YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

C PROGRAMLAMA DİLİNDE BİT İŞLEMCİLERİNİ KULLANARAK VERİ KARŞILAŞTIRMAK

Matematik Müh. Ömer KUTLAY

Matematik Mühendisliği Ana Bilim Dalında
hazırlanan

YÜKSEK LİSANS TEZİ

Tez Danışmanı : Prof. Dr. Abdüssamet MARŞOĞLU

İSTANBUL, 1996

İÇİNDEKİLER

TEŞEKKÜR

Tez konusunun seçiminde ve tezi hazırlamamda yardımlarını esirgemeyen değerli Hocam Sayın Prof. Dr. Abdüssamet MARŞOĞLU ' na ve Y.T.Ü Araştırma Laboratuvarı ' nda çalışma imkanı sağlayan Matematik Mühendisliği Bölüm Başkanı değerli Hocam Sayın Prof. Tahir Şişman ' a teşekkür ederim.

Ömer KUTLAY

İstanbul , 1996

ÖZET

Bilgisayar alanındaki gelişmeler, hızla gelişen günümüz teknoloji sinde şüphesiz önemli bir alan teşkil ediyor. Gün geçtikçe daha iyi özelliklere sahip,çok daha değişik amaçlara hitap eden bilgisayarlar tasarlanıyor. Daha az maliyet, daha az donanım karmaşası ve optimum tatmin anlayışı ile tasarımcılar her geçen gün piyasaya yeni ürünler sunuyorlar. Bu ürünler üzerinde öncekilere göre çok daha iyi özellikte birimler mevcut.Örneğin ,işlemci ünitesi üzerinde sıkça çalışılan bölümlerden biri. Aynı işlem makinalardaki tasarıma bağlı olarak farklı ..biçimlerde icra edilebiliyor. Bir karşılaştırma işlemi bazı bilgisayarlarda hardware olarak yapılırken bazılarında da software destekli yapılıyor. Bu nedenle çok çeşitli karşılaştırıcılar mevcut .

Tezimde , tasarlanmış bir asenkron karşılaştırıcının [6], software olarak benzetimini gerçekleştirdim. Karşılaştırıcıdaki temel ilke eşitsizlik halinde küçük olan sayıyı göstermesidir. En yüksek mertebeli bit 'ten başlayarak karşılaştırma yapılıyor ve eşitsizlik halinde sonuç elde ediliyor. Bu durumda diğer bit 'ler için yapılacak işlem iptal ediliyor. İki devre bu işlemler için tasarlanmış diğer devre ise karakter katarlarındaki küçük-büyük harf ayrımını kaldırmak amacıyla küçük harfleri büyük harflerle değiştirmek amacı ile tasarlanmış.

Karşılaştırıcıda gerçekleştirilen tüm işlemleri C dilini kullanarak hazırladığım programa tam olarak adapte ettim. Donanımda, özellikle XOR ve AND kapıları kullanılmış olmasından,C dilinde bu işlemleri gerçekleştirebileceğim bit tabanlı işlemcilerin olması bu dili tercih sebebim oldu. Tamsayı,Float ,Bcd ve Karakter tipindeki verileri özelliklerine bağlı olarak tanımladığım, register ' ları temsil eden karakter katarlarına yükledim. Karşılaştırmayı katarların yüksek mertebeli bit 'lerinden başlayarak tasarımdaki mantık çerçevesinde gerçekleştirdim.

SUMMARY

Developings of the computer science have an important part of our technology. The computers , which have better quality and respond to different aims, are projected day by day. Sure , the aim of the experts is to make a new unit that costs less money and has less hardware complication. The new computers have better qualified features than the old ones. Processing unit is the one of the units on that many works are made. The same process can be executed by different ways depend on the computer. In some computers , the comparison process is executed by hardware and in some computers by software. This is because there are many different types of comparators.

In my project, I simulated the asynchronous comparator, which was projected by MARŞOĞLU, Abdüssamet (1995), using the C programming language. Design emphasizes inequality and indicates smaller one. When the result is obtained, there is no need to compare the subsequent bit pairs. Two circuits work to execute these processes.The third one converts smaller letter to capital letter.

I just simulated all the processes executed in the comparator . I prefered using the C programming language because it has special operators, XOR and AND that are main features of the hardware. Characters and the types of numbers which are Integers,Floating Point Data ,BCD Numbers are stored in the character strings which represent the related registers. Comparison starts from the highest order bit pair.

BÖLÜM 1. GİRİŞ

Bilgisayarda matematiksel ve mantıksal işlemlerin gerçekleştirildiği bir işlem ünitesi mevcuttur (ALU). Bu işlemler mikro bilgisayarlarda software olarak gerçekleştirilir.Makinaya bağlı olarak 8,16 ve 32 bit 'lik hazırlanmış işlemcilerde (64 bit de mevcut) veri tipi ve niteliği ne olursa olsun işlem ,tasarlanan işlemci içerisinde mevcut bit üzerinde gerçekleştirilir. Yani , 32 bit ' lik makinada 2 bit ' lik bir veri dahi 32 bit içerisinde işlem görür. Tabii ki bu, işlem süresi açısından bir dezavantaj olduğu için tasarımcılar daha gelişmiş bilgisayarda yeni işlemciler düşünmüşler ve bu amaçla da software(yazılım) tasarımdan ziyade hardware(donanım) tasarıma yönelmişlerdir.Günümüz gelişmiş bilgisayarlarındaki bu tasarımlar işlem süresini oldukça düşürmüştür.

Karşılaştırmanın software olarak yapıldığı mikro bilgisayarlarda, statu bit veya durum_kod bit ' leri adı verilen bit 'ler sözkonusudur. Bunlar olası durumların analizi amacıyla merkezi işlem ünitesindeki bir register ' da saklanırlar. Bu register, matematiksel mantık işlemlerinin gerçekleştirildiği devre (ALU : Aritmetic Logic Unit) ile beraber işlem görür. ALU, merkezi işlem ünitesi (CPU) içindedir. Statu bit' leri sırasıyla C, S , Z ve V ile temsil edilirler ve statu register ' ında saklıdırlar. Bit' lerin işlem görüp görmemesi ALU ' dan alınan sonuca bağlıdır.Buna göre ;

1) Taşınan bit (carry bit) sözkonusu ise **C** aktif (yani 1) , aksi durumda pasif (yani 0) yapılır. ALU ' dan taşıma ile ilgili çıkış C aktif iken "1",pasif iken "0" dır.

2) **S** , en yüksek mertebeli bit "1" ise aktif, "0" ise pasif yapılır. Bu bit sonucun pozitif veya negatif olduğunu açıklar.

3) **Z** , sonuç 0(sıfır) ise aktif , aksi durumda pasif yapılır.

4) **V** , ALU 'nun sahip olduğu bit sayısına bağlı olarak işlem sırasında bir taşma olursa aktif, aksi halde pasif yapılır. Örneğin ,8 bit 'lik bir ALU için sonuç 127 ' den büyük veya -128 'den küçük olursa ,aktiflik sözkonusudur.

Aşağıdaki tablo statu bit ' lerinin hangi durumlarda ne değerini aldıklarını göstermektedir:

Dallanma Şartı	Statu Bit ' i
Sonuç 0	Z=1
Sonuç 1	Z=0
Carry Bit Var	C=1
Carry Bit Yok	C=0
Sonuç Pozitif	S=0
Sonuç Negatif	S=1
Taşma Var	V=1
Taşma Yok	V=0

Karşılaştırma işleminden kasıt, aslında iki sayının birbirinden çıkartılmasından başka birşey değildir. İşlem sonucuna bağlı olarak statu bit ' lerinin değerlendirilmesi sözkunusudur. Bazı bilgisayarlar karşılaştırmadan hemen sonra uygulanan şartlı dallanma operasyonlarına ihtiyaç duyar. Test edilecek özel şartlar ,iki sayının işaretli olup olmamasına bağlıdır.

8 bit ' lik bir aritmetik işlem ünitesine yerleştirilecek en büyük işaretsiz sayı 255 ' dir. Bu sınır ,işaretli sayılar için -128 ile +127 arasındadır. Çıkartma işlemi sayıların işaretli veya işaretsiz olmasına göre özel bir yapı almaz Her iki durumda işlem aynıdır. Çıkartılacak sayının 2 ' ye komplementi alınır ve ilk sayıyla toplanır. Örneğin , A=1 1 1 1 0 0 0 0 ve B=0 0 0 1 0 10 0 olsun. ALU ' da gerçekleşen işlemler aşağıdaki gibidir;

	A :	1 1 1 1 0 0 0 0	
$\overline{B}+1$	: +	1 1 1 0 1 1 0 0	
A - B	:	1 1 0 1 1 1 0 0	C=1, S=1, V=0, Z=0

Toplama işleminden oluşan carry bit ' i C ' nin "1" değerini almasını sağlar. Soldan ilk bit 'in "1" olması S ' yi aktif yapar. Herhangi bir taşma sözkonusu olmadığından V=0 'dır. Sonuç 0 ' dan farklı olduğu için Z=0 olur. Verilen iki bilgi işaretsiz olarak düşünüldüğünde A sayısı 240 ve B de 20 olduğundan, sonuç decimal olarak 240 - 20 = 220 , binary olarak 1 1 0 1 1 1 0 0 'dır.240 > 20 olduğundan A>B ve A≠B ilişkileri mevcuttur. Bu iki ilişki aynı zamanda C 'nin 1 ve Z ' nin de 0 olmasından çıkarılır.

Sayıların işaretli olduğu düşünülüp A için -16 ve B için de +20 alınırsa , bu durumda çıkarma sonucu desimal olarak (-16) - (+20) = -36 , binary olarak 1 1 0 1 1 1 0 0 dir.

$$
\begin{aligned}
A: \quad & 11110000 = -16 \\
\overline{B}: \quad & 11101011 \\
+ & \ 00000001 \\
& \text{---------} \\
& 11101100 = -20 \\
A-B: \quad & 11011100 = -36
\end{aligned}
$$

Bu sonuç statu bit ' lerine nagatiflikten S=1, taşma olmamasından V=0 ve sonucun 0 'dan farklı olmasından Z=0 olarak yansır.

Dikkat edilirse işaret nitelikli olmayan sayıların karşılaştırılmasında C ve Z değerlerine bakılırken işaret nitelikli sayıların değerlendirilmesinde S , V ve Z değerlerine bakılır.

Bazı bilgisayarlarda, çıkartma işleminde C bit ' i "borrow bit" olarak adlandırılır.Çıkartılan sayının büyük olmasıyla bir sonraki yüksek mertebeli bit'ten alınacak değere ihtiyaç duyulur.Bu nedenle C bit ' ine "borrow bit" denir.Tabii bu durum yalnızca çıkarılan sayının küçük olması durumunda geçerlidir.A - B işleminde A>B ise borrow bit sözkonusu olamaz.

O halde, bilgisayar açısından iki verinin karşılaştırılması yukarıda anlatıldığı gibi gerçekleştiriliyor . Ben de bu amaca hitaben hardware olarak tasarlanmış asenkron bir karşılaştırıcının çalışma ilkesinin software olarak simulasyonunu Yüksek Lisans Tez ' i olarak hazırladım. Programlama dili olarak C dilini seçmemdeki amaç ,bu dilin bit bazında işlem yapan işlemcilere sahip olmasıdır. XOR ve AND işlemcileri karşılaştırmalarda esas alınan işlemcilerdir.

Karşılaştırıcı,Değerli Hocam Prof. Dr. Abdüssamet M ARŞOĞLU tarafından sıralama işleminde kullanılmak amacı ile tasarlanmış bir asenkron karşılaştırıcıdır.Karşılaştırıcının çalışma ilkesi , eşitsizlik durumunda küçük olan sayıyı göstermesidir . Üç farklı devre üzerinde işlem yapılıyor . Birinci devre en yüksek mertebeli bit çiftinden başlayarak aynı seviyedeki bit ' leri karşılaştırıyor. İkinci devre bit çiftinde eşitlik olması halinde , takibeden bit çiftlerinin karşılaştırmasını ,eşitsizlik halinde ise karşılaştırmanın sona ermesini ve sonucun elde edilmesini sağlıyor. Üçüncü devre , küçük ve büyük harflerin ASCII kodlarının farklı olmasından dolayı , küçük harfleri büyük harflere çeviriyor.

Karşılaştırıcıda temel olarak iki adet paralel giriş / çıkışlı 8 bit ' lik register mevcuttur. İşlem yeni girilen veriler üzerinde , bu verileri temsil eden bit lerde herhangi bir değişim olmadan sonuç elde edilinceye dek sürüyor. Karşılaştırma aynı sıradaki bit çiftlerinde , yüksek mertebeliden başlamak suretiyle yapılıyor.

Bit çiftinin karşılaştırmasından sonra , eğer eşitlik elde edilmezse , bu durumda karşılaştırma kesilir ve kalan bit ' ler yeni verilerin yüklenmesine kadar üzerlerinde işlem yapılmaksızın register ' da saklı kalır . Eşitlik olması halinde karşılaştırma devam eder. Eğer tüm bit çiftlerinde eşitlik elde edilmişse bu durumda girilen iki verinin birbirine eşit olduğu sonucuna varılır . Sonuç , eşitlik devresine gönderilir ve devre işlem görür . Aksi takdirde , pozitif ve negatif devreler işlem görür ve eğer bu devrelerden birinde işlem yapılırsa , önce girilen veriler paralel

çıkıştan hafizaya (memory) gönderilir ve yeni veriler paralel girişten register ' lara yüklenir .

Bit çiftlerinin karşılaştırılması , aynı sıradaki flip_flop ' lardan elde edilen verilerin XOR kapısına girmesi ile sağlanır . Eşitsizlik halinde XOR kapısının çıkışı " 1 " dir.Bu durumda mevcut bit çifti en yüksek mertebeli çift olup net sonucun eldesi için karşılaştırmaya tabii tutulur . Küçük olanın tayini için flip_flop ların Q' çıkışları teker teker XOR kapısı çıkışları ile AND 'lenir. Küçük olan bit " 0 " ve Q' çıkışı da " 1 " ' dir .

Bit çiftlerinde eşitlik olması halinde her iki bit ya 0 , ya da 1 ' dir. Bu durumda XOR çıkışı " 0 " olur. XOR çıkışları "1" ' e dönüştürülür ve bir sonraki bit çiftinin XOR çıkışları ile AND ' lenir. Eğer , çiftler eşit ise XOR ve AND kapısı çıkışları "0" olur. Bit çifti eşit değilse , bu durumda XOR çıkışı "1" ' dir.

Buraya kadar gerçekleştirilen işlemlerin özetini yapacak olursak ;

1 - Karşılaştırmaya en yüksek mertebeli bit ' ten başlanır.
2 - XOR Kapısı ile karşılaştırma yapılır.
3 - XOR çıkışı "1" ise işlem durdurulur. XOR çıkışı Q' çıkışları ile AND ' lenir ve küçük olan sayı tayin edilir .
4 - XOR çıkışı "0" ise işleme devam edilir.
5 - 2.,3. ve 4. adımlar eşitsizlik durumuna kadar tekrarlanır.
6 - Eğer , tüm çiftler eşit ise , o takdirde verilen değerler eşittir.

Büyük harflerin ASCII kodu Hexadecimal (16 'lık) sayı sisteminde $(41)_{16}$ ' den , küçük harflerin $(61)_{16}$ ' den başlar. Her bir kod alçak ve yüksek olmak üzere iki kısma sahiptir . Büyük harflerin bit temsilindeki yüksek kısım "010X" , küçük harflerin "011X" dir . Sağdan 5 rakam her ikisinde de tamamen

aynıdır. 6. rakam büyük harflerin temsilinde "0" , küçük harflerin temsilinde "1" olur. Bu durumda küçük harfleri büyük harflere dönüştürürken sadece 6. rakamı "0" yapmak yeterlidir. Bu amaçla yedinci flip_flop ' taki girdiler altıncı flip_flop ' lara gönderilir ve dönüştürülür. Eğer , yedinci veri "1" ise , "0" değeri altıncı flip_flop ' a gönderilir . Yedinci verinin "0" olması durumunda ise altıncı flip_flop girdisinde herhangi bir değişiklik yapılmaz.

Veriler , karşılaştırıcı içinde binary kodlarıyla işleme alındığından , bu işlemlerin programlamaya simule edilmesinde binary sistemi kullandım. Veri tipi ne olursa olsun , girilen bilgiler ilgili register ' ları temsil eden katarlar içerisine yükleniyor . Örneğin , integer tipli veriler 8 bit ' lik register ' ları temsil eden 8 karakterlik katarlarda saklanıyor. Aynı şekilde karakter bilgileri de 8 karakterlik katarlarda işlem görüyor. Girilen iki verinin tipine bağlı olarak katarlara yüklenmesinden sonra karşılaştırma bit tabanlı XOR işlemcisinin kullanılmasıyla yapılıyor. XOR işleminde farklı operandlar için "1" elde edileceğinden , bit çiftlerinin karşılaştırılmasında "1" sonucu kontrol ediliyor. Bu sonuç elde edildiğinde büyük / küçük sayının tayini için "1 ile AND ' leme" yöntemi kullanılıyor . Şu durumda bir bit çifti mevcut ve "1" olan bit ' e sahip katar büyük sayıyı ,"0" olan bit ' e sahip katar da küçük sayıyı temsil ediyor. Eğer ,çiftin herhangi bir bit ' i 1ile AND ' lenir ve sonuçta yine "1" verirse, o takdirde bu bit ' in temsil ettiği sayı diğerinden büyük olur. İşte bu amaçla programda ilk katarın bit ' i AND ' leniyor ve sonuç kontrol ediliyor . Sonuç "1" ise birinci ikinciden , "0" ise ikinci birinciden büyük oluyor. Bu mantık verilerin tiplerine ve niteliklerine bağlı olarak programda kullanılıyor.

Bit çiftlerinin XOR ' lanmasında farklılık yakalanmazsa en düşük mertebeli bit ' e kadar işlem devam ettiriliyor ve yine fark olmazsa iki verinin de eşit olduğu sonucuna varılıyor.

Tasarımda işaret bit ' leri için ayrılan flip_flop ' lar program içinde katarların soldan ilk karakteri olarak temsil ediliyor. Bu karakterler ,verilerin işaret

incelemesinde "1 ile AND " ' leme işlemine tabii tutuluyorlar . Fakat burada büyük sayı pozitif , dolayısıyla işareti "0" olacağından AND ' leme sonucu "0" ile kontrol ediliyor . İlk katarın bit ' i işleme alınıyor ve sonuç "0" ' veriyorsa pozitif sayı elde ediliyor.

BÖLÜM 2. BİT İŞLEMCİLERİ

Diğer programlama dillerine nazaran C , bit bazında işlem yapan işlemcilere sahiptir. C , birçok programlama işinde Assembler dilinin yerini alacak şekilde tasarlandığından Assembler dilinde yapılan çoğu operasyonları içermesi önemli bir özelliğidir.

Bit işlemcileri , C 'nin int , char ve long veri tiplerine karşılık gelen 1 byte (8 bit) 'ın veya 1 word (2 byte) 'ün test edilmesi, düzenlenmesi veya değiştirilmesi fonksiyonlarını kapsar. Float , double,long double,void ve daha karmaşık tiplerden veriler üzerinde bit işlemleri yapılamaz. Bit işlemcileri ; sürücü ile ilgili problemlere , modem programlarına , disk,dosya ve yazıcı ile ilgili problemlere uygulanabilir. İşlemciler aşağıda gösterildiği gibidir.

işlemci	etkisi
&	AND
|	OR
^	XOR(exclusive or)
~	NOT (1' in tümleyeni)
>>	Sağa Kaydırma
<<	Sola Kaydırma

Bit işlemcileri AND (&),OR (|) ve NOT (~) ' un doğruluk tablosu, mantık işlemcileri AND (&&), OR (||) ve NOT (!) gibidir. XOR (^) bit işlemcisinin mantıksal işlemcilerde karşılığı bulunmamaktadır. Doğruluk tablosu aşağıdaki gibidir.

p	q	p&q
0	0	0
1	0	0
0	1	0
1	1	1

p	q	p|q
0	0	0
1	0	1
0	1	1
1	1	1

p	q	p^q
0	0	0
1	0	1
0	1	1
1	1	0

İfade	**Binary Değer**
A:	1001 0110 0001 0001
B:	0001 1010 1101 1001
A&B:	0001 0010 0001 0001
A|B :	1001 1110 1101 1001
A^B :	1000 1100 1100 1000

Tablodan da görüldüğü gibi XOR işlemi , işlenenlerden yalnız biri doğru olduğunda doğru , diğer durumlarda yanlış sonuç verir . Bit işlemcileri disk dosyalama ortamlarında , modemler ve yazıcılar gibi cihazlar için yazılan uygula ma programlarında oldukça sık kullanılır.

Örneğin eşlik bit'i gibi belirli bit'leri maskelemek ya da maskelenmiş bit'leri test etmek için bit işlemcileri kullanılabilir. (1 byte verideki diğer bit'lerin taşımada değişip değişmediğini görmek için eşlik bit'i kullanır. Eşlik bit'i genellikle bir byte 'taki en anlamlı bit'tir.)

En yaygın kullanıma bakıldığında, ***AND*** bit işlemcisinin bitleri sıfıra değerlediği düşünülebilir ; yani iki işlenenden herhengi biri 0 ise işlemin sonucu 0 olur. Örneğin , aşağıdaki fonksiyon ilk olarak RS_232_oku() fonksiyonunu kullanarak modem portundan bir karakter okur , daha sonra eşlik bit'i (en anlamlı

bit=7.bit) sıfıra değerlenir. Bu şekilde seri iletişim portundan alınan veri , 7 bit'lik gerçek değerine dönüştürülerek fonksiyona değer olarak atanır. Burada belirtmek gerekir ki , fonksiyona atanan değer char tipinde olduğundan gene 8 bit 'liktir, yalnızca en anlamlı bit sıfır değerini taşımaktadır.

```
char modem_oku()
{ char ch;
   ch = RS_232_oku(2,0,0);
   return(ch & 127);
}
```

Verilen örnek fonksiyonda , ch & 127 ifadesinde , ch değişkeninin ilk 7 bit'i (0. bit'ten 6. bit'e kadar) 1 değeriyle , en anlamlı bit olan 7. bit ise 0 değeri ile karşılaştırılmaktadır. İlk 7 bit'i 1 olan iki tabanlı (binary) sayının on taban lı (decimal) karşılığı 127 'dir. ch ' ye eşlik bit'i kullanılarak A karakteri değerinin atandığını varsayalım.

işlem	iki tabanlı değer	on tabanlı karşılığı
	1 1 0 0 0 0 0 1	193
&	0 1 1 1 1 1 1 1	127
	---	
	0 1 0 0 0 0 0 1	65 ('A')

Bit'leri 1'e değerlemek için ise AND işlemcisinin tersi olarak bakılabilecek ***OR*** bit işlemcisi kullanılabilir. OR ile yapılan karşılaştırmada iki işlenenden herhangi biri 1 ise diğeri de 1 'e değerlenir . Örneğin , 69 değerine (ASCII 'E') yukarıdaki örneğin tersi olarak eşlik bit'i atayalım . Bu durumda 7. bit'i 1'e değerlememiz gerekmektedir.

işlem	iki tabanlı değer	on tabanlı değer
	0 1 0 0 0 1 0 1	69 ('E')
|	1 0 0 0 0 0 0 0	128
	---	
	1 1 0 0 0 1 0 1	197 (eşlikli ' E ')

OR işlemcisinin özel bir türü olan ***XOR***, yalnızca karşılaştırılan iki işlenen birinden farklı olduğunda (yani bir bit 0 değerli iken,diğeri 1 değerli ise) sonuç bit'ini 1 ' e değerleyen bir işlemcisidir. Örneğin, 255 ile 240 değerine XOR işlemini uygularsak,

işlem	iki tabanlı değer	on tabanlı	onaltı tabanlı
	1 1 1 1 1 1 1 1	255	FF
^	1 1 1 1 0 0 0 0	240	F0
	---		
	0 0 0 0 1 1 1 1	15	0F

Bu örnekte işlenenlerin onaltı tabanlı sayı karşılığı özellikle verilmiş tir. Bit işlemle rinin yapılması gereken durumlarda iki tabanlı sayıların onaltı tabanlı sayı karşılığının hesaplanması (ya da tersi) on tabanlı sayılara göre çok daha kolay olduğundan genellikle tercih edilirler.

AND, OR ve XOR bit işlemcileri işlemlerini bit'ler üzerinde gerçekleştirirler. Bu nedenle mantıksal ya da ilişkisel işlemcilerin kullanıldığı biçimde, koşul ifadelerinde kullanılmazlar . Örneğin , x değişkeni 7 değerine sahipse , x & 8 yanlış , buna karşılık x && 8 doğru sonuç verir.

AND operatörü , istenilen bit'in 'on' ya da 'off' durumunu kontrol amacı ile de kullanılan yararlı bir operatördür . Örneğin 'durum' değişkeninin 4. bit'ini kontrol etmek için ;

```
if (durum & 8) printf(" 4.bit 'on' durumunda");
```

ifadesini yazmak gerekir . Burada 'durum' değişkeninin 8 ile And ' lenmesindeki amaç , binary kodlamasında 4. bit 'in 1 olmasından kaynaklanır.(00001000=8) Bu durumda 'if ' komutu, sonucu 1 olarak verirse istenilen bit ' in 'on ', aksi takdirde 'off ' durumunda olduğu anlaşılacaktır.

XOR ' un tipik bir kullanımı da bir değeri iki olası değer arasında değiştirmektir. Örneğin , siyah rengin 7 ve beyaz rengin de 0 olduğunu varsayalım. Video niteliklerini beyazdan siyaha , bazı durumlarda da siyahtan beyaza dönüştü ren bir program içerisinde , değiştirme işleminin ne şekilde olacağından emin olunmayacağından XOR bit işlemcisinin kullanılması ile bu işlem gerçekleştirilir.

```
renk ^= 7 ;          /* rengi ya 0 ya da 7 yap */
```

Eğer renk 7 (siyah) ise bu durumda 1 1 1 binary kodu ile 1 1 1 Xor 'lanacak. 1 1 1 ^ 1 1 1 = 0 olacağından renk 0 (beyaz) ' a dönüşecek. Aksi durumda , renk 0 (beyaz) ise 1 1 1 binary kodu 0 0 0 ile Xor ' lanacak.
1 1 1 ^ 0 0 0 = 1 1 1 olacağından renk 7 (siyah) ' ye dönüşecektir.Görüldüğü gibi Xor ' lama ile 0 7 ' ye, 7 de 0 ' a dönüştürüldü.

AND işlemcisi bir durum değişkeninin bit'lerinden herhangi birinin durumunu (1 ya da 0 olup olmadıklarını) test etmek için de kullanılabilir. Örneğin, bir cihazın durumunu gösteren durum adlı bir değişkenin 1. bit 'i cihazın hazır olduğunu göstersin :

```
if (durum & 2 ) printf (" Cihaz Kullanıma Hazır \n");
```

2 ile karşılaştırma yapmamızın nedeni 0000 0010 iki tabanlı sayısının on tabanlı karşılığının 2 olmasıdır. 2 sayısında yalnızca 1.bit 1, diğer bit'ler ise 0 değerli olduğu için , 'durum ' adlı değişkenin 1.bit'i dışındaki bitlerinin değeri ne olursa olsun , AND işlemiyle sonuç bit'leri 0 olacaktır. 2 ile 'durum' değişkeninin 1. bit'leri karşılaştırıldığında ise her ikisinin de 1 olması halinde sonuç bit'i 1, bunun dışında 0 olacaktır. Dolayısıyla, if deyimindeki koşul ifadesi,

```
durum & 2 : 2  (!0,doğru),  eğer 1. bit 1 ise
durum & 2 : 0  ( 0, yanlış),  eğer 1.bit  0 ise
```

şeklinde sonuç verir.

Aşağıdaki fonksiyon , verilen sayının iki tabanlı karşılığını ekrana basmaktadır. Burada AND bit işlemcisi kullanılarak i parametresiyle fonksiyona verilen sayının her bir bit'inin 1 olup olmadığı test edilmekte ve sonuca göre ekrana 1 ya da 0 yazılmaktadır.

```
void display_binarycode(int i)
{
  register int i;
  for (t=128;t>0;t=t/2)
  if (i & t ) printf ( " 1 ") else printf ( " 0 ");
}
```

Dos İşletim Sisteminde belleğin 1047 no ' lu adresindeki 16 bit ' in her biri , klavye üzerindeki kontrol tuşlarının durumunu yansıtır . Bu bitlerden herhangi birinin değeri '1' ise karşılık gelen kontrol tuşu basılı durumdadır. Bu

lokasyondaki bitlerden birini inceleyebilmek için , diğer bit ' lerin sıfırlanması gerekir. Bu amaçla bitsel AND operatöründen yararlanılır.

Aşağıda verilen program , 1047 no ' lu lokasyonun bitlerini düşük seviyeden yüksek seviyeye kadar tek tek taramakta ve hangi tuşların basılı durumda olduğunu ekrana yazdırmaktadır ;

```
#include " stdio.h "
#include " dos.h "
unsigned int far *i,j,k;
char tuslar[16][12] = {"Sağ Shift ",
                       "Sol Shift ",
                       "Ctrl ",
                       "Alt ",
                       "Scroll Lock ",
                       "Num Lock ",
                       "Caps Lock ",
                       "Insert ",
                       "Sağ Alt ",
                       "Sol Alt ",
                       "Sys Rq ",
                       "Pause ", }

main()
{ FP_SEG( i ) = 0;
  FP_OFF( i ) = 1047;
  k = 1;
  printf( " Basılı Durumdaki Tuşlar.....");
  for ( j = 0xffff; j ; --j);
  for ( j = 0 ; j < 16 ; j++);
```

```
        { if ( *i & k ) printf ( " , %s ",tuşlar [j]);
            k*=2;
        }
}
```

Unutulmaması gereken önemli bir nokta şudur :İlişkisel ve mantıksal işlemciler her zaman doğru ya da yanlış tek bir sonuç verirler. Buna karşılık bit işlemcileri duruma göre 1 ya da 0 değerlerinden farklı sayısal değerler üretebilir.

Değişken ismi	Veri
a	24
b	12

```
main()

{ int a=24,b=12;
  int c,d,e;
  c=a & b;
  d=a ^ b;
  e=a | b;
  printf("%d %d %d \n",c,d,e);
}
```

Programın icrası sonucunda ;

a & b = 8

a ^ b = 20

a | b = 28 bulunur.

11000=24 ve 1100=12 binary kodlarıyla;

AND a & b = (01000) = 8

OR a | b = (10100)= 20

XOR a ^ b = (11100)= 28 ' dir.

KAYDIRMA OPERATÖRLERİ , << ve >> , bütün bitleri sağa ya da sola tanımlandığı biçimde taşıma amacı ile kullanılır.Sağa kaydırma operatörünün en genel kullanım şekli ;

değer << kaydırılan pozisyon sayısı

sola kaydırma ifadesi;

değer << kaydırılan pozisyon sayısı

Kaydırma işlemi bir rotaston olmadığından , başlangıç ve bitim noktalarından kaydırılan bitlerin diğer uca dönmesi mümkün değildir. Bu durumda söz konusu bitler ' 0 ' değerini alır.

Bit tabanlı kaydırma operatörleri D/A (Display/ Analog) dönüştürücüleri ve durum bilgilerinin okunması gibi harici giriş ünitelerinin kodlanması işlemlerinde oldukça faydalı operatörlerdir . Bu operatörler aynı zamanda tamsayıların çarpma ve bölme işlemlerinde de seri bir işlem olanağı sunar. Sağa kaydırma operatörü sayının 2 ile bölünmesi , sola kaydırma da 2 ile çarpılması demektir. Eğer kaydırılan pozisyon adedi 'k' ise bu durumda 2^k sözkonusudur.

karakter x	işlemler	x ' in değeri
x = 7	0 0 0 0 0 1 1 1	7
x << 1	0 0 0 0 1 1 1 0	14
x << 3	0 1 1 1 0 0 0 0	112
x << 2	1 1 0 0 0 0 0 0	192
x >> 1	0 1 1 0 0 0 0 0	96
x >> 2	0 0 0 1 1 0 0 0	24

Örnekte görüldüğü gibi herbir sola kaydırma işlemi sayının 2 ile çarpılmasına karşılık geliyor. Dikkat edilirse herhangi bir rotasyon işlemi yok. Sınırlardan taşan bit ' lerin fonksiyonu kalmıyor . Açılan bit alanı da ' 0 ' ile doldurulyor.

Kaydırma operatörünün etkisini ekran üzerinde görmek için aşağıdaki basit programın yazılması yeterlidir :

```
void binary_code()
{ register int i;
  for (i=128;i>1;i/2)
  if (j & i) printf("1 "); else printf("0 ");
  printf("\n");
}
main()
{int j,i;
 j=1;
 for (i=0;i<8;i++)
 {binary_code();j=j<<1;}
```

```
printf("\n");
for (i=0;i<8;i++)
{j=j>>1;binary_code();}
}
```

Programın icrası ile aşağıdaki çıktı alınır :

```
0 0 0 0 0 0 0 1
0 0 0 0 0 0 1 0
0 0 0 0 0 1 0 0
0 0 0 0 1 0 0 0
0 0 0 1 0 0 0 0
0 0 1 0 0 0 0 0
0 1 0 0 0 0 0 0
1 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 1 0 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 1
```

C , her nekadar bir rotasyon operatörüne sahip değilse de , bu işlemi kaydırma operatörleri ile yapmak oldukça basittir . Sola kaydırma işlemi , bit ' in sol uçtan sonra fonksiyonunu yitirmeyip sağ uçtan itibaren sürdürmesi haricinde

düşünülürse bir rotasyona benzer . Örneğin , 1 0 1 0 bilgisinin 1 defa rotasyonu sözkonusu ise 0 1 0 1 olması gerekir.

Kaydırma operatörlerinin pozitif tamsayılara uygulanmasında trailing bit ' ler daima 0 olur . Fakat tamsayı negatif ise bu durumda sağa kaydırma işlemi sonucu tahmin edilemez. Bazı makinelerde trailing bit ' ler daima 0 , diğerleri de 1 olur. Buna göre denilebilir ki sağa kaydırma işleminin negatif sayılara uygulanmasında çıkacak sonuç tamamiyle makineye aittir.

Kaydırma operatörleri ile bazı işlemleri basitleştirmek mümkündür. Örneğin , donanım bilgilerinin saklı olduğu bir flag 'i kullanarak mevcut olan sürücü adedini öğrenmek , aşağıdaki program parçası ile gerçekleştirilebilir :

```
short int  flag,mask,değer,adet;
mask=0x18;                    /*  0 0 0 1 1 0 0 0 = 24 */
değer = flag & mask ;
if ( değer ==0) adet=0;
else if (değer==8) adet==1;
     else (if değer == 16) adet=2; else n=3;
```

Araştırılan değer nnnNNnnn formatında olduğundan decimal olarak alacağı değerler 0 , 8 , 16 veya 24 olacaktır. Halbuki , sağa kaydırma operatörü ile sözkonusu 4. ve 5. bit ' ler sağ uca taşındıktan sonra direkt olarak sürücü adedi elde edilebilir. Şöyle ki :

```
unsigned short int flag,mask,değer,adet;
mask = 0x18;
değer = flag & mask ;
adet = değer >> 3 ;
```

Son iki satır için adet = (flag & mask) >> 3 ifadesi de kullanılabilir.

IBM PC karakter operasyonları , bit işlemcilerinin nasıl kullanılabileceklerini gösterir. Örneğin 16 bit ' lik bir tamsayının 16 bit ' lik bir registera kopyalanacağını düşünelim. Bilindiği gibi 1word 16 bit 'ten(2 byte) oluşur. Soldan 8 adet bit ASCII karakterlerini temsil eder. Sağdan 8 bit ise 2 tane 4 bit ' lik flagleri içerir. İlk 4 bit 0 ile 15 arasında olmak üzere yazı renklerini , sonraki 4 bit ise zemin renk bilgilerini içerir . Word ' ün ilk byte ' na ' A ' (ASCII 65) , zemin flag ' ine 7 ve yüzey flag ' ine de 10 niteliklerini yerleştirelim . Bu durumda 16 bit ' lik word ' ün son durumu şu şekilde olur ;

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

Niteliklerin binary kodlaması ; (0 1 0 0 0 0 0 1) = 65 , (0 1 1 1) = 7, (1 0 1 0) = 10 'dur . Karakterin yüklenebilmesi için ' A ' 'nın öncelikle ' 0 ' bir değerle OR ' lanması , sonra da 8 bit sola kaydırılması gerekir.

```
ch = 0x41          /* 0 1 0 0 0 0 0 1 = 65 ('A')*/
değer = 0;
değer = ( değer | ch ) << 8;
```

İşlem sonunda word ' ün ilk byte ' ı istenen karakter kodunu içerecektir.

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

İkinci byte ' ta, flaglerin nitelenmesi için aşağıdaki operasyonlar gerçekleştirilir :

```
zemin = 7 ;
nitelik = nitelik | zemin ;          /*  nitelik = 0 0 0 0 0 1 1 1 */
nitelik << = 4 ;                     /* nitelik = 0 1 1 1 0 0 0 0 */
```

```
yüzey = 10 ;
nitelik = nitelik | yüzey ;
```

Son olarak mevcut iki karakterin OR ' lanması ile istenilen bilgiler register ' a yüklenmiş olur:

```
değer   = 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0      /* ilk byte ' taki ' A '
nitelik = 0 0 0 0 0 0 0 0 0 1 1 1 1 0 1 0      /* diğer byte ' taki
                                                  flag ' ler */
-----------------------------------------
 OR     0 1 0 0 0 0 0 1 0 1 1 1 1 0 1 0        /* istenen register */
```

KOMPLEMENT OPERATÖRÜ ise bir bit ' in durumunu değiştirir. Tüm 1 olan bit ' ler 0 ' a, 0 olan bit ' ler de 1 ' e dönüştürülür. Komplement operatörünün ilginç bir kullanımı bilgisayar da saklı olan karakter setini göstermektir . Aşağıdaki program , verilen bir karakterdeki bit ' lerin komplementini almaktadır. Değiştirilen bu bit ' ler karakter setine karşılık gelir. Örneğin ,' d ' karak terinin verilmesi ile ekrana cent işaretinin yazıldığı görülür. Bu şekilde bilgisayarda yüklü olan karakterler yazdırılabilir.

```
#include "stdlib.h"
#include "conio.h"
main()
{char ch;
            do {
 ch=getch();
 printf("%c",~ch);
                }
           while (ch!='q');
}
```

Bit işlemcileri şifreleme rutinlerinde sıkça kullanılır. Örneğin , bir disk dosyasının okunmaması için birtakım bit işlemlerinin yapılması gerekir. En basit yöntemlerden bir tanesi herbir bit ' in komplementini almaktır.

Orjinal Byte	0 0 1 0 1 1 0 0
İlk Değilleme	1 1 0 1 0 0 1 1
Son Değilleme	0 0 1 0 1 1 0 0

İkinci kodlamada orjinal byte elde edildiğinden ilk kodlama mevcut byte ' ın şifrelenmiş durumunu gösterir. Bir işlem ile yine eski durum elde edilir.

```
void  kodla(char ch)
{
 return(~ch);
 }
```

Bir karakteri kodlamak için kodla() fonksiyonunun icrası yeterlidir. Fonksiyon bir daha icra edilirse karakterin orjinal hali elde edilir.

Komplement operatörünün işlevi, assembly dilinde 1. tamamlayıcı olarak adlandırılır. Bir sayının 1. tamamlayıcısının 1 bit ' i veya 1 değeri ile toplanması ile elde edilen değer 2. tamamlayıcı olarak isimlendirilir ve sayının negatifine karşılık gelir . Derleyici programlar , negatif sayıların belleğe yerleştirilmesinde bu yöntemi kullanırlar ;

```
#include "stdio.h"
main()
{ int i = 5;
  i = ~ i ; i ++;
  printf( "i = %d \n ",i );
}
```

Programının icrası ile i tamsayısının son durumda negatif değerini aldığı görülür. (i = -5).

BÖLÜM 3. VERİ TİPLERİ

Veri tiplerinin çeşitli olmasına karşın , bilgisayarın çalışma ilkesi ikili (binary) sisteme dayandığından , depolanan ve işleme koyulan bilgilerin esas itibariyle 1 ve 0 ' lardan oluşması gerekir. Veri tipleri iki ana gruba ayrılır : Birinci grup ***" Sayısal Tipler "*** olup 3 alt gruba sahiptir. Bunlar , sadece sayıların yüklenmesi için kullanılan tiplerdir. İkinci grup ise hem sayısal hem de karaktersel bilgilerin yüklenmesi amacı ile kullanılan ***" Alfanümerik Kodlar "*** dır.Aşağıda görüldüğü gibi iki alt grup içerir .

1- Sayı Tipleri
   a) Tamsayı (İnteger veya genel olarak sabit_nokta(fixed_point))
   b) Kayan Nokta (Floating_Point veya Real)
   c) Ondalık (Decimal veya BCD)

2- Karakter Kodları
   a) ASCII
   b) EBCDIC

3.1. TAMSAYI TİPİ (*Integer Format*)

Sayılar , ispatsız olarak kabul edilen soyut matematiksel kavramlardır ve yazılış tarzları için birçok sistem geliştirilmiştir. Negatif sayıların dışında diğer tamsayıların temsili , onluk sistamde şu genel formda yaplır :

$$a_n.10^n + a_{n-1}.10^{n-1} + ... + a_0.10^0 = a_n.10^n + ... + a_1.10^1 + a_0$$

$a_n, a_{n-1}, ..., a_0$ katsayıları 0 ile 9 arasındaki tamsayılardır. Normal olarak kısaltılmış formda yazılırken , 10 ' un katları ve toplama (+) işareti iptal edilerek ;

$a_n a_{n-1} \ldots\ldots a_1 a_0$

halinde yazılır. $a_n, \ldots\ldots, a_0$ tamsayılarına ***rakam*** adı verilir ve 0,1,.....,9 olmak üzere 10 adettir. Örneğin ;

$$6 \times 10^3 + 2 \times 10^2 + 3 \times 10 + 5$$

6275 tamsayına karşılık gelir .

Asıl amaç , bu sayıları bilgisayarın anlayacağı şekildeki bir forma dönüştürmektir . Yani , sadece 1 ve 0 rakamlarını kullanarak aynı sayıyı ifade etmek . 10 tabanında işlem yaparken on adet sembol kullanıldığına göre , o halde iki sembol ile kısıtlanan bir sistemde 2 ' nin kuvvetleri sözkonusu olacaktır.Bu durumda , 1 ve daha büyük tamsayıların kuvvetlerine göre sayıların ifadesi en genel olarak ;

$$a_n.x^n + \ldots\ldots + a_1.x + a_0$$

şeklindedir . Burada , x pozitif bir tamsayıdır ve ***Taban*** (base , radix) adını alır. Eğer , taban 10 ise sistem "*Onluk Sayı Sistemi* "(Decimal Number System), 2 ise " *İkilik Sayı Sistemi*"(BinaryNumber System) adını alır. İkilik sayı sisteminin sembol leri 1 ve 0 ' dır ve herbirine ***bit*** denir.Örneğin , 10 tabanında $(11)_{10}$ tamsayısı ikilik sistemde aşağıdaki gibi gösterilir :

$$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 = (1\,0\,1\,1)_2$$

Bu iki sayı sistemlerinin haricinde , iki tane daha sistem vardır ki bunlar 8 tabanında işlem gören *Sekizlik Sayı Sistemi* (Octal Number System) ve 16 tabanında işlem gören *Onaltılık Sayı Sistemi* (Hexadecimal Number System). Tablo-1 ' de sayı sistemleri ve sembolleri parentez içinde bit karşılıkları ile verilmiştir.

Binary (Base 2)	Octal (Base 8)	Decimal (Base 10)	Hexadecimal (Base 16)
0	0(000)	0(0000)	0(0000)
1	1(001)	1(0001)	1(0001)
	2(010)	2(0010)	2(0010)
	3(011)	3(0011)	3(0011)
	4(100)	4(0100)	4(0100)
	5(101)	5(0101)	5(0101)
	6(110)	6(0110)	6(0110)
	7(111)	7(0111)	7(0111)
		8(1000)	8(1000)
		9(1001)	9(1001)
			A(1010)
			B(1011)
			C(1100)
			D(1101)
			E(1110)
			F(1111)

Tablo -1 Sayı Sistemleri

Hernekadar bilgisayarın Hexadecimal ve Octal Sistemlerde sayılarla doğrudan işlem yapması mümkün olmasa da , bu sistemlerin binary (ikilik) sistemlerle basit bir ilişkisi mevcuttur. Bu durumda notasyonun azaltılması sözkonusudur. Örneğin , yukarıdaki örnekte $(11)_{10}$ sayısını binary sistemde temsil etmek için 4 adet rakamın düzenlenmesine ihtiyaç duyuldu.
$(1\ 0\ 1\ 1)_2$. Halbuki , aynı sayı Octal sistemde 2 rakam $(13)_8$, Hexadecimal sistemde sadece 1 rakam (B sembolü) kullanılarak temsil edilir.

Kullanılan rakam sayısının az olmasına duyulan ihtiyaç, sayıların daha büyük olması durumunda kendini gösterir.

Farklı sayı sistemlerinde aynı sembollerin (0 ve 1) kullanılmasından, verilen formatın temsil ettiği gerçek sayıyı bulmak için , işlemin hangi tabanda yapılacağını belirtmek gerekir. Örneğin, 1101 şeklindeki bir ifade 13 sayısı olabileceği gibi aynı zamanda binyüzbir sayısı da olabilir. İşte bu çelişkiyi kaldırmak amacı ile ifade parentez içinde belirtilir ve işlem yapılacak sistem parantezin altına yazılır.

$$(1\ 1\ 0\ 1)_2 = (13)_{10}$$

Genel olarak onluk sistemde işlem yapılmasından ve bilgisayarın da ikilik sistem de işlem yapmasından dolayı bir sistemden diğer sisteme dönüşüm sözkonusudur. İkilik sistem den onluk sisteme dönüşüm ;

$$a_n.2^n + a_{n-1}.2^{n-1} + ... + a_1.2 + a_0$$

şeklinde olur . $a_n, a_{n-1}, ..., a_0$ katsayıları 0 ve 1 ' den oluşan tamsayılardır. Yukarıda verilen form itibariyle $(1\ 1\ 0\ 1\ 0\ 1)_2$ binary sayısının onluk sistemdeki karşılığı ;

$$1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2 + 1 = (53)_{10}$$

olur. Bu metodun dışında daha sistematik bir yöntem ***Horner Kuralı*** ' dır.

$$a_n.x^n + a_{n-1}.x^{n-1} + ... + a_1.x + a_0 = ((..(a_n.x + a_{n-1})x..)x + a_1).x + a_0$$

Örneğin ;

$$1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2 + 1$$

$=((((\ 1 \times 2 + 1).2+0).2+1).2+0).2+1 = (53)_{10}$

Horner Kuralı , aynı zamanda onluk sistemde verilen bir sayının ikilik sistemde karşılığını bulmak için de kullanılır. Şöyle ki ;

$$\frac{((...(a_n x+a_{n-1})x...)+a_1)x+a_0}{x} = (...(a_n x + a_{n-1})x...)x + a_1 + \frac{a_0}{x}$$

$(...(a_n.x+a_{n-1}).x...).x+a_1$ sonucu bölümü , a_0 ise kalanı verir. Bölüm üzerindeki işlem tekrarlanırsa ,

$$\frac{(...(a_n.x+a_{n-1}.x).x...).x+a_1}{x} = (...(a_n.x + a_{n-1})x...)x + a_2 + \frac{a_1}{x}$$

eşitliği elde edilir . Bu defa kalan a_2 ' dir . Bu şekilde tekrarlanan bölümlerle $a_0.....a_n$ katsayıları bulunur. İşlem aşağıda gösterildiği gibi gerçekleşir :

	kalanlar
2) 53	$1 = a_0$
2) 26	$0 = a_1$
2) 13	$1 = a_2$
2) 6	$0 = a_3$
2) 3	$1 = a_4$
2) 1	$1 = a_5$
0	

$(53)_{10} = (1\ 1\ 0\ 1\ 0\ 1)_2$ kalanlar sondan başlayarak yazılır.

3.1.2. FIXED_POINT (*Sabit Nokta*) TEMSİLİ

Sayılar , aritmetik işlemlerde kullanılmak üzere işaret , içerik ve ondalık nokta özellikleri gözönüne alınmak suretiyle düzenlenir. İşaret , sayının pozitif veya negatif olma durumunu açıkladığından , aritmetik işlemlerde ihtiyaç

duyulan bir özelliktir. Ondalık veya binary nokta pozisyonu ise sayının tam ve kesirli kısmını belirtmek amacı ile saklanır.

Bir sayının işareti , pozitif (+) ve negatif (-) bilgilerinden oluşan bir gruptur. İkilik sayı sisteminde sadece bir adet "bit" ile temsil edilir. Tüm pozitif sayılar için sayının işareti '0', negatif sayılar için ise '1' ile gösterilir. Bir register içerisindeki işaretli sayıların binary temsili yapılırken n bit ' in sayı için kullanılacağı düşünülürse , bu durumda (n+1) adet bit ' e ihtiyaç duyulur. İşaret bit ' i, register' in soldan birinci bit ' idir.

Register ' daki ondalık nokta temsili , register içinde iki flip_flop arasındaki pozisyon ile karakterize edildiğinden biraz karmaşıktır . Bu noktanın tanımlanmasında iki temel yol mevcuttur : sabit bir pozisyon *(fixed_position)* belirlemek veya noktayı hareket ettirmek *(floating_point)*. Sabit noktada, ondalık pozisyon daima bir pozisyondadır ; ya register ' in en solunda yer alıp sayı parçalı olarak alınır , ya da register ' in en sağına yerleştirilip sayı bir bir integer olarak alınır. Her iki durumda da ondalık pozisyon fiziksel olarak mevcut değildir. Sayının register ' da saklanmasına göre varlığından sözedilebilir . Daha ilerki konularda sözedilecek kayan nokta (floating_point) temsilinde iki register kullanılır.Bir register sayının saklanmasını sağlarken , diğer register da ondalık noktanın pozisyonunu içerir. (Bkz. Kayan Nokta Temsili)

Yukarıda , pozitif sayılar için işaret bit ' inin '0' , negatif sayılar için de '1' olduğundan bahsedildi. Pozitif sayılarda sayının içeriği onluk sistemde sürekli 2 ' ye bölme ile kalanların organizasyonundan oluşuyor. O halde , bilgisayar negatif sayıları ne şekilde temsil eder?. Bunun için bir sayının değil'inden *(complement)* sözetmek gerekir . Herbir r tabanlı sistem için iki tip komplement tanımlamak mümkündür.

(1) (r-1) ' in Komplementi
(2) r ' nin Komplementi

İkilik sistem için 1 ' in Komplementi ve 2 ' nin Komplementi gibi.

3.1.3 (r - 1) ' in Komplementi

r tabanında herhangi bir sayının (r-1) komplementi , sayıyı oluşturan herbir rakamın (r-1) ' den çıkartılmasından ibarettir. Örneğin , on tabanındaki sayılar için r - 1 = 10-1=9 ' un , iki tabanındakiler için r-1=2-1=1 ' in komplementi sözkonusudur. Buna göre , 835 sayısı nın 9 ' a göre komplementi ;

9 - 8 = 1
9 - 3 = 6
9 - 5 = 4 -------> 164 sayısını verir.

İkilik sistemde 1 0 1 0 sayısının 1 ' göre komplementi ;

1 - 1 = 0
1 - 0 = 1
1 - 1 = 0
1 - 0 = 1 -------> 0 1 0 1 .

Bu sistemde her rakam 1 ' den çıkartıldığından ya 1 - 0 = 1, ya da 1 - 1 = 0 işlemleri sözkonusudur . Heriki durumda da bulunan sonuçlar çıkarılan rakamın değili 'ne karşılık gelir . Buradan denilebilir ki , bir binary sayının 1 ' e komplementi o sayının bit bit değillenmesinden başka birşey değildir. Buna göre en basit yöntem , verilen binary sayıdaki 0 ları 1, 1 leri de 0 yapmaktır.

Octal ve Hexadecimal sayıların (r-1) komplementinde herbir rakam 7 ve F (15) den çıkarılır. Bu sayılar binary kodda ise o takdirde 1 'ler 0 'a , 0 'lar da 1 'e çevrilmek suretiyle komplement işlemi tamamlanır.

r ' nin Komplementi

Tanım olarak bir sayının r ' ye göre komplementi, (r -1) komplementine 1 eklemekten ibarettir. Örneğin , on tabanında 835 sayısının 10 ' a komplementi ;

1) r - 1 = 10 - 1 = 9 ' a komplementi

9 - 8 = 1
9 - 3 = 6
9 - 5 = 4 ------> 164

2) 164 + 1 = 165 sayısıdır.

3.1.4. 2 ' ye Komplement İle Negatif Sayı Temsili

Bilgisayarın çalışma ilkesi ikilik (binary) sisteme dayandığından, depolanan ve işleme konulan bilgilerin esas itibariyle 1 ve 0 ' lardan oluşması gerektiği " Veri Tipleri " başlığı altında vurgulanmıştı. Bu nedenle bilgisayar, negatif sayıların içeriği ve işareti olmak üzere değerlendirir. Eğer , sayı n bit ' lik bir alana yerleştirilecekse soldan birinci bit ' in işaret bit 'i olmasından dolayı kalan (n-1) bit sayıyı temsil eder. İşaret bit ' inin pozitif sayılar için '0', negatif sayılar için '1' olarak kullanılmasıyla oluşan forma ***İşaret_nitelikli Form (sign magni tude format)*** denir. Buna göre 8 bit ' lik bir alana (1 byte) $-(2^7)=-128$ ile $(2^7-1)=127$ arasında ki işaretli tamsayılar ifade edilebilir . O halde , negatif sayı temsili yaparken, ikilik sayı sistemin de 2 ' nin komplementi ile sayıya ulaşmak en genel metodtur.

Örneğin ; on tabanında verilen 8 sayısının negatifi , bilgisayar tarafından aşağıda gösterildiği gibi temsil edilir :

1) $\sim(00001000)_2 = (8)_{10}$

2) $(11110111)_2$

3) $+ \qquad 1$

$(11111000)_2 = (-8)_{10}$

O halde konuyu özetlemek gerekirse ; pozitif bir tamsayı , iki tabanında işaretli olarak temsil edildiğinde işaret bit ' i '0' olur ve geriye kalan bitler bu tamsayıyı gösterir. Sayı negatif olduğunda işaret bit ' i '1' olur , fakat sayının geri kalan bölümü 3 ayrı metodla temsil edilir :

1) İşaret Nitelikli *(Sign_Magnitude)*
2) İşaretli_1 ' in Komplementi *(Signed_1 's Complement)*
3) İşaretli_2 ' nin Komplementi *(Signed_2 's Complement)*

İşaret nitelikli temsilde , sayının içeriği negatif işaretin hemen akabinde bulunur. Örneğin , 7 bitlik bir register içinde +9 sayısı 0 0 0 1 0 0 1 olarak yüklenirken, -9 sayısının yüklenmesinde üç ayrı metod aşağıdaki gibi gerçekleşir :

1. İşaret Nitelikli	1	0 0 1 0 0 1
2. İşaretli_1' in Komplementi	1	1 1 0 1 1 0
3. İşaretli_2 'nin Komplementi	1	1 1 0 1 1 1

3.2. KAYAN NOKTA *(Floating_Point)* TEMSİLİ

Floating_point sayıların temsilinde iki kısma ihtiyaç duyulur. İlk kısım işaretli tamsayıyı belirtir ki buna ***mantis*** *(mantissa)* denir. İkinci kısım ise ondalık noktanın yerini tayin eder ki buna da ***üs*** *(exponent)* adı verilir. Örneğin , on tabanında + 6132.789 sayısı kayan nokta gözönüne alınırsa aşağıdaki gibi ifade edilir :

	işaret		işaret	
mantis:	**0**	.6132789	üs:	**0** 04

Mantis ' in soldan birinci pozisyonu sayının pozitif olduğunu göstermek amacı ile ' 0 ' dır . Ondalık mantis register' larda saklanırken 29 tane flip_flop ' a ihtiyaç duyulur : Herbir BCD (Binary Coded Decimal) dijit için 4 flip_flop ve 1 tane de işaret için. Register içinde desimal nokta fiziksel olarak mevcut değildir, sadece olduğu farzedilir. Bu noktanın gerçek pozisyonunu üs için ayrılan register tayin eder. Üs, +04 desimal sayısını binary kodda sakladığı zaman anlaşılmalıdır ki olduğu farzedilen ondalık nokta sağa doğru 4 adet kaydırılacak. Buna göre + 6132.789 sayısının mantis ve üs parçalarına ayrılması + $.6132789 \times 10^{+4}$ şeklinde olur.

Bazı bilgisayarlar , mantis bilgisini sabit noktalı ondalık değer ($.1235 \times 10^{+2}$ gibi) olarak değil de sabit_noktalı tamsayı değer olarak değerlendirir . ($62.73 = 6273. \times 10^{-2}$) . Hangi tabanda işlem yapılacağı ise register ' da tanımlanmıştır. Örneğin , mantis için tamsayı temsilini ve sistem olarak da 8 (octal) ' lik sistemi kullanan bir bilgisayar için ; + 36.754 octal sayısı kayan nokta temsili sözkonusu olduğunda aşağıdaki gibi değerlendirilecektir :

$$+ 36.754 = 36754 \times 8^{-3}$$

işaret işaret

mantis : **0** 36754 üs : **1** 03

Aynı sayı , binary kodlaması (BCD) yapılarak register ' a yazıldığında register ' ın gerçek değeri aşağıdaki gibidir :

0 011 110 111 101 100 **1** 000 011

Bir floating _point sayı $\mathbf{m \times r^e}$ genel formu ile temsil edilir. Mantis 'm' ve üs 'e' fiziksel olarak register içinde işaretleriyle birlikte yeralır. Sayı sistemi ve ondalık nokta pozisyonu farzedilen değerlerdir.

Bir floating_point sayı eğer soldan itibaren ondalık noktaya kadar 0 (sıfır) içermiyorsa , o takdirde bu sayı için ***normalize*** edilmiştir denir. Örneğin , mantis değeri 035 normalize olmamış bir değerdir. 350 sayısı normalizedir , çünkü ilk dijiti '3' tür ve '0' dan farklıdır.

3.3. DESİMAL SABİT NOKTA *(BCD)* TEMSİLİ

Desimal sayıların register ' lardaki BCD temsili , herbir rakamın ikilik sistem - de belirtilmesinden ibarettir. Desimal sayıların binary(ikili) kodda ele alınmasından dolayı fonksiyon ***Binary Coded Decimal*** adını alır. Herbir rakam 4 bit ' lik bir alanda kodlanır ve bu alanlar da dört adet flip_flop ' a ihtiyaç duyar . + 4385 sayısının BCD kodlaması düşünüldüğünde ; 1 flip_flop işaret için , her rakam için 4 flip_flop ' dan 4 x 4 = 16 flip_flop rakamlara olmak üzere 17 tane flip_flop ' a gerek duyulur. Aynı sayının 25 flip_flop ' luk bir register ' daki temsili aşağıdaki gibidir :

+	0	0	4	3	8	5
0	0000	0000	0100	0011	1000	0101

Negatif bir tamsayının BCD temsilinde üç farklı metod mevcuttur. Sayı sisteminin farklı olması dışında negatif bir tamsayının binary kodda temsil edilmesiyle aynıdır.

1. İşaret Nitelikli Temsil *(Signed_Magnitude)*
2. İşaretli_9 ' un Komplementi *(Signed_9 ' s Complement)*
3. İşaretli_10 ' nun Komplementi *(Signed_10 ' s Complement)*

Herbir temsil için öncelikle gerekli olan sayının pozitif olarak onluk sistemde ele alınması ve pozitif olduğundan dolayı da işaret bit ' inin '0' yapılmasıması . İşaret_nitelikli temsilde sadece işaret bit ' i '1' olur. Sayının içeriği pozitif halidir. Diğer iki temsilde ise , sayı 9 'un ve 10 ' un komplementi olarak değerlendirilir. İşaret , bazen 4 bit' lik kodlamaya uyulması amacı ile 4 bit üzerinden temsil edilir. Örneğin, pozitif sayılar için 1 1 0 0 olarak kodlanırken , negatif sayılar için 1 1 0 1 olarak kodlanır.

25 flip_flop ' luk bir register 'da - 4385 tamsayısının temsili işaret_nitelikli (signed_magnitude) olarak aşağıdaki gibidir :

-	0	0	4	3	8	5
1	0000	0000	0100	0011	1000	0101

Görüldüğü gibi sadece işaret bit ' i '1' oldu ve register ' ın kalan kısmı pozitif durumdaki ile aynı kaldı.

Desimal Sayı	BCD Sayı
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	0001 0000
20	0010 0000
50	0101 0000
248	0010 0100 1000

Tablo - 2 BCD Sayılar

2) KARAKTER KODLARI

3.4. ALFASAYISAL *(Alphanumeric)* TEMSİL

Dijital bilgisayarlardaki birçok uygulama sadece sayı tiplerini değil, aynı zaman da alfabetik harfleri ve bazı özel sembolleri de kapsayan veri organizasyonunu içerir. Tüm bu bilgilerden oluşan gruba **Alfanümerik Karakter Grubu** *(Alphanumeric Character Set)* denir ve 10 tane rakamı, alfabenin 26 harfini ve bazı özel sembolleri (**$,=,& gibi**) içerir.Grupta büyük harfler 32 ile 64 arasında, hem büyük hem de küçük harfler 64 ile 128 arasındadır. İlk durum için

binary kodlaması bit ' lik bir register gerektirirken , ikinci durumda 7 bit ' lik bir register ' a ihtiyaç duyulur . Standard alfasayısal binary kodlaması 128 karakteri kodlamak amacı ile 7 bit ' lik register kullanır. Bu kod sistemi **ASCII** *(American National Standard Code)* kısaltması ile temsil edilir.

Aşağıdaki tabloda A ' dan Z ' ye kadar büyük harflerin , 0 - 9 arası rakamların ve birkaç özel sembolün binary kodlaması verilmiştir . Rakamların binary kodlamasında BCD dönüşümü yapılıp , register ' ın ilk üç bit ' i '0 1 1' olarak tanımlanır.

Karakter	Binary Kod	Karakter	Binary Kod
A	100 0001	0	011 0000
B	100 0010	1	011 0001
C	100 0011	2	011 0010
D	100 0100	3	011 0011
E	100 0101	4	011 0100
F	100 0110	5	011 0101
G	100 0111	6	011 0110
H	100 1000	7	011 0111
I	100 1001	8	011 1000
J	100 1010	9	011 1001
K	100 1011		
L	100 1100		
M	100 1101	boşluk	010 0000
N	100 1110	.	010 1110
O	100 1111	(	010 1000
P	101 0000	+	010 1011
Q	101 0011	$	010 0100
R	101 0010	*	010 1010

Karakter	Binary Kod	Karakter	Binary Kod
S	101 0011	)	010 1001
T	101 0100	-	010 1101
U	101 0101	/	010 1111
V	101 0110	,	010 1100
X	101 1000	=	010 1100
Y	101 1001		
Z	101 1010		

Tablo 3 - ASCII Karakter Seti

Binary kodlama dijital bilgisayar operasyonlarında hiç kuşusuz önemli bir rol oynar. Register ' ların sadece ikilik sistemdeki bilgileri kullanabilme özelliğinden , kodlama iki tabanında olur. Register ' lardaki bilgiler incelendiğinde , aslında bu bilgilerin binary sayılardan ziyade birtakım veri tiplerini temsil ettiği görülebilir.

ASCII karakter kodu haricinde diğer bir kodlama da **EBCDIC** ***(Extended BCD Interchange Code)*** ' dir. Herbir karakter için 8 bit kullanılır ve 9. bit parite bit ' dir . EBCDIC , ASCII karakter setindeki tüm sembolleri içerir. Aradaki tek fark karakterlerin temsilindeki bit organizasyonudur.

BÖLÜM 4. VERİLERİN PROGRAMDAKİ TEMSİLLERİ

Karşılaştırılması istenen tipler ana menü içerisinde sunulur. Bunlar İntegers (pozi tif veya negatif) , Floating Point Data , BCD Numbers ve Characters ' dır. Işıklı bar , seçenekler üzerinde hareket ettirilir ve < Enter> ile seçim yapılır.

4.1. TAMSAYILAR (Integers)

Sırasıyla birinci ve ikinci sayı bilgisayara okutulur. Bu durumda sayıların pozitif veya negatif olma olasılıkları vardır. Her iki tip sayı da 2 ' lik sayı sisteminde 1 Word (2 byte=16 bit) içerisine yerleştirilir. Pozitif olan tamsayılar için işaret bit ' i , yani 15. bit ' 0 ' dır. Bu bit negatif sayılarda ' 1 ' olur . Negatif sayıların bilgisayar açısından kavramı önceki konularda anlatılmıştı. Bunların en genel olanı pozitif halinin tümleyenine 1 eklemektir . Yani , 2 ' nin tümleyenini kullanmak. Örneğin , 10 ikilik sistemde 0 0 0 0 1 0 1 0 kodlaması ile işlem görürken , -10 ise 1 1 1 1 0 1 1 0 kodlaması ile işlem görür.

```
 10 = 0    0 0 0 1 0 1 0

-10 = 1    1 1 1 0 1 0 1

           +           1
      ----------------------
      1    1 1 1 0 1 1 0
```

Program , koşullara göre gerekli işlemleri yaptıktan sonra sayıların 16 bit üzerinde binary kodlaması elde edilmiş olur . Bundan sonra bit bazında karşılaştırma işlemine geçilir Herbir sayı için temsil edilen word ' ler soldan itibaren sırayla bit bit karşılaştırılır. 15. bit , yani soldan ilk bit işaret bit ' i olduğundan sayıların karakteristiği bu bitler yardımı ile anlaşılır. Önce likle her iki word ' e ait

işaret bit ' leri XOR ' lanır . Sonucun ' 1 ' olması sayılardan birinin pozitif , diğerinin negatif olmasını açıklar . Herhangi bir bit , örneğin 1. word ' ün bit ' i ' 1 ' ile AND ' lenir . Sonuç ' 0 ' ise 1. word ile temsil edilen sayının büyük olduğu anlaşılır . Çünkü , ' 1 ' ile AND ' lenen ve sonucu ' 0 ' yapan bit ancak ve ancak '0' ' dır. Pozitif sayılarda da işaret bit ' i 0 olduğuna göre , bu sayıya pozitif 'dir denir ve diğer sayının negatif olduğu da bilindiğinden 1. sayı 2.sayıdan büyüktür sonucuna varılır.

1. Sayı = 10 ------> 0 0 0 0 1 0 1 0

2. Sayı = -10 ------> 1 1 1 1 0 1 1 0

* işaret bit ' leri XOR ' lanır.

0 ^ 1 = 1

* 1. sayının bit ' i ' 1 ' ile AND ' lenir.

0 & 1 = 0

* Sonucun 0 olması bu bit ' in 0 , dolayısıyla 1.sayının pozitif olduğunu gösterir.

İşaret bit ' lerinin XOR ' lanması ile 0 elde edildiğinde , yani iki sayının da pozitif veya negatif olduğu durumda her iki word 14. bit ' ten 0 .bit ' e doğru teker teker XOR ' lanır. 1 yakalandığı takdirde bit ' lerin farklı olduğu anlaşılır. 1 olan bit hangi sayıya ait ise o sayı diğerinden büyük olur. Bu bit ' i bulmak için de herhangi word ' ün , örneğin 1. ' nin bit ' i ' 1 ' ile AND ' lenir. Sonuç 1 ise 1. sayı 2.sayıdan büyük olur . Çünkü , '1 ' ile AND ' lenen ve sonucu ' 1 ' yapan bit ancak ve ancak 1 dir . Bu durumda diğer bit '0' olacaktır. '1' ile AND ' leme sonucu ' 0 ' ise 1.word ' ün bit 'inin ' 0 ' , diğerinin ' 1 ' olduğu , dolayısıyla bu kez

2. sayının 1.sayıdan büyük olduğu sonucuna varılır. Farklı bit ' ler yakalandığı an sonuç elde edilir ve kalan bit ' ler gözönüne alınmaz.

Son bit ' lere kadar farklılık yakalanmazsa , bu her iki word 'ün bit ' lerinin aynı olduğunu gösterir. Her iki sayının eşit olduğu aşikardır. Sayıların nega tif olması durumunda karşılaştırma mantığında hiçbir değişiklik yoktur.

7 6 5 4 3 2 1 0

1. Sayı = 10 -----> 0 0 0 0 1 0 1 0

2. Sayı = 11 -----> 0 0 0 0 1 0 1 1

Görüldüğü gibi 1. bit ' e kadar bütün bit ' ler aynı. Fakat 0. bit ' te farklılık yakalanıyor.

0 ^ 1 = 1

* 1. Sayıya ait bit , yani ' 0 ' , ' 1 ' ile AND ' leniyor.

0 & 1 = 0

* Sonucun '0' olması diğer word ' e ait olan bit ' in '1' olduğunu , dolayısıyla 2. sayının 1.sayıdan daha büyük olduğunu gösteriyor.
Sayılar negatif ise ;

7 6 5 4 3 2 1 0

1. Sayı = -10 ----> 1 1 1 1 0 1 1 0

2. Sayı = -11 ——> 1 1 1 1 0 1 0 1

Bu durumda farklı bit ' ler 1. bit ' lerde yakalanıyor. Aynı mantıkla '1' olan bit bulunuyor. Bu bit sayının üstünlüğünü temsil ettiğinden -10 , -11 ' den daha büyüktür sonucuna varılır.

4.2. KAYAN NOKTALI (Floating Point) SAYILAR

Float tipi sayıların bilgisayar ortamındaki min. değeri -3.4×10^{38},max. değeri $+3.4 \times 10^{38}$ ' dir. Ana menüde " FLOATING POINT DATA " bölümünden float sayıların bit bazında karşılaştırma işlemine geçilir. Integer tip sayılarda kodlama 16 bit ' lik alanda olurken bu alan float tipler için 32 bit ' tir. Yine, soldan 1. bit işaret bit ' i için ayrılmıştır. Bu 32 bit ' lik register yalnızca sayının mantis bilgisini saklamak için kullanılıyor. Üs bilgisi için 16 bit 'lik bir register düşünülmüştür. Bu register ' ın da ilk bit ' i işareti temsil eder.

Menüden seçim yapıldıktan sonra program, kullanıcının karşılaştırılacak iki sayıyı girmesini ister. Sayılar sırasıyla verildikten sonra mantis ve üs bilgileri tayininin yapılması gerekir. Önceki konularda " Kayan Nokta (Floating Point) Temsilinde " sözedildiği gibi float sayılar derleyici tarafından normalize edilerek yüklenir. Burada hatırlatılmalıdır ki , float tipi sayılar bilgisayar tarafından okutulurken, sayı ne kadar büyük olursa olsun ilk 8 rakam dan sonrası anlamsız olur. Sekizinci rakam da yuvarlatılmış haldedir. Örneğin , float tipinde bir sayı 12345678901234567890.1 olarak verildiğinde aslında bu sayının gerçek değerinin 12345670891470850.0 olduğu görülecektir. Bu özellikten dolayı girilen float sayıların mantis bilgisi , program tarafından 8 dijit ile sınırlandırılmıştır. Önemli olan sayının üs bilgisinin değerlendirilmesidir.

Normalizasyon işleminde sayının 1 ' den küçük veya büyük olmasına göre değerlendirme yapılır. "Mutlak itibariyle" 1 ' den küçük ise ; 10 ile çarpma yapılarak 1 ' den küçük kaldığı müddetçe işleme devam edilir. Şöyle ki ;

Verilen sayı 0.00258 olsun . Normalize edildiğinde 0.258×10^{-2} olması gerekiyor. O halde ;

1. Adım $0.00258 \times 10 = 0.0258$
2. Adım $0.0258 \times 10 = 0.258$

Bu durumda 2 adım sonra sayının istenen formatı elde edildi. Burada adım sayısı

dikkat edilirse üs bilgisini temsil ediyor. Fakat, sayının 1 ' den küçük olduğu için negatif olacaktır.

Mutlak sayı 1' e eşit veya büyük ise bu durumda da arzu edilen for - ma 10 ' a bölme işlemi uygulayarak varılır. Bu kez sayı 1 ' den büyük olduğu müddetçe işleme devam edilir. Örneğin, verilen sayı 25.26 olsun.

1. Adım 25.26 / 10 = 2.526
2. Adım 2.526 / 10 = 0.2526

Son yapılan işlem şarta uymadığından döngüden çıkılır ve son olarak elde edilen değer istenilen formattaki değerdir. Aynı zamanda işlem yapılan adım sayısı da üs bilgisini temsil eder. Bu defa sayının 1 ' en büyük olması nedeniyle pozitif olacak - tır.

Buraya kadar, okutulan iki sayının normalize işlemi tamamlanmış ve sayı mantis ile üs kısımlarına ayrılmıştır. Bundan sonraki aşama mantis ' in 32 bit ' lik, üs ' ün 16 bit ' lik register ' lara yüklenmesidir. Tabii önemli bir nokta işaretlerin gözönüne alınması. Bu yüzden sayıların sırayla işaret kontrolü yapılıyor. Eğer, sayı negatif ise bu durumda öncelikle pozitif hali alınarak 32 bit ' lik alana binary kodda yeleştiriliyor. 2 ' nin komplementi kullanılarak, yani 32 bit ' lik bu bilginin değilini alıp 1 ekleyerek, sayı bilgisayara yüklenebilecek formata dönüştü- rülüyor. Eğer sayı pozitif ise, bu durumda mantis bilgisi doğrudan 32 bit ' lik alana 2 ' lik sistem içerisinde yerleştiriliyor.

Mantisler için yapılan bu işlemlerden sonra aynı tip işlemler bu kez üs bilgileri için de yapılıyor. Yalnız tek fark, üs bilgilerinin 16 bit ' lik alanda binary olarak kodlanması.

Artık bu işlemler sonunda mantis ve üs bilgilerinin ilgili registerlar - daki binary kodlamasını temsil eden dört karakter bilgi mevcut. Karşılaştırılmak üzere iki adet 32 karakterlik mantis bilgisi, iki adet 16 karakterlik üs bilgisi.

Karşılaştırma işleminde öncelikle işaret bit 'leri ele alınıyor.Bu bilgiler karakter setlerinin 1 . karakterine karşılık gelir. Her iki işaret bit bazında XOR ' lanır . Eğer sonuç ' 1 ' ise , yani işaret bit ' leri farklı ise o takdirde sayılardan biri pozitif, diğeri de negatif olacaktır. Bu durumda , tabii ki büyük olan sayı pozitif olandır.

Pozitif sayının tayininde ise bitsel AND (&) operatöründen yararlanılıyor. Pozitif sayının işareti ' 0 ' olacağından , bu bit' i ***'1' ile AND*** ' ler ve sonuçta da '0' elde edersek , diyebiliriz ki bu işarete sahip sayı pozitifdir ve dolayısıyla diğer sayıdan büyüktür. Bu mantıkla birinci sayının işareti '1' ile AND ' leniyor ve sonuç '0' ise girilen ilk sayı ikincisinden büyük, sonuç '0' değilse bu kez ikinci sayı birinciden büyük oluyor.

İşaret bit ' lerinin incelenmesinde farklılık yakalanmazsa bu durum - da iki alternatif var ; ya iki sayı da pozitif , ya da ikisi de negatif. Bu nedenle ,mantisleri temsil eden katarların işaret bit ' leri AND ' leniyor.

1) Sonucun '0' olması durumunda her iki sayının da pozitif olması sözkonusu. İki işaret bit ' i '0' , 0 & 0 = 0. Bu durumda üsleri temsil eden katarlar bit bazında karşılaştırılıyor. Öncelikle işaret bit ' leri XOR ' lanıyor ve sonuç ' 1 ' ise , yani işaretler farklı ise , " 1 ile AND ' leme " yönteminden pozitif olan tayin ediliyor. Dolayısıyla buradan hangi üs' ün büyük olduğu saptanıyor.

İşaret bit ' leri farklı değilse , o takdirde her iki üs katarı 14. bit ' ten 0 . bit ' e kadar karşılıklı XOR ' lanıyor ve ' 1 ' yakalandığı an yine " 1 ile AND ' leme" yöntemi ile büyük sayıya ulaşılıyor. Fakat AND ' leme sonucu burada ' 1 'dir. Üslerin karşılaştırılmasından elde edilen sonuca göre eğer birinci üs ikinciden büyükse birinci sayı ikinciden , büyük değilse ikinci sayı birinciden büyük oluyor.

Üslerin eşit olduğu elde edilirse bu durumda da mantislerin karşılaştırılması yapılıyor. 32 bit ' lik iki katar yukarıda anlatılan mantıkla bit -bit karşılaş-

tırılıyor. Birinci katarın bit'i '1', yani AND'leme sonucu '1' ise birinci sayı ikinciden, '1' değilse ikinci sayı birinciden büyük oluyor. XOR'lama sonucu farklılık yakalanmazsa sayıların eşit olduğu sonucuna varılıyor.

2) Sonucun '1' olması durumunda her iki sayının da negatif olması sözkonusu. İki işaret bit'i '1', 1 & 1 = 1. Bu durumda da üsleri temsil eden katarlar bit bazında karşılaştırılıyor. Yukarıda anlatılan mantıkla karşılaştırılan üs bilgilerinin eşit olması durumunda mantis bilgileri de aynı mantıkla karşılaştırılıp sonuca gidiliyor. Eşit olmadıkları yakalandığında bu defa karşılaştırma işleminden alınan sonuca göre birinci üs ikinciden küçükse birinci sayı ikinciden büyük, büyükse ikinci sayı birinciden büyük olduğu sonucuna varılıyor.

Programın icrasında float sayılarla yapılan işlemler aşağıda gösterildiği gibidir;

* **<u>Her İki Sayı Pozitif ve Üsler Eşit;</u>**

1.Sayı : $0.12 \times 10^3 = (120)_{10}$

mantis (32 bit) : $(0 \ldots\ldots 1\ 0\ 1\ 0)_2 = (12)_{10}$

üs (16 bit) : $(0 \ldots\ldots 0\ 0\ 1\ 1)_2 = (3)_{10}$

2.Sayı : $0.13 \times 10^3 = (130)_{10}$

mantis : $(0 \ldots\ldots 1\ 0\ 1\ 1)_2 = (13)_{10}$

üs : $(0 \ldots\ldots 0\ 0\ 1\ 1)_2 = (3)_{10}$

Üsler öncelikle işaret bit'leri gözönüne alınarak karşılaştırılır:

1.Üs İşareti : 0

2.Üs İşareti : 0 ------> 0 ^ 0 = 0 Her iki üs de aynı işaretli.

	1.Üs		2.Üs	
14.	0	^	**0**	= 0
13.	0	^	**0**	= 0
12.	.	^	.	= 0
.	.		.	
.	.		.	
2.	0	^	**0**	= 0
1.	1	^	1	= 0
0.	1	^	1	= 0

Sonuçta ' 1 ' (Farklı Pozisyon) yakalanmadığı için üslerin eşit olduğu elde edilir. Bu durumda mantislerin bitsel karşılaştırılması yapılır. Öncelikle işaret bit ' leri XOR ' lanır.

1.Mantis İşareti : **0**

2.Mantis İşareti : **0** -----> 0 ^ 0 = 0 Her iki mantis aynı işaretli.

31. bit ' ten 0. bit ' lere kadar karşılaştırılır ;

	1.Mantis		2.Mantis		
31.	0	^	0	= 0	
30.	0	^	0	= 0	
.	.		.		
.	.		.		
2.	0	^	0	= 0	
1.	1	^	1	= 0	
0.	0	^	1	= **1**	---> Fark yakalandı. Birinci register 'ın bit 'i **' 1 ' ile AND 'lenir.**

0 & 1 = 0 Bu durum ikinci register bit ' inin '1' olduğunu, dolayısıyla ikinci sayının büyük olduğunu açıklar.

Sonuç olarak ($0.13 \times 10^3 = 130 > 0.12 \times 10^3 = 120$) .

* **Her İki Sayı Pozitif ve Üsler Farklı ;**

1.Sayı : $0.12 \times 10^3 = (120)_{10}$

mantis (32 bit) : (**0** .....1 0 1 0)$_2$ = $(12)_{10}$

üs (16 bit) : (**0** .....0 0 1 1)$_2$ = $(3)_{10}$

2.Sayı : $0.13 \times 10^2 = (13)_{10}$

mantis : (**0** .....1 0 1 1)$_2$ = $(13)_{10}$

üs : (**0** .....0 0 1 0)$_2$ = $(2)_{10}$

Üslerin işaret bitleri değerlendirilir ;

1.Üs İşareti : **0**

2.Üs İşareti : **0** —> 0 ^ 0 = 0 Her iki üs aynı işaretli.

Üsler 14. bit ' ten 0. bit ' e kadar karşılıklı XOR ' lanır;

	1.Üs		2.Üs	
14.	0	^	0	= 0
13.	0	^	0	= 0
.	.		.	
.	.		.	
2.	0	^	0	= 0
1.	1	^	1	= 0
0.	1	^	0	= **1** —— > Fark yakalandı. Birinci register bit ' i **1 ile AND lenir**

1 & 1 = 1 Bu durum alınan bit ' in ' 1 ' olduğunu ,yani birinci üs' ün daha büyük olduğunu açıklar.

Her iki sayının pozitif olmasından denilebilir ki; büyük üs 'e sahip olan sayı daha büyüktür. O halde birinci sayı ikinciden büyüktür.

($0.12 \times 10^3 = 120 > 0.13 \times 10^2 = 13$).

* **Her İki Sayı Negatif ve Üsler Farklı**

1.Sayı : $-0.12 \times 10^3 = (-120)_{10}$

mantis (32 bit) : (**1** .....1 0 1 1 0 $)_2 = (-120)_{10}$

üs (16 bit) : (**0** .....0 0 0 1 1 $)_2 = (3)_{10}$

2.Sayı : $-0.13 \times 10^2 = (-13)_{10}$

mantis : (**1** .....1 0 1 0 1 $)_2 = (-13)_{10}$

üs : (**0** .....0 0 0 1 0 $)_2 = (2)_{10}$

Üslerin işaret bit ' leri XOR ' lanır;

1.Üs İşareti : **0**

2.Üs İşareti : **0** ---> 0 ^ 0 = 0 Her iki üs aynı işaretli.

14. bit ' ten 0 .bit ' e kadar bit -bit karşılaştırma yapılır ;

	1.Üs	2.Üs
14.	0	0 = 0

	1.Üs	2.Üs	
13.	0	0	= 0
.	.	.	= 0
.	.	.	
2.	0	0	= 0
1.	1	1	= 0
0.	1	0	= **1** ---> Fark yakalandı. Birinci register bit 'i **1 ile AND' lenir**

1 & 1 = 1 Bu durum alınan bit ' in '1' olduğunu ,dolayısıyla birinci üs ' ün daha büyük olduğunu açıklar.

Her iki sayının negatif olmasından dolayı , küçük üs ' e sahip sayı daha büyük olur. Yani ,ikinci sayı birinci sayıdan büyüktür.

$(-0.13 \times 10^{2} = -13 > -0.12 \times 10^{3} = -120)$.

4.3. BCD SAYILAR

BCD kodlamada, her bir rakam 4 bit ' lik register ' larda saklandığı için, toplam (basamak sayısı x 4) bit ' lik register ' a ihtiyaç duyulur. İşaret için 1 flip_flop kullanılırken , rakamlar için her rakam başına 4 flip_flop kullanılır. Programda verilen sayılar rakam sayısına bağlı olarak binary kodlanmış ve soldan ilk bit işaret bit ' i olarak ayrılmıştır.

Menüden seçim yapıldıktan sonra program, kullanıcının karşılaştırılacak iki sayıyı girmesini ister. Sayılar sırayla verildikten sonra rakamların binary kodlama işlemi yapılır.Teker teker alınan rakamlar 4 bit 'lik bir alan içerisinde temsil edilir. Negatif sayıların temsilinde işaret_nitelikli temsil (sign_magnitude) esas alınmıştır. Yani , girilen negatif sayının pozitif hali BCD kodlanırken , sayının register ' daki temsilinde soldan ilk bit '1' yapılır. Bu bit pozitif sayılar için ' 0 ' dır. Pozitif ve negatif sayıların BCD kodlaması , programın icrası ile aşağıda gösterildiği gibi gerçekleşir :

1.Sayı : 1234

işaret	1	2	3	4
0	0001	0010	0011	0100

2.Sayı : -5978

işaret	5	9	7	8
1	0101	1001	0111	1000

Verilen sayıların basamak adeti birbirinden farklı olduğunda program basamak sayısı az olan sayıyı kodlarken eksik olan her basamak için sağdan 4 bit ' lik '0' (sıfir) yerleştirir. Örneğin, sayılar 125 ve 12 olduğunda kodlama işlemi aşağıdaki gibidir :

1.Sayı : 125

işaret	1	2	5
0	0001	0010	0101

2.Sayı : 12

işaret	0	1	2
0	0000	0001	0010

Buraya kadar,her iki sayının kodlaması işaretleri de gözönüne alınarak yapılmış ve ilgili karakter setlerine aktarılmıştır. Bundan sonra yapılacak işlem, mevcut karakter setlerinin bit bazında karşılaştırılması. Bunun için programda önce likle işaret bit ' leri kontrol ediliyor. İşaret bit ' lerinin XOR ' lanması sonucu '1' ise o takdirde işaretlerin farklı olduğu yorumuyla pozitif sayının tayini yapılıyor. Daha önceki kontrollerdeki gibi **1 ile AND ' leme** metoduyla pozitif sayıya ulaşılıyor. Birinci karakter setinin , ki bu ilk sayının register ' daki temsili, ilk karakteri 1 ile And ' lendiğinde sonuç ' 0 ' ise o taktirde bu sayının pozitif olduğu kanaatine varılıyor. Çünkü , ancak ve ancak '0' , 1 ile And ' lendiği zaman '0' sonucunu verir. Bu durumda birinci sayı ikinci sayıdan daha büyüktür denir.

İşaret bit ' lerinde farklılık yakalanmazsa, o takdirde temsili karakter setleri soldan ikinci bit ' lerinden itibaren bit_bit karşılaştırılıyor. Dikkat edilen nok-

ta, işaret_nitelikli temsil esas alındığından her iki sayının pozitif veya negatif olması durumunda yapılacak yorumlar. Her iki sayı pozitif iken yapılan yorum ile negatif iken yapılan yorum tamamiyle farklı. Bunun için bit ' lerin karşılaştırılmasında farklılık yakalandığı an , işaret bitleri AND ' leniyor.

1) Sonuç '0' ise her iki sayının pozitif olduğu (0 & 0 = 0) saptanıyor. Bu durumda ,1 ile AND ' leme metodu ile büyük olan sayı bulunuyor. Her iki sayı da pozitif olduğu için normal olarak büyük olan sayı yine büyüklük özelliğini koruyor.

2) Sonuç ' 1 ' ise her iki sayının negatif olduğu (1 & 1 = 1) saptanıyor. Büyük olan sayının bulunmasından sonra , sayıların negatif olmasından dolayı büyük olan sayı esas itiberiyle diğerinden daha küçük oluyor.

Bit ' lerin karşılaştırılmasında hiç farklılık yakalanmazsa , o takdirde karakter setlerindeki her bir bit ' in karşılıklı eşit olduğu sonucuna varılıp , sayıların eşit olduğu bulunur.

1.Sayı : 25

işaret	0	2	5
0	0000	0010	0101

2.Sayı : 125

işaret	1	2	5
0	0001	0010	0101

İşaret bit ' leri XOR ' lanır ;

1.Sayının İşareti : **0**

2.Sayının İşareti : **0 ----> 0 ^ 0 = 0** İki sayı da aynı işaretli.

Karakter setleri karşılaştırılır ;

Bit	1.Sayı		2.Sayı	
1.	0	^	0	= 0
2.	0	^	0	= 0
3.	0	^	0	= 0
4.	***0***	^	***1***	**= 1** --> Farklılık yakalandı.

Sayıların pozitif mi , negatif mi olduğu tayin edilir ;

1.Sayının İşareti : 0

2.Sayının İşareti : 0 ----> **0 & 0 = 0** Her iki sayı pozitif.

İlk karakter setinin mevcut bit ' i (0) , 1 ile AND ' lenir;

0 & 1 = 0 . Bu ikinci karakter bit ' inin '1' olduğunu , yani ikinci sayının birinci sayıdan büyük olduğunu gösterir.

4.4. KARAKTERLER (Characters)

Karakter katarlarının karşılaştırılmasında dikkat edilmesi gereken önemli husus, karakterlerin hem büyük hem de küçük harf olabilecekleridir. Nitelikleri farklı katarların karşılaştırılmasıyla geçerli bir sonuca varılamaz. Bu nedenle, her iki katarın da büyük harflerle oluşturulması en mantıklı yaklaşım metodudur. Katarlar sırasıyla analiz edilerek küçük harflerin büyük harflerle değiştirilmesi sağlanır. Bu işlemi bit bazında gerçekleştiren fonksiyon yapısı aşağıdaki gibidir :

Büyük Harflere Dönüşüm :

String ilk karakterinden başlayarak kontrol edilir. Alınan karakterin ASCII kod karşılığı ilgili fonksiyonda tayin edilir. Sağdan itibaren 5. bit sözkonusu olduğundan, bu bit üzerinde kontrol yapılır. Eğer, bit '1' ise yani karakter küçük harf ise o taktirde karakterin ASCII değerinden 32 eksiltilir. (5. bit ' in 0 yapılması.). Bu şekilde her iki karakter setinin büyük harflerle yazılması sağlanır.

String : "aHmeT"

'a' ---> ASCII 97

7 6 5 4 3 2 1 0

97 = 0 1 1 0 0 0 0 1

x = 97-32 = 65

65 = 0 1 0 0 0 0 0 1 : **'A'**

String[0] = ' A ' ----------> "AHmeT" haline gelir.

Aynı işlemler 2. karakter ve son karaktere kadar olanlar için tekrarlanır. 5. bit ' in ' 0 ' olması durumunda döngüye devam edilir.

Karakter Setlerinin Karşılaştırılması :

Verilen iki katarda herbir karakter karşılıklı kontrol edilir. Bu aşamada karakterler artık 'A' ile ' Z ' arasındadır. Baştan başlayarak karakterler alınır. 1. ve 2. katarlardan alınan karakterlerin ASCII karşılığı bulunur . Bunların binary kodlaması elde edilir .0. bit ' ten 7. bit ' e kadar her iki byte ' ın bitleri XOR ' lanır. Sonucun ' 1 ' olması halinde , iki bit ' in farklı olduğu anlaşılacağından , & operatörü kullanılması ile hangi katarın önce geldiği saptanır.

Şöyle ki ; 1. byte ' tan alınan bit '1 ' ile AND ' lenir.Sonuç yine ' 1 ' ise o takdirde denilir ki 1.katar 2.katardan sonra gelir. Çünkü , AND ' leme ile '1' sonucunu veriyorsa o halde bit ' 1 ' dir. Sonuç ' 0' ise bu kez 2.katar 1. katardan sonra gelir. Hiç '1' yakalanmazsa , bu her iki katarın eşit olması anlamına gelir.

1. Katar : "AHMET"

2. Katar : "MEHMET"

'A' -------> ASCII (65) --------> 0 1 0 0 0 0 0 1

'M' ----- -> ASCII (77) --------> 0 1 0 0 1 0 1 1

Bit ' ler karşılıklı kontrol edilir ;

1.Katar		2.Katar	
0	^	0	: 0
1	^	1	: 0
0	^	0	: 0
0	^	0	: 0
0	^	***1***	: 1 ----> 1. katara ait bit '0' , '1' ile AND '
0	^	0	lenir.
0	^	1	
0	^	1	

0 & 1 = 0 Bu sonuç 2. katara ait bit ' in '1' olduğunu, dolayısıyla 2. katar ' ın 1. katar ' dan sonra geldiğini açıklar.

BÖLÜM 5. PROGRAM LİSTESİ

```
/*COMPARING THE DATA UTILISING THE BITWISE OPERATORS */
/*                KUTLAY ÖMER, MATH. ENGINEER                */

#include "stdlib.h"
#include "stdio.h"
#include "conio.h"
#include "string.h"
#include "math.h"
#include "time.h"
char secim[5][20];
char chr1[40],ch2[40],cap[40],cntrl[5];
char tus1,tus2,*karakter1,*karakter2,sakla[2000],chr1_1[20],ch2_2[40];
char *katar,ln[78],cr[40],cr1[40],cr2[40],bir[2],*cr3,*cr4;
char toplam[40],*kt_1,*kt_2,ch_pw1[20],ch_pw2[20];
int l,num1,num2,i,k,n,p,r,j,say1,say2,elde,cnt,son_cnt,sayi,kalan;
int m,x,o,ch_cont,ssut,ssat,sut,sat,lngth,dus,op_1,op_2,sat1,sut1;
int w,w1,s_2,s_1,status,txt,grnd,kts,kts_1,pwr_1,pwr_2,blk,omer;
int sign;
float f_num1,f_num2,f_num11,mnts_1,mnts_2;
float f_ord,kalan1,f_num,f_w,f_lt1,f_lt2;
double d_num,tbn,us;

void save_scr()
{
 gettext(sut,sat,sut+lngth+1,sat+dus+1,sakla);
 window(sut+1,sat+1,sut+lngth-1,sat+dus);
 clrscr();window(1,1,80,25);
}
void put_scr()
{
 puttext(sut,sat,sut+lngth+1,sat+dus+1,sakla);
```

```
}
void chr_set()
{o=0;
 for (p=0;p<=255;p++)
  if (bir[0]==p) {o=p;p=256;}
}
void shadow()
{
 gotoxy(sut+2,sat+dus+2);memset(ln,' ',lngth);ln[lngth-1]=0;
 cprintf("%s",ln);for (i=sat+1;i<=sat+dus+2;i++) {
 gotoxy(sut+lngth+1,i);cprintf("%s"," ");
                                    }
}
void normal()
{textcolor(15);textbackground(0);
}
void shine()
{textcolor(0);textbackground(15);
}
void clor()
{textcolor(txt+blk);textbackground(grnd);}
void crt_pict()
{i=dus;
 gotoxy(sut,sat);memset(ln,'-',lngth);ln[lngth-1]=0;
 cprintf("%s%s%s","+",ln,"+");dus=i;
 for (i=1;i<=dus;i++)
  {
   gotoxy(sut,sat+i);cprintf("%s","¦");gotoxy(sut+lngth,sat+i);
   cprintf("%s","¦");
  }
```

```
 gotoxy(sut,sat+dus+1);cprintf("%s%s%s","+",ln,"+");normal();
 window(sut+1,sat+1,sut+lngth-1,sat+dus);clrscr();
 window(1,1,80,25);
}
/*-----------------CREATE BACKGROUND PICTURE-----------*/
void head()
{
 normal();
 clrscr();textcolor(14);textbackground(1);
 sut=1;sat=1;dus=1;lngth=78;crt_pict();
 gotoxy(2,2);memset(ln,' ',78);ln[77]=0;txt=4;grnd=15;clor();
 cprintf("%s",ln);gotoxy(1,24);cprintf("%79s"," ");
 gotoxy(3,2);cprintf("B I T  B A S E D  C O M P A R I S O N S");
 txt=14;grnd=1;clor();
 gotoxy(1,24);cprintf(" YILDIZ TECHNICAL UNIVERSITY");
 cprintf("%30s%s"," ","Master's Degree  ");
 memset(ln,' ',78);ln[77]=0;txt=1;grnd=11;clor();
 for (i=4;i<=23;i++)
  {gotoxy(1,i);cprintf("%s%s%s"," ",ln," ");
  }
}
/*-------------NEGATIVE CONVERTION-----------------------*/
void negative_cont()
{/*free(karakter2);karakter2=(char *)malloc(40);*/
 elde=0;karakter2[0]=0;k=0;
 if (strstr(cntrl,"float")==NULL) l=15;else l=31;
 for (i=l;i>=0;i--)
 {cr1[0]=cap[i];cr1[1]=0;op_1=atoi(cr1);
  cr1[0]=cr[i];cr1[1]=0;op_2=atoi(cr1);
  j=op_1+op_2;k=j+elde;
```

```
 if (k==2) {karakter2[i]='0';elde=1;} else
        {itoa(k,karakter1,10);karakter2[i]=karakter1[0];elde=0;}
}
 /*  RETURN ( KARAKTER2 ) */
}/*-----------------GETTING STRINGS-----------------*/
void read_character()
{sut=21;sat=9;dus=4;lngth=37;txt=14;grnd=1;clor();strcpy(cntrl," ");
 crt_pict();txt=0;grnd=0;clor();shadow();txt=4;grnd=15;clor();
 window(22,10,57,13);clrscr();window(1,1,80,25);
 gotoxy(23,11);printf("The 1st String.....:");
 gotoxy(23,12);printf("The 2nd String.....:");gotoxy(45,11);
 scanf("%s",chr1);gotoxy(45,12);scanf("%s",ch2);
}
/*---------- ALL CONVERTIONS ----------*/
void bin_convertion()
{
   itoa(m,karakter1,2);
   /*gotoxy(1,1);printf("%d %s",m,karakter1);getch();*/
   j=strlen(karakter1)-1;
   for (r=0;r<=j;r++) karakter2[7-r]=karakter1[j-r];
   k=7-strlen(karakter1);
   for (r=0;r<=k;r++) karakter2[r]='0';
   karakter2[8]='\0';
}
void hex_convertion()
{
   itoa(m,karakter1,16);
   gotoxy(1,1);printf("%d %s",m,karakter1);getch();
   j=strlen(karakter1)-1;
   for (i=0;i<=j;i++) karakter2[15-i]=karakter1[j-i];
```

```
   k=15-strlen(karakter1);
   for (i=0;i<=k;i++) karakter2[i]='0';
   karakter2[16]='\0';
}
void hex_control()
{for (j=0;j<=7;j++)
 {cr[0]=chr1_1[j];cr[1]=0;op_1=atoi(cr);
  cr[0]=ch2_2[j];cr[1]=0;op_2=atoi(cr);
  if ((op_1^op_2)==1) {k=1;if ((op_1 & 1)==1)
     printf("%d is Greater Than %d",say1,say2);else
     printf("%d is Greater Than %d",say2,say1);j=8;i=16;
                }
 }
}
/*-------------CONVERTION TO CAPITAL LETTERS-------------*/
void small_capital()
{/*karakter2=malloc(40);*/
 for (i=0;i<=strlen(cap)-1;i++)
  { bir[0]=cap[i];chr_set();m=o;bin_convertion();
    strcpy(chr1_1,karakter2);if (chr1_1[2]=='1')
    {x=m-32;cap[i]=x;}
  }
}
/*----------------COMPARING CHARACTER SETS--------------*/
void char_control()
{strcpy(cap,chr1);
 if (strstr(cntrl,"float")==NULL) {small_capital();strcpy(chr1,cap);}
 strcpy(cap,ch2);
 if (strstr(cntrl,"float")==NULL) {small_capital();strcpy(ch2,cap);}
 sat=17;sut=17;dus=1;lngth=45;txt=14;grnd=1;clor();crt_pict();
```

```
txt=4;grnd=15;clor();gotoxy(18,18);cprintf("%44s"," ");
gotoxy(19,18);
n=max(strlen(chr1),strlen(ch2));l=0;
for (i=0;i<=n;i++)
{cr1[0]=chr1[i];cr2[0]=ch2[i];
 strcpy(bir,cr1);chr_set();m=o;bin_convertion();strcpy(chr1_1,kara kter2);
 strcpy(bir,cr2);chr_set();m=o;bin_convertion();strcpy(ch2_2,karak ter2);
  for (ch_cont=0;ch_cont<=7;ch_cont++)
   {cr[0]=chr1_1[ch_cont];cr[1]=0;op_1=atoi(cr);
    cr[0]=ch2_2[ch_cont];cr[1]=0;op_2=atoi(cr);
    if ((op_1^op_2)==1) {l=1;if ((op_1 & 1)==1)
       printf("%s  Is After  %s",chr1,ch2);else
       printf("%s  Is After  %s",ch2,chr1);ch_cont=8;i=n+1;
                    }
   }
}
 if (l==0) { gotoxy(21,18);
    printf("These Character Sets Are The Same");
         }
}
/*-------------------COMPARING THE NUMBERS----------------*/
void control()
{
 sat=10;sut=58;lngth=18;dus=2;save_scr();txt=14;grnd=1;clor();
 crt_pict();
      txt=4;grnd=15;clor();
      gotoxy(59,11);cprintf("%17s",chr1);
      gotoxy(59,12);cprintf("%17s",ch2);
 k=0;sat=17;sut=17;dus=1;lngth=45;txt=14;grnd=1;clor();crt_pict() ;
 gotoxy(18,18);txt=4;grnd=15;clor();cprintf("%44s"," ");
```

```
if (strstr(cntrl,"intgr")!=NULL) gotoxy(29,18);else gotoxy(19,18);
/*-------------------NEGATIVE SITUATION-----------------*/
cr1[0]=chr1[0];cr2[0]=ch2[0];cr1[1]=0;cr2[1]=0;
op_1=atoi(cr1);op_2=atoi(cr2);n=1;
if ((op_1 ^ op_2)==1) { if ((op_1 & 1)==0)
   printf("%d Is Greater Than %d",s_1,s_2);else
   printf("%d Is Greater Than %d",s_2,s_1);n=0;p=0;
                } else p=15;
          /* Bit 's  Are  Being Compared */
for (i=1;i<=p;i++)
{cr[0]=chr1[i];cr[1]=0;op_1=atoi(cr);
 cr[0]=ch2[i];cr[1]=0;op_2=atoi(cr);
 if ((op_1^op_2)==1) { k=1;if ((op_1 & 1)==1)
   printf("%d is Greater Than %d",s_1,s_2); else
   printf("%d is Greater Than %d",s_2,s_1);i=16;n=0;
               }
} if (k==0 && n!=0) {gotoxy(18,18);
  printf("These numbers given are equal to each other");
               }
}
/*-------------CONVERTION TO 16 BIT BASED SYSTEM----------*/
void sixtn_based()
{/*free(karakter1);
 karakter1=malloc(40);
 free(karakter2);karakter2=malloc(40);*/
 i=0;
 while (w>=1)
  {w=w/2;kalan=sayi-(w*2);itoa(kalan,karakter1,10);
   i=i+1;sayi=w;karakter1[1]='\0';
   karakter2[16-i]=karakter1[0];w1=w;
```

```
 }i=i+1;itoa(w1,karakter1,10);karakter2[16-i]=karakter1[0];
 for (j=15-i;j>=0;j--) karakter2[j]='0';karakter2[16]='\0';
 /* RETURN ( KARAKTER2 ) */
}
/*----------------COMPLEMENT OF THE WORD----------------*/
void complement()
{if (strstr(cntrl,"float")==NULL) l=15;else l=31;
 for (j=0;j<=l;j++)
  {if (toplam[j]=='0') toplam[j]='1'; else toplam[j]='0';
  }
}
void message()
{
 gettext(14,20,70,22,sakla);
 sat=20;sut=14;dus=1;lngth=51;txt=14;grnd=1;clor();crt_pict();
 gotoxy(15,21);txt=4;grnd=15;clor();cprintf("%50s"," ");
 txt=4;blk=128;grnd=15;clor();
 gotoxy(23,21);cprintf("There Is Overflow On The Limits");
 getch();
 puttext(14,20,70,22,sakla);
}
/*----------------GETTING THE NUMBERS----------------*/
void read_numbers()
{sut=21;sat=9;dus=4;lngth=37;txt=14;grnd=1;clor();
 crt_pict();txt=0;grnd=0;clor();shadow();txt=4;grnd=15;clor();
 window(22,10,57,13);clrscr();window(1,1,80,25);
 gotoxy(23,11);printf("The First Number.....: ");
 gotoxy(23,12);printf("The Second Number.....: ");
 gotoxy(49,11);scanf("%d",&num1);say1=num1;s_1=say1;
 gotoxy(49,12);scanf("%d",&num2);say2=num2,s_2=say2;strcpy(cntrl, "intgr");
```

```
if (num1>32700 || num1<-31700 || num2>32700 || num2<-31700)
   {message();blk=0;read_numbers();}
}/*----------BASING FOR THE FLOATS----------*/
void f_sixtn_based()
{int kesir,isaret;
 double f_num11;
 /*free(karakter1);free(karakter2);*/
 karakter1=malloc(40);karakter2=malloc(40);
 karakter2="................................";
 i=0;
 while (f_w>=1)
  {f_w=f_w/2;kalan1=modf(f_w,&f_num11);
   f_w=f_num11;kalan1=ceil(kalan1);
   karakter1=fcvt(kalan1,0,&kesir,&isaret);
   i++;karakter1[1]=0;
   karakter2[32-i]=karakter1[0];
  }
 for (j=31-i;j>=0;j--) karakter2[j]='0';karakter2[32]='\0';
 /* RETURN ( KARAKTER2) */
}
/*----------DECLARE THE MANTISIS----------*/
void mantis_sep()
{ int kesir,isaret;
  /*free(karakter1);karakter1=malloc(40);*/
  karakter1=fcvt(f_num,8,&kesir,&isaret);karakter1[strlen(karakter1 )]='\0';
  f_ord=atof(karakter1);
}
/*----------NORMALIZING----------*/
void normalizing()
{i=0;
```

```
if (f_num<0) {kts=-1;kts_1=-1;}else {kts=1;kts_1=1;}
if ((f_num*kts)<1) {i=-1;
  do {f_num11=f_num*kts;f_num=f_num*10*kts;i++;kts=1;}
  while (f_num<1);f_num=f_num11;i=0-i;
              } else
{ if ((f_num*kts)>=1) {do {f_num=(f_num/10)*kts;i++;kts=1;}
                while (f_num>1);
                }
}f_num=f_num*kts_1;
}
/*---------------EXPONENTS BEING COMPARED----------------*/
void comp_exp()
{
p=15;k=0;n=1;
cr[0]=ch_pw1[i];cr[1]=0;op_1=atoi(cr);
cr[0]=ch_pw2[i];cr[1]=0;op_2=atoi(cr);
if ((op_1 ^ op_2)==1) {k=1;if ((op_1 & 1)==0) sign=1;else sign=2;
                i=16;n=0;
                } else
for (i=1;i<=p;i++)
{cr[0]=ch_pw1[i];cr[1]=0;op_1=atoi(cr);
 cr[0]=ch_pw2[i];cr[1]=0;op_2=atoi(cr);
 if ((op_1^op_2)==1) { k=1;if ((op_1 & 1)==1)
   sign=1; else sign=2;i=16;n=0;
                }
 } if (k==0 && n!=0) sign=0;
}
/*---------------BITS COMPARED IN FLOAT----------------*/
void float_scan()
{  n=1;k=0;
```

```
    for (i=1;i<=31;i++)
     {cr[0]=chr1[i];cr[1]=0;op_1=atoi(cr);
      cr[0]=ch2[i];cr[1]=0;op_2=atoi(cr);
      if ((op_1^op_2)==1) { k=1;if ((op_1 & 1)==1)
      printf("%.5f is Greater Than %.5f",f_lt1,f_lt2); else
      printf("%.5f is Greater Than %.5f",f_lt2,f_lt1);i=32;n=0;
                    }
     } if (k==0 && n!=0) {gotoxy(20,18);
    printf("These numbers given are equal to each other");
                    }
}
/*----------BRANCHING ON TO FLOAT NUMBERS----------*/
void float_control()
{cr1[0]=chr1[0];cr2[0]=ch2[0];
 cr1[1]=0;ch2[1]=0;
 op_1=atoi(cr1);op_2=atoi(cr2);gotoxy(19,18);
 if ((op_1^op_2)==1) {if ((op_1 & 1)==0)
   printf("%f Is Greater Than %f",f_lt1,f_lt2);else
   printf("%f Is Greater Than %f",f_lt2,f_lt1);
              } else
 if ((op_1 & op_2)==0) {comp_exp();
   if (sign==0) float_scan();
                else
     /*pwr_1>pwr_2*/
   if (sign==1) printf("%.5f Is Greater Than %.5f",f_lt1,f_lt2);else
          printf("%.5f Is Greater Than %.5f",f_lt2,f_lt1);
              }else
 if ((op_1 & op_2)==1) {comp_exp();
   if (sign==0) float_scan();
                else
```

```
   if (sign==2) printf("%.5f Is Greater Than %.5f",f_lt1,f_lt2);else
                printf("%.5f Is Greater Than %.5f",f_lt2,f_lt1);
                    }
}
/*---------- LOADING THE EXPONENT IN THE REGISTER -------*/
void char_exponent()
{
 if (i<0) {w=abs(i);sayi=w;sixtn_based();strcpy(toplam,karakter2);
    complement();strcpy(cap,toplam);
    strcpy(cr,"0000000000000001");
    negative_cont();
          } else
  {w=i;sayi=w;sixtn_based();}
}
/*---------------------FLOAT NUMBERS----------------------*/
void float_number()
{int kesir,isaret;
 sut=21;sat=9;dus=4;lngth=40;txt=14;grnd=1;clor();
 crt_pict();txt=0;grnd=0;clor();shadow();txt=4;grnd=15;clor();
 l_one:
 window(22,10,60,13);clrscr();window(1,1,80,25);
 gotoxy(23,11);printf("The First Number.....: ");
 gotoxy(23,12);printf("The Second Number.....: ");
 gotoxy(49,11);scanf("%f",&f_num1);f_lt1=f_num1;
 gotoxy(49,12);scanf("%f",&f_num2);f_lt2=f_num2;
 if (f_num1==0 && f_num2==0) return;
 if (f_num1>3.4E+38 || f_num1<-3.4E+38 ||
    f_num2>3.4E+38 || f_num2<-3.4E+38 ) { message();blk=0;float_number();}
 f_num=f_num1;normalizing();mnts_1=f_num;p=kts_1;pwr_1=i;
 char_exponent();strcpy(ch_pw1,karakter2);ch_pw1[16]=0;
```

```
  /*---The First Exponent Has Been Stored In The Register --*/

f_num=f_num2;normalizing();mnts_2=f_num;r=kts_1;pwr_2=i;

  /*---The Second Exponent Has Been Stored In The Register--*/

char_exponent();strcpy(ch_pw2,karakter2);ch_pw2[16]=0;
gotoxy(1,18);
mnts_1=mnts_1*pow(10,9);mnts_2=mnts_2*pow(10,9);
/*-----------------------------------------*/
strcpy(cntrl,"float");
if (p<0) {mnts_1=fabs(mnts_1);mnts_1=ceil(mnts_1);f_w=mnts_1;
      f_num=f_w;f_sixtn_based();strcpy(toplam,karakter2);
      complement();strcpy(cap,toplam);
      strcpy(cr,"00000000000000000000000000000001");
      negative_cont();strcpy(chr1,karakter2);chr1[32]=0;
      free(karakter2);free(karakter1);
      } else
 {
 f_w=mnts_1;f_num=f_w;f_sixtn_based();strcpy(chr1,karakter2);
 chr1[32]='\0';free(karakter2);free(karakter1);
 }
if (r<0) {mnts_2=fabs(mnts_2);mnts_2=ceil(mnts_2);f_w=mnts_2;
       f_num=f_w;f_sixtn_based();strcpy(toplam,karakter2);
       complement();strcpy(cap,toplam);
       strcpy(cr,"00000000000000000000000000000001");
       negative_cont();ch2[0]='\0';strcpy(ch2,karakter2);
       ch2[32]=0;free(karakter2);free(karakter1);
      } else
 {
```

```
 f_w=mnts_2;f_num=f_w;f_sixtn_based();strcpy(ch2,karakter2);
 ch2[32]='\0';
 free(karakter2);free(karakter1);
 }
 sat=17;sut=17;dus=1;lngth=51;txt=14;grnd=1;clor();crt_pict();
 gotoxy(18,18);txt=4;grnd=15;clor();cprintf("%50s"," ");
 txt=4;grnd=15;clor();
 gotoxy(1,1);printf("1.%s",ch_pw1);
 printf("\n2.%s",ch_pw2);
 printf("\n1.%s",chr1);
 printf("\n2.%s",ch2);
 getch();
 float_control();getch();
 movetext(17,21,70,23,17,17);
 goto l_one;
}
/*-------------BINARY CODED DECIMAL CONTROL--------------*/
void bcd_cont()
{k=0;sat=17;sut=17;dus=1;lngth=45;txt=14;grnd=1;clor();crt_pict() ;
 gotoxy(18,18);txt=4;grnd=15;clor();cprintf("%44s"," ");
 txt=4;grnd=15;clor();gotoxy(19,18);
 cr1[0]=cr[0];cap[0]=cr2[0];cr1[1]=0;cap[1]=0;
 op_1=atoi(cr1);op_2=atoi(cap);n=1;l=op_1;r=op_2;
 if ((op_1 ^ op_2)==1) { if ((op_1 & 1)==0)
    printf("%s Is Greater Than %s",chr1,ch2);else
    printf("%s Is Greater Than %s",ch2,chr1);n=0;p=0;
                } else p=strlen(cr)-1;
          /* Bit 's  Are  Being Compared */
 for (i=1;i<=p;i++)
 {cr1[0]=cr[i];cr1[1]=0;op_1=atoi(cr1);
```

```
 cap[0]=cr2[i];cap[1]=0;op_2=atoi(cap);
 if ((op_1^op_2)==1) { k=1;if ((l & r)==0) {  /* positive */
   if ((op_1 & 1)==1)
   printf("%s is Greater Than %s",chr1,ch2); else
   printf("%s is Greater Than %s",ch2,chr1);i=p+1;n=0;
                                }else
                        if ((l & r)==1) {  /* negative */
   if ((op_1 & 1)==1)
   printf("%s is Greater Than %s",ch2,chr1); else
   printf("%s is Greater Than %s",chr1,ch2);i=p+1;n=0;
                                }
                }
 } if (k==0 && n!=0) {gotoxy(18,18);
   printf("These numbers given are equal to each other");
                }
}
/*-----------------DEFINE THE BCD FORM-------------------*/
void bcd_alloca()
{/*free(karakter1);free(karakter2);*/
 /*karakter1=malloc(40);karakter2=mallo/c(40);*/
 cr1[strlen(cr1)]='\0';toplam[1]='\0';
 if (cr1[0]=='-') k=1;else k=0;
 for (i=k;i<=strlen(cr1)-1;i++)
 {cr2[0]=cr1[i];cr2[1]='\0';
  strcpy(karakter1,cr2);karakter1[1]=0;m=atoi(karakter1);
  bin_convertion();gotoxy(1,20);
  for (j=4;j<=7;j++) {cr2[0]=karakter2[j];cr2[1]='\0';
                strcat(toplam,cr2);
                }
 /* RETURN (TOPLAM) */
```

```
 }
}/*----------------READ THE BCD NUMBERS----------------*/
void read_bcd()
{sut=21;sat=9;dus=4;lngth=37;txt=14;grnd=1;clor();
 crt_pict();txt=0;grnd=0;clor();shadow();txt=4;grnd=15;clor();
 strcpy(chr1,"12");
 while (atoi(chr1)!=0 || atoi(ch2)!=0)  {
 window(22,10,57,13);clrscr();window(1,1,80,25);
 gotoxy(23,11);printf("The First Number.....: ");
 gotoxy(23,12);printf("The Second Number.....: ");
 gotoxy(49,11);scanf("%s",chr1);
 gotoxy(49,12);scanf("%s",ch2);
 if (chr1[0]=='-') toplam[0]='1';else toplam[0]='0';
 strcpy(cr1,chr1);bcd_alloca();strcpy(cr,toplam);
 if (ch2[0]=='-') toplam[0]='1';else toplam[0]='0';
 strcpy(cr1,ch2);bcd_alloca();strcpy(cr2,toplam);
 /*-------------------------------*/
  if (strlen(cr)>strlen(cr2))
   {for (i=1;i<=strlen(cr2)-1;i++) cap[i-1]=cr2[i];cap[i-1]=NULL;
    k=strlen(cr)-strlen(cr2)-1;
    for (j=1;j<=k;j++) cr2[j]='0';cr2[j+1]='\0';
    strcat(cr2,cap);
   }else
  if (strlen(cr2)>strlen(cr))
   {for (i=1;i<=strlen(cr)-1;i++) cap[i-1]=cr[i];cap[i-1]=NULL;
    k=strlen(cr2)-strlen(cr)-1;
    for (j=1;j<=k;j++) cr[j]='0';cr[j+1]='\0';
    strcat(cr,cap);
   }
  bcd_cont();getch();movetext(17,21,70,23,17,17);
```

```
                    } /*while*/
  return;
}
/*----------NUMBER ALLOCATION----------*/
void num_sep()
{ if (say1<0) {w=abs(say1);sayi=w;sixtn_based();strcpy(toplam,karakter2);
    complement();strcpy(cap,toplam);
    strcpy(cr,"0000000000000001");
    negative_cont();strcpy(chr1,karakter2);chr1[16]=0;
          } else
 {
  w=s_1;sayi=w;sixtn_based();strcpy(chr1,karakter2);chr1[16]='\0';
 }
 if (say2<0) {w=abs(say2);sayi=w;
   sixtn_based();strcpy(toplam,karakter2);
   complement();strcpy(cap,toplam);strcpy(cr,"0000000000000001");
   negative_cont();ch2[0]='\0';strcpy(ch2,karakter2);ch2[16]=0;
          } else
 {
  w=s_2;sayi=w;sixtn_based();
  strcpy(ch2,karakter2);ch2[16]=0;
 }
control();
}
void arrays()
{strcpy(secim[1]," INTEGERS         ");
 strcpy(secim[2]," FLOATING-POINT    ");
 strcpy(secim[3]," BCD NUMBERS       ");
 strcpy(secim[4]," CHARACTERS        ");
 strcpy(secim[5]," QUIT              ");status=0;
```

```
}
void cover()
{memset(ln,' ',50);ln[49]=0;txt=1;grnd=11;clor();
 for (i=5;i<=23;i++)
  {gotoxy(26,i);cprintf("%s%s%s"," ",ln," ");
  }
}
void remark()
{
 cover();
 if ((cnt)!=5) {
 sut=27;p=sat;sat=sat1-1;dus=6;lngth=47;txt=14;grnd=1;clor();
 crt_pict();txt=8;grnd=15;clor();
 window(28,sat+1,73,sat+6);clrscr();window(1,1,80,25);
 txt=0;grnd=0;clor();shadow();txt=1;grnd=15;clor();
 /*----------------------------------------*/
 switch(cnt)
  {case 1:gotoxy(29,sat+2);
  cprintf("Numbers Are Loaded Into 8 Bit-register .");
  gotoxy(29,sat+3);
  cprintf("The Limit For The Integers Is From -32768");
  gotoxy(29,sat+4);
  cprintf("To + 32767.So, When You Give A Number That");
  gotoxy(29,sat+5);
  cprintf("Exceedes The Range , The Message Is Shown.");
  break;
   case 2:gotoxy(29,sat+2);
  cprintf("A 32_bit Register Is Used To Load The Mantiss");
  gotoxy(29,sat+3);
  cprintf("Data,An 8_bit Register For The Exponent Data.");
```

```
 gotoxy(29,sat+4);
 cprintf("Numbers Can Be Given Up To 38 Times The Power");
 gotoxy(29,sat+5);
 cprintf("Of 10 ,Included Negative Numbers.");
 break;
  case 3:gotoxy(29,sat+2);
 cprintf("The Number Of The Bit In The Register Depends");
 gotoxy(29,sat+3);
 cprintf("On The Number Of Digits . 4_Bit Area Is");
 gotoxy(29,sat+4);
 cprintf("Allocated For Each Digit . One Flip_Flop Is");
 gotoxy(29,sat+5);
 cprintf("Used To Load The Sign.");
 break;
  case 4:gotoxy(29,sat+2);
 cprintf("The First Step Done On The Strings Is To");
 gotoxy(29,sat+3);
 cprintf("Convert All Small Letters To Capital");
 gotoxy(29,sat+4);
 cprintf("Letters,If They Are Exist . The Other Step");
 gotoxy(29,sat+5);
 cprintf("Is To Compare The Strings bit by bit");
 break;
  }
sat=p;     }
}
void inkey()
{gettext(26,6,76,23,sakla);
 proceed:num1=1;num2=1;status=0;cr3[0]='1';cr4[0]='1';gotoxy(80,2 5);
 f_num1=1;f_num2=1;remark();
```

```
tus1=getch();
if (tus1=='\0') {
  tus2=getch();
  if (tus2==';')  {
    sat=10;sut=56;lngth=18;dus=2;save_scr();shine();crt_pict();
    normal();
    gotoxy(58,11);printf("%16s",chr1);
    gotoxy(58,12);printf("%16s",ch2);
              }
  if (tus2=='H') {
    txt=4;grnd=15;clor();gotoxy(sut1,sat1);cprintf("%s",secim[cnt]);
    cnt=cnt-1;sat1=sat1-1;if (sat1<sat) {cnt=son_cnt;sat1=sat+son_cnt-1;
                              }
    txt=14;grnd=1;clor();gotoxy(sut1,sat1);cprintf("%s",secim[cnt]);
    remark();
            }
  if (tus2=='P') {txt=4;grnd=15;clor();gotoxy(sut1,sat1);cprintf("%s",secim[cnt]);
    cnt=cnt+1;sat1=sat1+1;if (sat1>sat+son_cnt-1) {cnt=1;sat1=sat;}
    txt=14;grnd=1;clor();gotoxy(sut1,sat1);cprintf("%s",secim[cnt]);
    remark();
             }
           } /* TUS2*/
      else
  if (tus1==13) { puttext(26,6,76,23,sakla);
    switch(cnt)
    {case 1: while (num1!=0 | num2!=0)
       {
          read_numbers();num_sep();getch();
          movetext(60,4,77,8,60,10);
          movetext(16,20,70,23,17,17);
```

```
        }
      status=1;tus1=27;
      break;
      case 2:while (f_num1!=0 | f_num2!=0)
        {   float_number();
        }
          status=1;tus1=27;
      break;
      case 3:read_bcd();status=1;tus1=27;break;
      case 4:while (cr3[0]!='0' | cr4[0]!='0')
           { read_character();strcpy(cr3,chr1);strcpy(cr4,ch2);
            char_control();getch();
            movetext(60,4,77,8,60,10);
            movetext(16,20,70,23,17,17);
           }
      status=1;tus1=27;
      break;
      case 5:exit(0);break;
     }
              }
  if (tus1==27)   {
switch(status)
{case 0: /*sound(700);delay(10);nosound();*/exit(0);break;
 case 1: head();cnt=1;son_cnt=5;sut=4;sat=5;dus=7;lngth=20;txt=14;grnd=1;
      clor();crt_pict();grnd=0;shadow();sut=5;sat=7;sut1=sut;sat1=sat;
      arrays();grnd=15;clor();
      window(5,6,23,12);clrscr();window(1,1,80,25);
      txt=4;grnd=15;clor();
      for (i=1;i<=5;i++) { gotoxy(5,6+i);cprintf("%s",secim[i]);}
      txt=14;grnd=1;clor();gotoxy(sut1,sat1);cprintf("%s",secim[1]);
```

```
        break;
 }
                }
 goto proceed;
}
void main_menu()
{head();cnt=1;son_cnt=5;sut=4;sat=5;dus=7;lngth=20;txt=14;grnd=1; clor();
 crt_pict();txt=0;grnd=0;clor();shadow();
 sut=5;sat=7;sut1=sut;sat1=sat;txt=8;grnd=15;clor();
 window(5,6,23,12);clrscr();window(1,1,80,25);
 txt=4;grnd=15;clor();
 for (i=1;i<=5;i++) {gotoxy(5,6+i);cprintf("%s",secim[i]);}
 txt=14;grnd=1;clor();gotoxy(sut1,sat1);cprintf("%s",secim[1]);
 inkey();
}

/*---------- MAIN PROGRAM ----------*/
main()
{int kesir,isaret;
 karakter1=malloc(40);karakter2=malloc(40);
 clrscr();arrays();main_menu();
}
```

SONUÇ

Sıralama işleminde kullanılmak amacı ile tasarlanmış asenkron karşılaştırıcının çalışma prensibi , C programlama dili kullanılarak simule edildi . Donanımdaki tüm işlemler , programda özel işlemciler kullanılarak aynen gerçekleştirildi. Register bilgileri , temsili karakter katarlarına orjinal durumunu gösterir biçimde yüklendi. İşlem, tasarımdaki karşılaştırma mantığı gözönüne alınarak yapıldı.

Sonuçta görüldü ki , tasarımdaki işlemler program içerisinde de aynen gerçekleşti. O halde denilebilir ki , karşılaştırıcının çalışma ilkesi yazılım destekli benzetimi yapılmak suretiyle desteklendi.

KAYNAKLAR

[1]. BAYBURAN, B., *Microsoft Standard C* , Beta Yayıncılık,İstanbul,1991. 195-196.

[2]. GLENN , A.G. ve LIU,Y. , *Microcomputers For Engineers And Scientists.*, Prentice / Hall International Editions , 1980.

[3]. GRADY, M. T., *Turbo C Programming Principles & Practices* ., McGRAW-HILL Book Company,Singapore,1989,64-66.

[4]. LAWRENCE, H. M. ve ALEXANDER E. Q., *Joy Of C* , 2nd Edition . John Wiley & Sons , Inc.,1991.

[5]. MANO M., *Computer System Arcitecture* ., Prentice / Hall International Inc., Second Edition , 1982 , 75-91.

[6]. MARŞOĞLU, A. , *A Bit Based Asynchronous Comparator Algorithm For ASCII Code* , Yıldız Teknik Üniversitesi Dergisi, 1995/1.,61-63.

ÖZGEÇMİŞ

Doğum Tarihi	: 16/03/1972
Doğum Yeri	: İstanbul
İlköğretim	: Gelibolu Namık Kemal İlkokulu, 1978-1983.
Orta ve Lise Öğretim	: Gelibolu Lisesi ,1983-1989.
Yüksek Öğretim	: Yıldız Teknik Üniversitesi Matematik Mühendisliği , 1989-1993. Bölüm 3. 'sü olarak mezun oldu.
Yüksek Lisans	: Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Matematik Mühendisliği, 1993.