşılık gelmektedir. {bu devreyi çizip orjinal resim konulacak}

SORU 26: tam toplayıcı (full adder)

3 bitlik giriş değerlerinin (iki tabanındaki girişler) toplamını veren devredir. Buna göre A, B ve C girişleri için aşağıdaki tablo elde edilir. (aşağıdaki tablodaki + işareti önermeler arası veya ile karıştırılmamalıdır. + işareti toplamayı ifade eder)

A	B	C	E	T
-	-	-	-	-
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1 1 1 1 1				

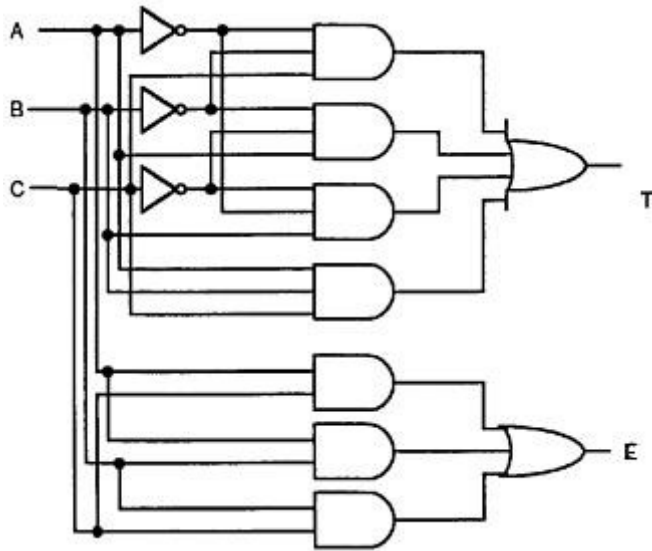
yukarıda verilen çizelgede A, B ve C sayılarının toplamı E ve T değerlerinde gösterilmiştir. Örneğin A 1, B 1 ve C 1 değerleri için (çizelgenin son satırı) toplam 3 olmaktadır ($1+1+1=3$) bu değer için ikilik tabandaki karşılığı 11'dir. 2 çıkış olmasının sebebi toplanan sayıların tek bit ile ifade edilememesinden dolayıdır. Yukarıda verilen toplam değerlerini veren devreyi tasarlarken E ve T ikillerini (Bit) ayrı ayrı düşünmek gerekir. Bu toplama işlemini yapan devrenin tasarımında karnaugh haritalarından faydalanılırsa T ve E ikilleri (bit) için aşağıdaki haritalar çizilebilir:

T	BC				
A		00	01	11	10
0			1		1
1	1			1	

E	BC				
A		00	01	11	10
0				1	
1		1	1	1	1

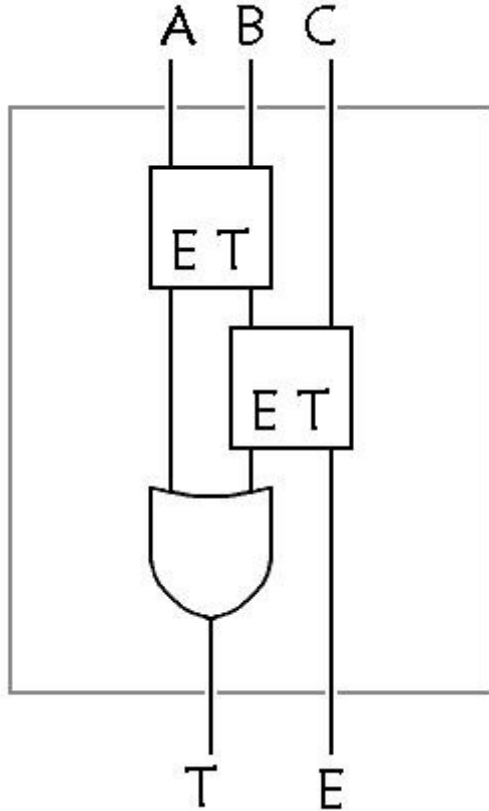
yukarıdaki haritada, üç giriş için toplam değerlerini veren iki farklı çıkış (T toplam ve E elde var) bitlerinin değerleri gösterilmektedir. Bu değerler arası komşuluklar kırmızı daire içerisinde alınmıştır. Bu 1lerden oluşan adalar sadeleştirildiğinde:
 $T = A'B'C + A'BC' + ABC + AB'C'$
 $E = AB + AC + BC$
olarak bulunur.

Bu devrenin tasarımı aşağıdaki şekilde verilmiştir:



Yukarıdaki devre “ve” ve “veya” kapıları kullanılarak yapılan devre tasarımıdır. Bu devre “özel veya” ve “yarım toplayıcı” devreleri kullanılarak da yapılabilir. Aşağıda bu devre tasarımları da verilmiştir:
 Bir tam toplayıcının yahut devreleri (“özel veya” (xor)) ile tasarımı

Bir tam toplayıcıyı iki adet yarım toplayıcı ile tasarlamak da mümkündür.



Tam toplayıcı aynı zamanda [kod çözücü \(decoder\)](#) ve [çoklayıcı \(multiplexer\)](#) ile de tasarlanabilir.

SORU 27: yarım toplayıcı (half adder)

ikilik tabanda verilen iki giriş değerini toplayan devredir. Buna göre A ve B girişleri için aşağıdaki tablo elde edilir. (aşağıdaki tablodaki + işareti önermeler arası veya ile karıştırılmamalıdır. + işareti toplamayı ifade eder)

A	B	E	T
-	-	-	-
0	0	0	0
0	1	0	1
1	0	0	1
1 1 1 0			

yukarıda verilen çizelgede A ve B sayılarının toplamı E ve T değerlerinde gösterilmiştir. Örneğin A 1 ve B 1 değerleri için toplam 2 olmaktadır ($1+1=2$) bu değer in ikilik tabandaki karşılığı 10'dır. 2 çıkış olmasının sebebi toplanan sayıların tek bit ile ifade edilememesindendir.

Yukarıda verilen toplam değerlerini veren devreyi tasarlarken E ve T ikillerini (Bit) ayrı ayrı düşünmek gerekir. Bu toplama işlemini yapan devrenin tasarımında karnaugh haritalarından faydalanılırsa T ve E ikilleri (bit) için aşağıdaki haritalar çizilebilir:

T	A
B	
0	0 1
1	0 1
1	1

E	A
B	
0	0 1
1	0 1
1	1

yukarıda verilen şekilde T ve E bitlerinin çıkış değerleri karnaugh haritası üzerinde işaretlenmiştir. Komşuluk durumu olan 1 değeri olmadığı için iki bit değeri de sadeleştirilemeden aşağıdaki önermeler halinde yazılmak durumundadır:

$$T=AB'+A'B$$

$$E=AB$$

Bu devrenin tasarımı aşağıdaki şekilde verilmiştir:

yukarıdaki şekilde, A ve B girişleri için T ve E bitlerini veren örnek yarı toplayıcı şekli verilmiştir.

Şayet dikkat edilirse E kapsının sonuç değerleri [yahut işlemi \("özel veya" \(xor\)\)](#) sonuçları ile aynıdır. Buna göre E devresinin "ve" kapısı , T devresinin ise "özel veya" (xor) kapısı olduğu görülür.

SORU 28: doğruluk çizelgesi (truth table)

Mantıksal işlemlerin tahlil edilmesinde kullanılan önemli âletlerden birisidir. Buna göre herhangi bir mantıksal önermenin (kaziye) muhtemel sonuçları bu tablo vasıtasıyla gösterilebilir. Çalışma şekli önermede (kaziye) bulunan giriş değerlerinin bütün muhtemel girişleri için bir satır oluşturmak ve sonucunu ayrı ayrı hesaplamak şeklinde yapılır. Misal olarak çok kullanılan “ve” işlemini mütâlaa edelim. Bu işlemin iki önerme için bir bağlayıcılık özelliği bulunmaktadır ve bu işlem 2 farklı önermenin (kaziye) aynı anda gerçekleşmesi durumunu doğru, diğer durumları hatalı kabul eder. Aşağıda iki farklı önerme verilmiştir:

1. üniversitede öğrenci olmak
2. devre analizi yapabilmek

bu durum aşağıdaki tabloda ve bağlacı ile ifade edilmiştir:

üöo	day	VE (And)
0	0	0
0	1	0
1	0	0
1	1	1

yukarıdaki tabloda, 1. önerme üöo (üniversitede öğrenci olmak) ve 2. önerme day (devre analizi yapabilmek) şeklinde ifade edilmiştir. Buna göre yukarıdaki tablonun ilk satırının anlamı:

üniversitede öğrenci olmak ve devre analizi yapabilmek (ikisi de 0 olduğu için), olarak yorumlanabilir. Sonuç ise 0'dır yani üniversitede öğrenci olmak ve devre analizi yapabilmek bu örnek için olumsuzdur.

Yani yukarıdaki tabloda her satırda bir ihtimal incelenmiş, neticede ise bütün ihtimaller tek bir tabloda gösterilmiştir. İşte bu tabloya doğruluk çizelgesi (truth table) denilmektedir. Benzer şekilde aynı önermeler için “veya” işlemi incelenirse:

1. üniversitede öğrenci olmak
2. devre analizi yapabilmek

bu durum aşağıdaki tabloda veya bağlacı ile ifade edilmiştir:

üöo	day	VEYA (Or)
0	0	0
0	1	1
1	0	1
1	1	1

Yukarıda anlatılanlara göre herhangi bir önermenin doğruluk çizelgesi çıkarılabilir. Örneğin yukarıda anlatılmış olan “ve” ve “veya” bağlaçları (âtıfları) kullanılarak aşağıdaki F değerinin doğruluk çizelgesi inşâ edilebilir:

F=	A'B	+	AB	eşitliği	için:
A			B		F
-			-		-
0			0		0
0			1		1

1
1 1 1

0

0

Yukarıdaki 2 girişli denklemler için oluşturulan doğruluk çizelgesi nasıl bütün olası sonuçları gösteriyorsa, aynı durum daha fazla girişi olan örnekler için de kullanılabilir. Örneğin aşağıda 3 farklı giriş için (p q r) doğruluk çizelgesi verilmiştir:







p	q	r	qr	p+(qr)
1	1	1	1	1
1	1	0	0	1
1	0	1	0	1
1	0	0	0	1
0	1	1	1	1
0	1	0	0	0
0	0	1	0	0
0	0	0	0	0

Yukarıdaki çizelgede dikkat edilecek olan husus 3 farklı giriş için toplam 8 muhtemel (2 üzeri 3) hal olmasıdır ve her hal için bir satır yazılmasıdır. Dikkat edilirse hiç bir satır diğerinin tekrarı değildir. Yukarıdaki tablo aşağıdaki şekilde de yazılabilir:

p	q	r	qr	p+(qr)
D	D	D	D	D
D	D	Y	Y	D
D	Y	D	Y	D
D	Y	Y	Y	D
Y	D	D	D	D
Y	D	Y	Y	Y
Y	Y	D	Y	Y
Y	Y	Y	Y	Y

Yukarıdaki tabloda D harfi doğru, Y harfi ise Yanlış sonuçları ifade etmektedir. Yani önermelerin doğru ve yanlışlığına göre sonucun nasıl olduğu bu tablodan görülebilir.

Daha fazla bilgi için

-  [Karnaugh haritaları](#)
-  Yarım toplayıcı (half adder)
-  Tam toplayıcı (full adder)
-  çıkarıcı (subtractor)
-  çoğunluk biti (majority bit)
-  büyüktür (greater than)

SORU 29: karnaugh haritası (karnaugh map)

Bool cebirinde verilen mantıksal gösterimleri sadeleştirmek için kullanılan haritadır. Buna göre bir mantıksal devrenin eleman sayısını azaltmak için de kullanılabilir. Örneğin 3 girişli (3 adet farklı binary (ikili) giriş değeri (0 veya 1 olabilen)) devrede kullanılan “ve” ve “veya” kapılarının sayısını azaltabiliriz.

Bu yöntemde giriş değerlerinin alabileceği bütün alternatifler bir tablo üzerinde gösterilerek bu alternatiflerden hangilerinde çıkış olması isteniyorsa bu değerlere 1 yazılır. Sonuçta yazılı olan 1 rakamları arasında komşuluk incelemesi yapılarak sadeleştirilirler.

[flashvideo file=http://www.bilgisayarkavramlari.com/wp-content/uploads/karnaugh.flv /]

Örneğin iki girişi olan bir devredeki girişler A ve B olsun ve istenilen çıkış değeri F aşağıdaki şekilde verilmiş olsun:

$$F = A'B + AB$$

Yukarıdaki terime bakıldığında, bool cebirinin temel özellikleri kullanılarak (ortak parantez):

$$F = (A' + A)B$$

sonucuna varılabilir. Bu eşitlik de sadeleştirilerek (bir girişin tersiyle veya 1'dir) $F = B$

sonucuna varılabilir. Demek ki ilk verilen eşitlikte A terimleri fazla terimlerdi. Ancak bunu görmek her zaman kolay olmamaktadır. Özellikle de giriş sayıları arttıkça çok sayıda işlem yapmak ve sade hali elde etmek zahmetli olmaktadır. Ve ne kadar sadeleşirse sadeleşsin en sade forma erişilip erişilmediği her zaman muamma olarak kalmaktadır. Kesin ve her durum için geçerli olması açısından bu sadeleştirme işlemi karnaugh haritaları ile aşağıdaki şekilde yapılabilir:

1) Öncelikle istenilen devrenin [doğruluk çizelgesi \(truth table\)](#) çizilir.

A	B	F
-	-	-
0	0	0
0	1	1
1	0	0
1	1	1

yukarıdaki tablo $F = A'B + AB$ eşitliğinin [doğruluk çizelgesidir \(truth table\)](#). Bu tabloda dikkat edileceği üzere giriş değerleri olan A ve B için alınabilecek bütün alternatifler listelenmiş ve her alternatif için F çıkış değerinin karşılığı yazılmıştır. Yani örneğin tablonun ilk satırı için $F = A'B + AB$ eşitliğinde A yerine 0 ve B yerine 0 yazılarak sonuç hesaplanmış ve tabloya yazılmıştır.

Sonuçta çıkan tabloyu karnaugh haritası olarak çizecek olursak aşağıdaki şekilde bir tablo bulunur:

	B	
A	0	1
	0	1
	1	1

Bu tabloda her hücrenin anlamı [doğruluk çizelgesindeki \(truth table\)](#) bir değere karşılık gelmesidir. Yani [doğruluk çizelgesini \(truth table\)](#) yeniden çizecek olursak bu tablodaki değerlerin karşılığı olan hücre numaraları aşağıdaki şekilde listelenebilir:

A	B	F	Tablo	Adresi
-	-	-	-	-----
0	0	0	0	0
0	1	1	1	1
1	0	0	0	2
1 1 1 3				

Yukarıdaki tabloda, tablo adresleri aslıdan A ve B bitleri ikilik tabanda yanyana yazıldığında elde edilen 2 haneli ikilik tabandaki sayıların 10luk tabandaki karşılıklarıdır. Aşağıda, karnaugh haritasındaki her hücrenin ikili ve onlu tabandaki adresleri verilmiştir:

	B	
A	0	1
0	00	01
1	10	11

2li tabandaki adresler. (bu adresler basitçe satır ve sütun karşılıklarının okunması ile elde edilir.)

	B	
A	0	1
0	0	1
1	2	3

10luk tabandaki adresler. (bu adresler basitçe bir önceki tablodaki ikilik sayıların 10luk sisteme çevrilmesi ile elde edilir.)

Dolayısıyla yukarıda bulunan [doğruluk çizelgesini](#) yukarıdaki adreslere göre yerleştirmek de mümkündür. Yani [doğruluk çizelgesindeki](#) adres değeri 1, haritadaki 1 numaralı adrese şeklinde.

Şimdi örnek problem olan $F = A'B + AB$ eşitliğini karnaugh haritası ile çözmek istediğimizi düşünelim. Yapılması gereken basitçe aşağıdaki şekilde olduğu gibi haritasını çizmek ve komşu olan 1 leri işaretlemektir. (daha ileride de anlatılacağı gibi komşuluk aranırken 2'nin üstü olan 2 4 8 16 gibi sayıdaki 1'lerin komşuluğuna bakılır)

	B	
	0	1
A	0	1
	1	1

Bu haritada komşu olan 1'ler işaretlendikten sonra ortak ortaya çıkan adanın terimleri okunur. Buna göre bu ada için A değeri değişkenlik göstermektedir. Yani çember içine aldığımız adamız, A 1 iken ve 0 iken geçerlidir. Bu durumda sonuçta A terimi olması beklenemez çünkü A'nın aldığı terimler sonucu etkilememektedir. B terimi incelendiğinde, bu terimin bütün ada için 1 olduğu görülür. Öyleyse sonuç B'dir denilebilir.

Karnaugh haritalarını 2'den fazla giriş için de kullanmak mümkündür. (Şimdiye kadar olanlar 2lik

Örneğin 3 giriş için aşağıdaki harita kullanılabilir:

	BC			
A	00	01	11	10
0	000	001	011	010
1	100	101	111	110

yukarıdaki şekilde 3 girişli (A,B ve C girişleri için) bir karnaugh haritası verilmiştir. Bu haritada her girişin alabileceği değerler gösterilmiştir. (giriş değerleri ikili tabanda olduğu için 2 alternatif olan 0 veya 1 değerlerini alabilirler) Buna göre üç giriş için 8 (2 üzeri 3) alternatif

bulunmaktadır, ve her alternatif tabloda gösterilmiştir. Yukarıdaki tablo'nun mantıksal değerler için gösterimi aşağıda verilmiştir:

		BC			
A		00	01	11	10
	0	$A'B'C'$	$A'B'C$	$A'BC$	$A'BC'$
	1	$AB'C'$	$AB'C$	ABC	ABC'

yukarıdaki şekilde 3 girişli (A,B ve C girişleri için) bir karnaugh haritası verilmiştir. Bu haritada her girişin alabileceği değerler gösterilmiştir. (giriş değerleri ikili tabanda olduğu için 2 alternatif olan 0 veya 1 değerlerini alabilirler bu değerlerin anlamı o girişin kendisinin veya tersinin alınması durumunda 1 çıkacağıdır) Buna göre üç giriş için 8 (2 üzeri 3) alternatif bulunmaktadır, ve her alternatif tabloda gösterilmiştir. Bu hücrelerin her birisinde ilgili girişin karşılığı olan mantıksal önerme yer almaktadır. Örneğin tablonun sol üst köşesinde bütün girişler 0 değerinde olduğu için bu hücrenin 1 olmasını sağlayacak mantıksal önerme mutlaka bütün girişlerin tersinin ve mantıksal bağlacı ile bağlanmış hali olmalıdır.

Aşağıda giriş sayısının 4 olması durumunda karnaugh haritasının mantıksal gösterimleri verilmiştir:

		CD			
AB		00	01	11	10
	00	$A'B'C'D'$	$A'B'C'D$	$A'B'CD$	$A'B'CD'$
	01	$A'B C'D'$	$A'BC'D$	$A'BCD$	$A'BCD'$
	11	$AB C'D'$	$ABC'D$	$ABCD$	$AB CD'$
	10	$AB' C'D'$	$AB'C'D$	$AB'CD$	$AB' CD'$

Aşağıdaki tablo yukarıdaki şekilde düzeltildi. Aşağıdaki tablo hatalı ve yukarıdaki tablo doğrudur. Detay için yorum kısmına bakabilirsiniz. Uyarısı için yekta beye teşekkür ederim.

		CD			
AB		00	01	11	10
	00	$A'B'C'D'$	$A'B'C'D$	$A'B'CD$	$A'B'CD'$
	01	$A'B C'D'$	$A'BC'D$	$A'BCD$	$A'BCD'$
	11	$AB C'D'$	$ABC'D$	$ABCD$	$AB CD'$
	10	$AB' C'D'$	$AB'C'D$	$AB'CD$	$AB' CD'$

Karnaugh Haritasında asgari terimlerin kullanımı (Minterms)

Karnaugh haritalarında kullanılan ve bir devredeki girdilerin daha hızlı gösterilmesini sağlayan bir notasyondur. Bu yöntem karnaugh haritasındaki her hücreye bir numara verme esasına dayanır. Örneğin aşağıdaki gösterimi ele alalım:

$$F(A,B,C) = \Sigma 2, 3,4,6,7$$

Bu gösterimde anlatılan, karnaugh tablosundaki 4 terimin sonuçta olması gerektiğidir. Bu terimlerin değerleri haritadaki adreslerden elde edilir:

	BC			
A	00	01	11	10
0	000	001	011	010
1	100	101	111	110

Yukarıdaki tablo, ilgili hücrelerin içerdiği satır ve sütun numarasının birleşimidir. Bu tablodaki ikilik tabanda olan sayıları onluk tabana çevirecek olursak:

A/BC	00	01	11	10
0	0	1	3	2
1	4	5	7	6

Bu tabloda verilen sayıları işaretlediğimizde :

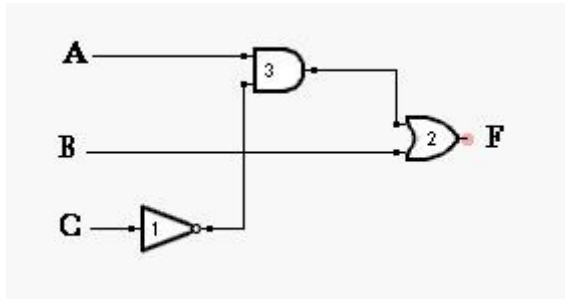
A/BC	00	01	11	10
0	0	1	3	2
1	4	5	7	6

Sonuç olarak sadeleştirilmiş devre

$$F(A,B,C) = B + AC'$$

olarak bulunur.

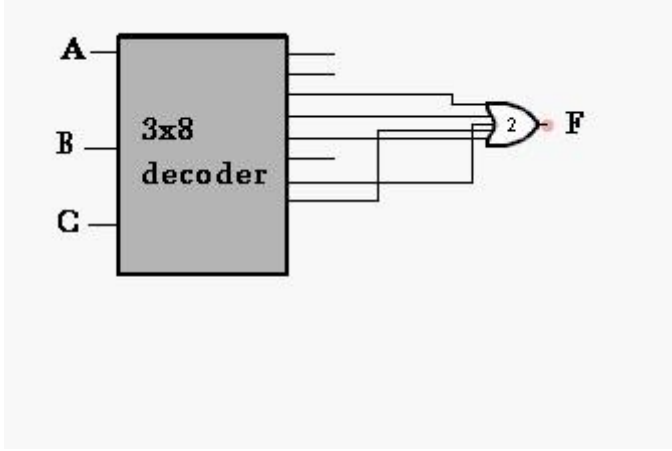
Bu devre çizildiğinde aşağıdaki şekilde bir sonuç elde edilir:



Yukarıda bahsedilen asgariterimler (minterms) aslında decoder kullanılan bir devre için çok daha hızlı sonuca ulaşılmasını sağlar.

Örnek bir decoder devresinde hangi sonuçların or (veya) kapısı ile bağlanacağını belirler.

Örneğin yukarıdaki verilen asgariterimleri (minterms) ele alalım ve aynı devreyi bir dekoder yardımı ile çizmeye çalışalım:



Yukarıdaki devrede görüldüğü üzere decoder çıkışının asgariterim numaraları alınmıştır. (yani çıkışlardan 0,1,5 boş bırakılmış ve 2,3,4,6,7 bağlanmıştır.)

SORU 30: de morgan kuralı (de morgan rule)

mantıksal devre tasarımı açısından oldukça kullanışlı olan bu kurala göre :

$$(x \text{ ve } y)' \Leftrightarrow x' \text{ veya } y'$$
$$(x \text{ veya } y)' \Leftrightarrow x' \text{ ve } y'$$

yani ve kapısıyla bağlı bir devrenin olumsuzu, devrenin giriş değerlerinin olumsuzlarının veyası şeklinde yazılabilir. Benzer şekilde, veya kapısı ile bağlı girişlerin olumsuzu, girişlerin olumsuzlarının ve kapısıyla bağlanmış halinde yazılabilir.

Bu kuralın kümelere uygulanmış halini de aşağıdaki şekilde yazabiliriz:

$$(A \cup B)' = A' \cap B' \text{ ve benzer şekilde } (A \cap B)' = A' \cup B' \text{ dir}$$

SORU 31: veya kapısı (or gate)

2 farklı önermeden birisinin gerçekleşmesi durumunu inceleyen kapıdır. Mantıkta buluna veya bağlacı şeklinde çalışır. Bu durumu aşağıdaki örnek üzerinde inceleyelim. Aşağıda iki farklı [önerme](#) verilmiştir:

1. _____ üniversitede _____ öğrenci _____ olmak
2. devre analizi yapabilmek

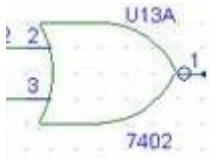
bu durum aşağıdaki tabloda veya bağlacı ile ifade edilmiştir:

üöo	day	VEYA (Or)
0	0	0
0	1	1
1	0	1
1	1	1

yukarıdaki tabloda, 1. önerme üöo (üniversitede öğrenci olmak) ve 2. önerme day (devre analizi yapabilmek) şeklinde ifade edilmiştir. Buna göre yukarıdaki tablonun ilk satırının anlamı:

üniversitede öğrenci olmak ve devre analizi yapabilmek (ikisi de 0 olduğu için), olarak yorumlanabilir. Sonuç ise 0'dır yani üniversitede öğrenci olmak ve devre analizi yapabilmek bu örnek için olumsuzdur.

Veya kapıları [mantık devrelerinde](#) aşağıdaki şekilde ifade edilir:



bu şekilde basit bir iki girişli veya kapısı (or gate) gösterilmiştir. Kapının iki girişi (sol taraftan) ve bir çıkışı (sağ taraftan) bulunmaktadır.

SORU 32: Ve kapısı (and gate)

2 farklı önermenin aynı anda gerçekleşmesi durumunu inceleyen kapıdır. Mantıkta buluna ve bağlacı şeklinde çalışır. Bu durumu aşağıdaki örnek üzerinde inceleyelim. Aşağıda iki farklı [önerme](#) verilmiştir:

1. üniversitede öğrenci olmak
2. devre analizi yapabilmek

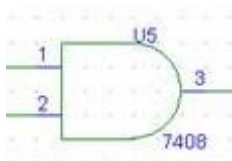
bu durum aşağıdaki tabloda ve bağlacı ile ifade edilmiştir:

üöo	day	VE (And)
0	0	0
0	1	0
1	0	0
1	1	1

yukarıdaki tabloda, 1. önerme üöo (üniversitede öğrenci olmak) ve 2. önerme day (devre analizi yapabilmek) şeklinde ifade edilmiştir. Buna göre yukarıdaki tablonun ilk satırının anlamı:

üniversitede öğrenci olmak ve devre analizi yapabilmek (ikisi de 0 olduğu için), olarak yorumlanabilir. Sonuç ise 0'dır yani üniversitede öğrenci olmak ve devre analizi yapabilmek bu örnek için olumsuzdur.

Ve kapıları [mantık devrelerinde](#) aşağıdaki şekilde ifade edilir:



bu şekilde basit bir iki girişli ve kapısı (and gate) gösterilmiştir. Kapının iki girişi (sol taraftan) ve bir çıkışı (sağ taraftan) bulunmaktadır.

SORU 33: Önermeler (kaziye) Mantığı (Propositional Logic)

önerme mantık bilimi açısından anlam taşıyan en ufak olgudur. Örneğin: “insanın iki gözü vardır” bir önermedir. Bu önermelerin gerçek hayattan alınması zorunlu değildir.

Örnenin sonucu, önerildiği gibi gerçekleşirse bu durumda önermenin sonucu olumludur. Aksi durumda, yani önerildiği gibi bir sonuçla bitmezse bu durumda önermenin sonucu olumsuzdur.

Önermenin						sonucu:
Doğru	anlamli	ise	mantiksa	olarak	1	(logic 1)
Yanlıř	anlamli	ise	mantıksal	olarak	0	(logic 0)

ile ifade edilir.

SORU 34: Bir tümleyeni

Konunun diğeri isimleri : (1 tümleyeni, İşaretili sayı gösterimi, Ones' Complement, 1's Complement, Signed number representations)

ikilik tabandaki bir sayının 1 tümleyeni her sayının tersidir. Örneğin sayı:

10110011

olarak verilmiş olsun. Bu sayının 1 tümleyeni:

01001100

olarak bulunur.

Bu bilgi [2 tümleyeni](#) nin hesabında da kullanılmaktadır.

Bir tümleyeni aynı zamanda sayının eksi değeri olarak gösterilmesine de yaramaktadır. Aşağıdaki tabloda sayıların alabileceği değerler ve bu sayıların ikili ve onluk tabandaki gösterimleri verilmiştir:

İkili sayı	1 tümleyeninin 10luk	Yönsüz sayı
00000000	0	0
00000001	1	1
...
01111101	125	125
01111110	126	126
01111111	127	127
10000000	-127	128
10000001	-126	129
10000010	-125	130
...
11111110	-1	254
11111111	-0	255

yukarıdaki tabloda ikilik sistemde bir takım sayılar verilmiştir. Tablonun ikinci kolonunda bu sayıları 1 tümleyeni olarak yorumladığımızda 10luk sistemdeki karşılıkları, üçüncü kolonda ise bu sayıları normal birer ikilik sayı gibi görüp, 10luk sisteme çevirince çıkan karşılıkları verilmiştir.

Buna göre 0 sayısının bütün bitlerinin tersi -0 veya örneğin 10 sayısının bütün bitlerinin tam tersi -10 olmaktadır.

Bu ayrımı ilk bit belirlemektedir bu bit'e sign bit (yön biti) de denilmektedir.

SORU 35: İki tümleyeni

Konunun diğer başlıkları: 2 tümleyeni, two's complement

Bilgisayar bilimlerinde, sayılar genelde ikilik tabanda tutulmaktadır. Değerleri ikilik tabanda göstermenin bir devamı olarak eksi sayı ve artı sayıları da ayırmak gerekmektedir. bir tümleyeni gibi iki tümleyeni de eksi sayıları gösterim biçimlerinden birisidir. iki tümleyenini almak için önce bir tümleyeni alınır ardında sayıya ikilik tabanda 1 eklenir.

Örneğin

11011001

sayısının bir tümleyeni aşağıda verilmiştir:

00100110

bu sayıya 1 eklenerek, iki tümleyeni elde edilir:

00100111

Bu sayı aynı zamanda orjinal sayı olan 11011001 sayısının da negatif gösterimi olarak kullanılabilir.

Bunu bir örnek ile göstermek gerekirse, aşağıdaki çıkarma işlemini ele alalım:

11001001

10110101

-

00010100

bilindiği üzere aslında çıkarma işlemini, çıkarılan sayının negatifini alıp toplama olarak da yorumlayabiliriz.

Bu durumu aynı örnek için tecrübe edelim. Öncelikle çıkarılan sayı olan 10110101 sayısının negatifini alalım, yani iki tümleyenini: 01001010 sayısı elde edilir. Şimdi bu sayının gerçekten negatif olduğunu yukarıdaki örneği toplamaya çevirerek görelim:

```
11001001
01001010
+
-----
100010100
```

Görüldüğü üzere elde edilen sonucun başında bulunan 1 atılırsa, ilk işlemde çıkan sonuç ile aynıdır.