1.BÖLÜM

1.SAYI SİSTEMLERİ

Başka bir gezegeni ziyaret etmedikçe, nesneleri ondalık sayılar kullanarak sayarak hayatımızı geçiririz. Matematikçiler ondalık ifadesini 10 tabanında sayma sistemi olarak değerlendirirler. Çünkü bu 0 ile 9 arasında 10 rakamdan oluşmaktadır.

İnsanoğlu ondalık sayılarla daha rahat sayabilmektedir. (Belki de on el ve ayak parmağı olduğu için.) Ama bilgisayarlar böyle değil. Bunun yerine bilgisayarlar 2'lik tabandaki (binary) sayma sistemiyle sayarlar. Bu tabanın yalnız iki rakamı vardır. 0 ve 1. Bundandır ki bilgisayarlar kendi düzeylerinde haberleşirler. (Siz bunu assembly dili kullandığınızda yaparsınız.)

İkilik (binary) sayma sistemine alışık olmalısınız. Binary yanında, assembly-dili programcıları bir diğer sayma sitemi olan 16 tabanındaki (hexadesimal) sayma sitemine de iyi derecede alışık olmalıdırlar.

Bu bölüm daha önce hiçbir birikimi olmayan okuyucular için bilgisayar sayı sistemlerini ele alır. Eğer zaten binary ve hexadesimal sayma sitemlerini biliyorsanız bu bölümü atlayabilirsiniz.

1.1.İKİLİK (BINARY) SAYMA SİSTEMİ

Bir bütün program talimatlarını ve veriyi belleğinden alır. Bellek binlerce elektrikli bileşen içeren tümleşik daireler (ya da yongalar) kapsar. Işık anahtarları gibi bu bileşenlerde yalnız iki olası ayar vardır. "On" ya da "off". Hala yalnızca bu iki ayarla, bellek bileşenleri kombinasyonları herhangi bir boyutun sayılarınız gösterebilir. Nasıl mı? Okumaya devam edin.

Bellek bileşeninin On ve Off ayarları ikilik sayı sisteminin iki rakamıyla uyuşmaktadır. Bu bilgisayarın temel sistemidir. Yalnızca iki rakamımız var. 1 (On) ve 0 (Off). İkilik sayı sistemi bir 2 tabanı sitemidir. Yeniden, bu durum ikilik sayı sistemini 10 rakama sahip (0 ile 9 arasında) standart ondalık sayı sisteminden ayırır.

Belleğin anahtara benzer bileşenleri "bits" (binary digits'in kısaltması) diye adlandırılır. Gelenek olarak, bir bit "on" olduğunda 1 değerini "off" olduğunda 0 değerini alır. Bu gösterir ki bir ondalık rakamı (0 ile 9 arasında) düşünene kadar sınırlandırır.

Şimdi siz ondalık rakamları 9'dan büyük formda sayılara birleştirebiliyorsanız, binary rakamları da 1'den büyük sayılara birleştirebilirsiniz.

Bildiğiniz gibi 9'dan büyük ondalık sayıları göstermek için ek "onlu" rakamlara ihtiyacınız var. Benzer şekilde 99'dan büyük ondalık sayıları göstermek için bir "yüzlü" rakama gerek var ve böyle devam eder. Her ondalık rakam için hemen sağına 10'un katları rakamları eklemelisiniz.

Örneğin;

324 sayısını göstermek için

(3*100)+(2*10)+(4*1)

ya da böyle

$(3*10^2)+(2*10^1)+(4*10^0)$

Böylece, her ondalık rakamda 10'un gücü önündeki rakamdan daha büyüktür.Benzer kural ikilik sistemde de uygulanır. Burada *her ikilik rakamda ikinin gücü önündeki rakamdan daha büyüktür.* En sağdaki bit 2^0 'ın katına sahiptir (Ondalık 1). Sonraki bit 2^1'nin katına sahiptir (Ondalık 2) ve böyle gider...

Örneğin: İkilik 101 sayısı ondalık 5 değerindedir.

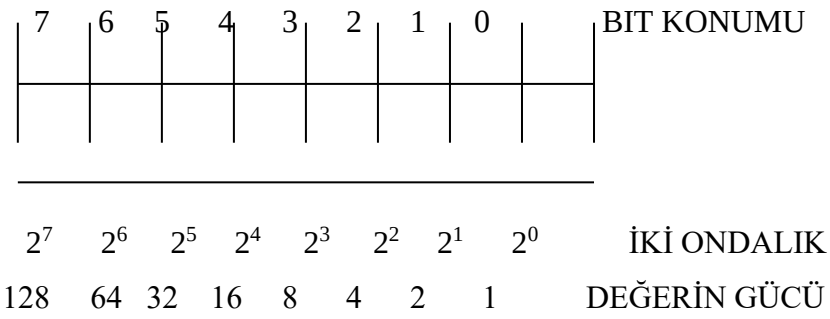
Çünkü:

$101_2 = (1*2^2) + (0*2^1)+ (1+2^0)$

$= (1*4) + (0*2) + (1*1)$

$= 5_{10}$

Binary sayılarının nasıl inşa edildiğini anlayabiliyor musunuz. Verilen herhangi bir bit konumunun değerini bit durumunun öncellenmesinin ağırlığını iki katına çıkarırsınız. Böylece ilk sekiz bitin binary ağırlığı 1, 2, 4, 8, 16, 32, 64 ve 128 olur. Bu durum Şekil 0-1'de gösterilmiştir.



Şekil .1.1 Sekiz ikili rakamın ağırlıkları.

Ondalık sayıyı ikilik sayıya çevirmek için bir dizi basit çıkarama işlemi yapın. Her çıkarma prosedürü bir tekil ikilik rakam (bit) değeridir.

Başlamak için en büyük olası ikilik ağırlığı ondalık değerden bu bit konumuna "1 girin". Sonra sonraki en büyük ağırlığı sonuçtan çıkarın ve bu bit konumuna bir "1" girin. Sonuç "0"

olana kadar devam edin. Herhangi bir güncel ondalık değerden çıkarılamayan ağırlıklı bir konuma bir “0” girin. Örneğin: Ondalık 50’yi ikiliğe çevirmek için.

```
    50
-   32      bit konumu 5 = 1
    18
-   16      bit konumu 4 = 1
    .
     2
-    2      bit konumu 1 = 1
     0
```

Değer bit değerlerine (3,2 ve 0 bitlerine) bir “0” girmek sonuç değerini vermeyi sağlar. Sonuç “110010”

Ondalık 50’nin eşiti ikilik değerin gerçekten 110010 olduğunu doğrulama için, 1’in konumlarının ondalık ağırlıklarını toplayın.

```
    32   (bit 5)
    16   (bit 4)
-    2   (bit 1)
   50
```

1.1.1.Sekiz Bit Bir Byte Eder

Apple II Ailesi Commodore 64 ve VIC-20, Radio Shack TRS-80 ve diğer ünlü mikrobilgisayarların mikroişlemcileri 8-bit çevresinde tasarlanırlar. Sekiz-bit mikroişlemciler zaten böyle isimlendirilmiştir. Çünkü bunlar bilgiyi bir anda sekiz bit olarak işlerler. İşleme sekiz bitten daha fazla olduğunda bu mikroişlemciler ek işlemler gerçekleştirmelidir.

Bilgisayar terminolojisinde bilginin bit 8-bit birimi “byte” olarak adlandırılır. Sekiz bitle, bir byte 0 (ikilik 00000000) ile 255 (ikilik 11111111) arasında ondalık değerlerle gösterilir.

Bir byte işleme biriminin temelidir. Mikrobilgisayarlar belleklerinin tutabildiği byte numaralarının açıklaması içinde tanımlıdır. Mikrobilgisayar üreticileri belleği 1,024 byte’lık blokların içinde inşa ederler. Bu belirli miktar 2^{10} byte gösteriminde bilgisayarların ikilik yönelimini yansıtır.

1.024 değerinin bir standart kısaltması vardır. Bu K harfidir. Bu sayede bir bilgisayar 256*1.024 (ya da 262,144) byte içeren “256K bellek” içerir.

1.1.2.İkilik Sayıları Toplamak

İkilik sayıları ondalık sayılarla aynı yolla toplayabilirsiniz: Bir sütundan sonrakine fazlayı taşıyarak. Örneğin ondalık 7 ve 9 değerlerini toplamak için: “onlar” sütunundan üretilmiş doğru sonucu (16) bir “1” taşımalısınız. Benzer bir şekilde, 1 ve 1 ikilik değerini toplamak için, “ikiler” sütunundan üretilmiş doğru değere (10) bir “1” taşımalısınız.

Çoklu-bit sayılarını toplamak biraz karmaşık olur. Çünkü siz bir önceki sütundan bir elde sağlamalısınız. Örneğin bu işlem iki elde gerektirir.

```
   1011
+    11
------
   1110
```

En sağdaki sütunun toplanması (1+1) “0” sonucunu üretir ve 1’in bir eldesi ikinci sütunun içinedir. Eldeyle birlikte ikinci sütunun toplanması (“1+1”+1) “1” sonucunu üretir ve bir elde de üçüncü sütunun içinedir.

İkilik toplamada genel kurallar bu tabloda gösterilmiştir.

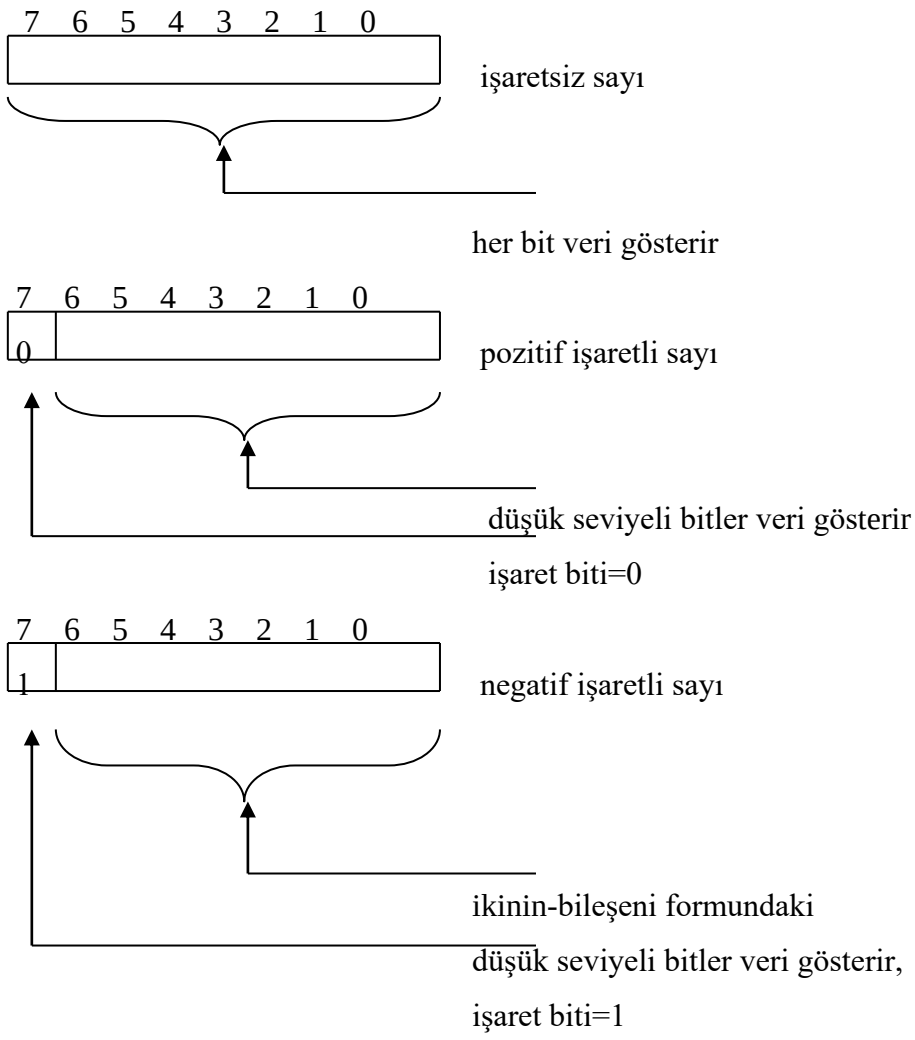
Tablo.1.1. İkilik Toplamada Genel Kurallar:

Girdiler			**Sonuçlar**	
Operand #1	**Operand #2**	**Elde**	**Toplam**	**Elde**
0	**0**	**0**	**0**	**0**
0	**1**	**0**	**1**	**0**
1	**0**	**0**	**1**	**0**
1	**1**	**0**	**0**	**1**
0	**0**	**1**	**1**	**0**
0	**1**	**1**	**0**	**1**
1	**0**	**1**	**0**	**1**
1	**1**	**1**	**1**	**1**

1.1.3.İşaretli Sayılar

Şimdiye kadar ikilik içindeki "işaretsiz sayıların" nasıl gösterildiğini tartıştık. Daha önce andığımız gibi bir işaretsiz sayının içindeki her bit onun konumunu yansıtan bir ağırlığa sahiptir. En sağdaki (ya da) en az-belirgin bit 1'in ağırlığına sahiptir. Her daha belirgin bit iki kez ağırlığa sahipse bu onun öncelidir. Böylelikle, eğer bir byte'taki her sekiz bit 0'sa byte 0 değerine sahiptir. Eğer hepsi 1 ise, byte 255 değerine sahiptir.

Bununla beraber, siz sık sık pozitif ve negatif değerleri işletmek isterseniz. Bu "işaretli sayılar" dadır. Bir byte bir işaretli sayı yalnızca yedi en az-belirgin bit (0-6 arası) veriyi gösterir. En belirgin bit (7) sayının işaretini belirtir. İşaret biti 0 ise sayı pozitif ya da sıfır ve 1 ise sayı negatiftir. Şekil 0-2 işaretli ve işaretsiz byte'ların sırasını gösterir.



Şekil.1.2. İşaretli ve işaretsiz sayıların gösterimi.

İşaretli sayı tutulurken, bir tekil byte 0 (ikilik 00000000) ve +127 (ikilik 01111111) arasında pozitif değerleri gösterebilir. Ve –1 (ikilik 11111111) ve –128 (ikilik 10000000) arasında negatif değerleri gösterir.

Not: -1 ikilik olarak 11111111'dir. Bu 10000001'den daha kolay değil midir? (Bu, 1 ve bir eksi işaret bitidir.) Hayır. Bu yanlış yanıtlar üretebilir. Düşünün Örneğin, +1 ve –1 eklerseniz ne olur. Yanıt kesinlikle 0 olmalı.

```
    00000001    = +1
+ 10000001    = -1
----------
    10000010    = -2
```

Böylece -1 göstermek için birkaç yol gerekir. +1 eklediğimizde 0 elde ederiz. –1 için 11111111 ile geldiğinde: bu doğru yanıtı üretir. Bunu test etmek için toplamamızı yeniden yapalım...

```
    00000001  = +1
+  11111111  = -1
  1 00000000  =  0
```

Başındaki fazladan 1-bit "elde"dir. Artık bit toplamadan gelir . Basitçe yok sayabiliriz.

1.1.4.İkinin Bileşeni

-1'deki gibi tüm negatif işaretli sayılar özel bir formda gösterilmiştir. Bu yapılan toplamalarda doğru yanıtları üretir. Buna "ikinin bileşeni formu" denir.

Negatif sayıların ikilik gösterimini bulmak için (bu ikinin bileşen formunu bulmaktır.) basitçe sayının pozitif formunu alın ve her biti tersine çevirin. (--- her 1'i 0'a ve her 0'ı 1'e ---) sonra sonuca 1 ekleyin. Örneğin devamında –32'nin ikinin bileşeni ikilik gösteriminin nasıl hesaplanacağını gösteriyor.

```
  00100000    +32
  11011111    her biti tersine çevir
+        1    1 ekle
  --------
  11100000    -32, ikinin bileşeni formunda
```

Şüphesiz, ikinin bileşeni dönüşümü negatif sayıları zor çözülür yapar. İyi ki aynı prosedürü bir (ikinin tümlenmişi) negatif sayının pozitif formunu bulmak için verdiğimizde de kullanabiliyoruz. Örneğin: 1101000 değerinin sahip olduğu değeri bulmak için aşağıdaki gibi bir yol izlenir.

```
  00101111    her biti ters çevir
+        1    1 ekle
  --------
  00110000    = 16+32=48
```

Assembler programları ayıları ondalık formda (işaretli ya da işaretsiz) girmenize isin verir ve kendiliğinden bütün dönüşümleri yapar. Bununla birlikte bu dönüşümleri kendiniz yapabilecekseniz bellekte ya da bir saklayıcının içinde saklanmış bir negatif sayının açıklamasını yapmak isteyebilirsiniz.

1.2.HEXADESİMAL (ONDALIK) SAYILAR:

Gerçi, ikilik sayı sitemi, belleğin içine gönderilmiş sayıları göstermek için işe yarar bir yoldur. Yalnız birinin stringi ve sıfırlarla çalışmak çok zordur. Bunlar zaten hataya meyillidir. Çünkü 10110101 gibi bir sayı, kötü tip tanımlamak için aşırı derecede kolaydır.

Yıllar önce, programcılar genel olarak bitlerin gruplarıyla işlem yapmanın tek bitlerle işlem yapmaktan daha iyi olduğunu buldular. En başlangıçtaki mikroişlemciler 4-bit aygıtlardı (bir anda dört bit bilgi işlenirdi). Bu nedenle mantıksal alternatif olan ikilik bitleri dördün grupları olarak numaralayan bir sistemdi.

Bildiğiniz gibi, dört bit ikilik değerleri 0000 ile 1111 arasında (ondalık 0 ile 15 değerleri arasına eşittir), 16 olası olası kombinasyonun bir toplamı olarak gösterir. Eğer sayma sistemi bu 16 kombinasyonda gösterirse, 16 rakama sahip olmalıdır. Bu da bir “taban 16 sistemi” olmalı…

Eğer taban 2 sayılıysa “binary” ve taban 10 sayılıysa “desimal” denir. Peki bir taban 16 sitemi için hangi terim uygundur? Evet herkes taban 16 sistemini bir Yunan sözcüğü olan “hex” (6 için) ile bir Latin sözcüğü olan “decem” (10 için) birleşimiyle oluşan “hexadecimal –

(hexadesimal)” ismiyle isimlendirdi. Bu sayede taban 16 sistemine “hexadesimal” (onaltılı) sayı sistemi dendi.

Hexadesimal sayı sistemindeki 16 rakamın ilk 10’u 0 ile 9 arasındadır. (ondalık 0 ile 9 arasındaki değerler) ve son altısı A ile F arasında (ondalık 10 ile 15 arası değerler) etiketlendi. Tablo 0-2 her hexadesimal rakamın ikilik ve ondalık değerini listeler.

Tablo.1.2. Hexadesimal Sayı Sistemi:

Hexadesimal Rakam	Binary Değeri	Ondalık Değeri	Hexadesimal Rakam	Binary Değeri	Ondalık Değeri
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	A	1010	10
3	0011	3	B	1011	11
4	0100	4	C	1100	12
5	0101	5	D	1101	13
6	0110	6	E	1110	14
7	0111	7	F	1111	15

İkilik rakamlar ve ondalık rakamlar gibi her hexadesimal rakamda kendi tabanının bazı katlarında bir ağırlığa sahiptir. 16 tabanlı hexadesimal sayı sisteminde her rakam ağırlığı bir sağındaki bir sağındaki rakamdan daha büyüktür. Bu 16 defa sürer. En sağdaki rakam bir 16^0’ın ağırlığındadır. Yanındaki 16^1 ağırlığında, ve böyle devam eder.Örneğin: Onaltılı 3AF değeri ondalık 943 değerine sahiptir. Çünkü:

$(3*16^2) + (A*16^1) + (F*16^0)$

ondalık formda

$(3*256) + (10*16) + (15*1) = 943$

1.2.1.Hexadesimal Sayıları Kullanma

BASIC ve diğer yüksek-düzeyli diller genellikle sayıları ondalık formda gösterirken, assembly dili genellikle sayıları hexadesimal formda gösterir. Bu adresleri, talimat kodlarını ve bellek bölgesiyle saklayıcılarının içeriğini kapsar. Böylelikle programınızdan en yüksek verimi almanız için “hexadesimal düşünmeyi deneyin”. Bu ilk başta zordur, ancak bu daha fazla deneyim kazanmanızı kolaylaştırır.

1.3.ASCII / BINARY KOD DÖNÜŞÜMLERİ

Bilgisayar her zaman klavye karakterlerini ASCII'ye çevirir. Girdi bir "sayı" sunarsa, işlemci onu işletmeden önce binary'ye ya da BCD'ye dönüştürmelisiniz.. Aynı şekilde önce bir sayı yazdırabilir ya da bunu ekran üzerinde gösterebilirsiniz. Bunu ASCII forma sokmanız gerekir.

Biz her problemi bu bölümde adresleyeceğiz: ASCII sayıdan binary'ye ve binary sayıdan ASCII'ye nasıl dönüştürülür. (ASCII ve BCD arasındaki dönüşüm benzer prosedür gerektirir. Bu alıştırma okuyucuya bırakılmıştır.)

1.3.1.ASCII Strigleri Binary'ye Dönüştürme:

Tablo, 6-8 0 ile 9 arasındaki rakamlar için ASCII kodlarını gösterir. Görebileceğiniz gibi değer aralığı 30H'dan 39H2a kadardır. Zaten ondalık rakamın binary eşiti basitçe ASCII kodun düşük dört biti olduğu not edilmiştir.

Tablo.1.3. Ondalık Rakamlar İçin ASCII kodları

ASCII Değer (Hex)	Ondalık Rakam(Dec)
30	0
31	1
32	2
33	3
34	4
35	5
36	6
37	7
38	8
39	9

Daha önce söylediğimiz gibi ondalık sayılar 10'un kuvvetleriyle çarpılmış rakamları dizisiyle açıklanmıştır. Örneğin:

237 = (7*1) + (3*10) + (2*100)

ya da

$237 = (7*10^0) + (3*10^1) + (2*10^2)$

Bir anda bir rakam olarak sayıları girmemizden beri, bir ASCII-tabanlı ondalıktan ikiliğe dönüşüm yordamı ondalık ağırlıkları için hesap yapmalı. Bu yordam bir ya da daha çok 10 ile çarpma işlemi içermeli anlamına gelir. Örneğin: operatör tipleri 93 ise, siz 3 eklemeden önce 9 ile 10'u çarpmalısınız. Genelde, dönüşüm süreci bu düzende işler.

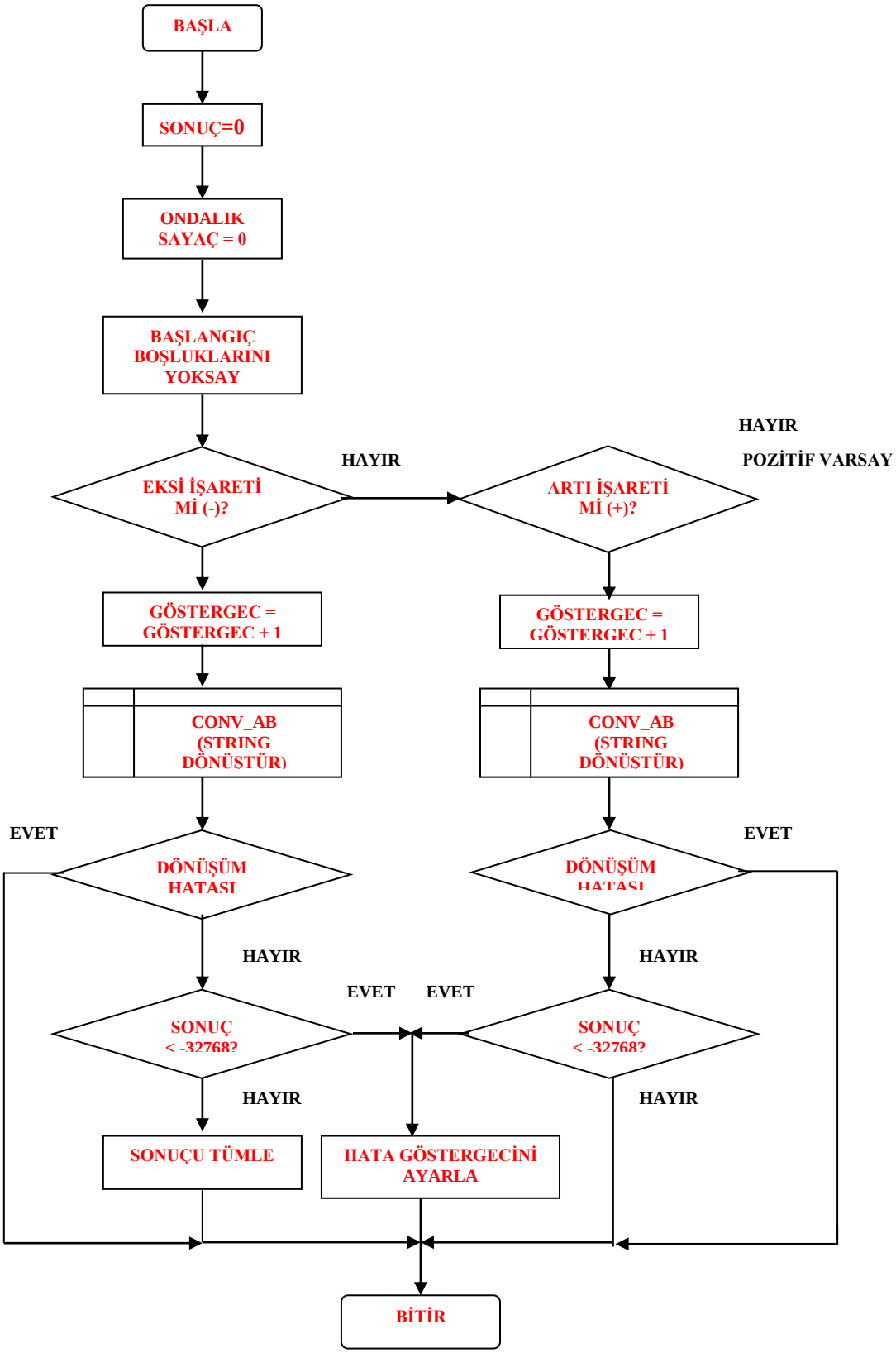
Dönüşüm yordamı ilk (en belirgin) rakamı ASCII kodun dört yüksek-düzende biti olarak parçalara ayırarak binary'ye çevirmelidir. Sonra binary değerini kısmi sonuca kaydeder. Yordam birkaç alt dizi rakamlarını binary'ye dönüştürmelidir., bir önceki kısmi sonucu 10'la çarpmalı, sonra yeni rakamı çarpıma (bu yeni bir kısmi sonuç üretir) eklemelidir.

1.4.ASCII'den Binary'ye Dönüşüm Algoritması

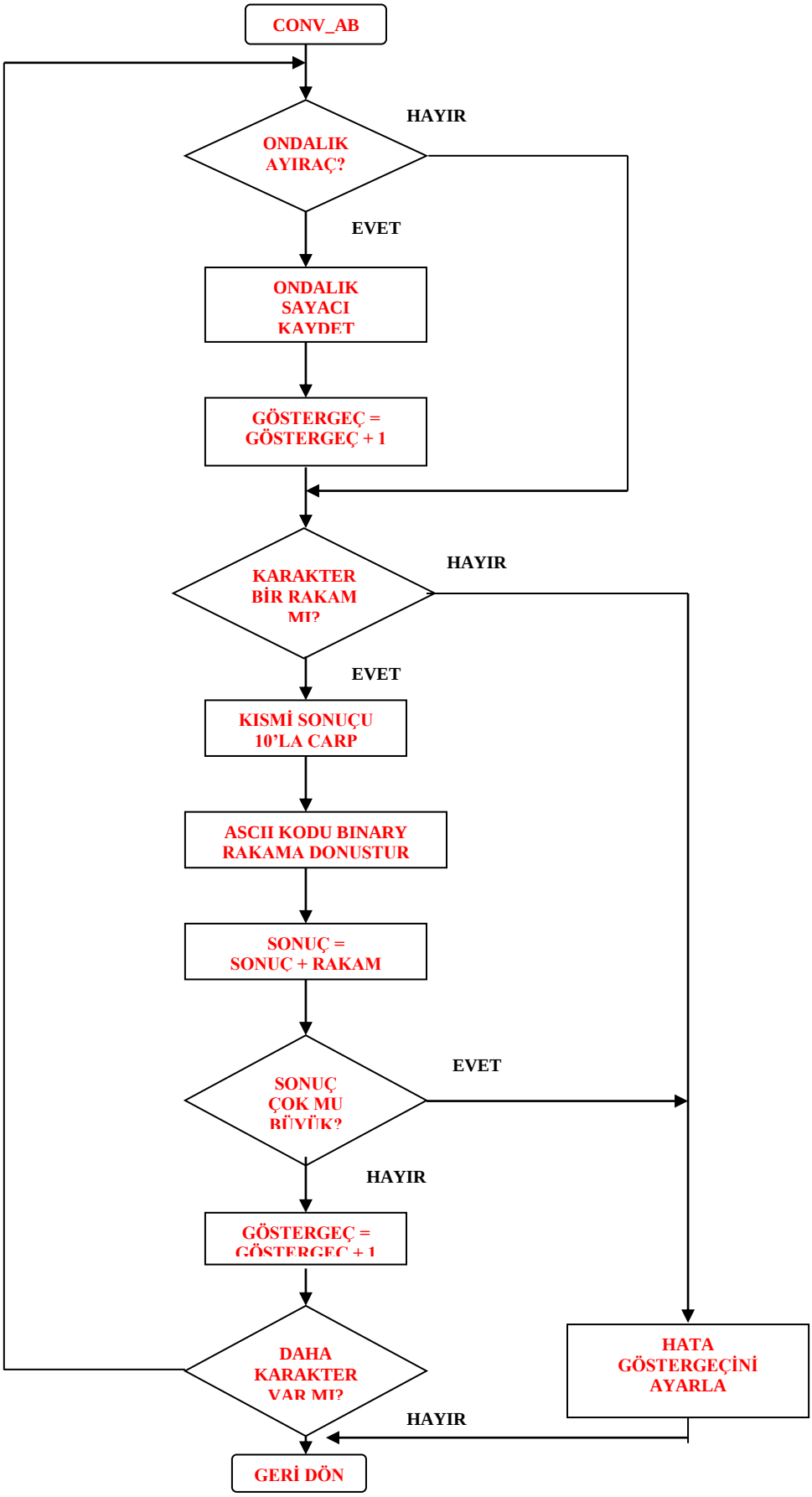
Siz genellikle negatif sayıları pozitif sayılar kadar iyi dönüştürmeye ihtiyaç duyarsınız. Ve sayılar bir ondalık ayıracı içerir, böylece dönüşüm programınız eksi (-) ve nokra (.) karakterlerini de hesaplamalı. Şekil 6-3 bellek içindeki bir ASCII stringi ikinin bileşeni (işaretli) binary sayıya çeviren bir algoritma için akış diyagramıdır. Sayıların 16-bite sığdığını varsayarız, böylece sayıların limiti –32768 ile +32767 dir. Başlamak için, sonuç ve ondalık sayaç (ondalık ayıracının sağındaki rakamların sayısı) sıfırlanır ve program geçen öndeki blokları tarar. Bu noktadan program sayının negatif ya da pozitif olup olmamasına bakarak iki yoldan birini alır. Bu yollar benzerdir. Dönüştürülmüş negatif negatif sayıların hariç tutulması bir pozitif sayı 32,767'ye karşı denetlenmişken negatif sayı –32,768'e karşı denetlenmiştir. Ve tümlenmiş olmalıdır. Şekil 6-3A'da güncel dönüşümü yapan CONV_AB adlı prosedürün akış diyagramı verilmiştir.
CONV_AB prosedürü sonraki string karakterinin ondalık ayıracı olup olmadığını denetleyerek başlar. Eğer ayıraç varsa, CONV_AB kalan karakter sayısını "ondalık sayaç" olarak kaydeder, sonra string göstergesini ilerletir. Eğer sonraki karakter ondalık ayıracı değilse CONV_AB onun ondalık rakam olup olmadığını denetler. Eğer bu karakter bir rakam değilse CONV_AB bunun geçersiz olduğunu bildirir ve bir hata göstergesi ayarlar. Sonra çağrılan programa geri döner.
Geçerli bir rakam karakteri bulma üzerine CONV_AB güncel kısmı sonucu 10'la çarpar. Sonra ASCII karakteri bir rakama çevirir ve sonra bunu sonuca ekler.
Eğer toplama elde üretirse, CONV_AB bir hata göstergeci ayarlar ve döner. Başka şekilde bu ayıracı arttırır ve ondalık ayıraç denetleme talimatlarına geri döner. Tüm string dönüştürüldüğünde, CONV_AB çağrılan programa geri döner.

1.4.1.Ascıı'den Bınary'ye Dönüşüm Programı

Örnek.1.1. bunu gerçekleştiren bir prosedür ve ondan önce algoritmasını gösteriyor. Bu prosedür (ASCII_BIN) data segment'teki (-belki de bir READ_KEYS prosedürüyle girilmiş Örnek 6-4 -) bir ASCII stringi bir 16-bit işaretli sayıya dönüştürür.
ASCII_BIN stringin başlangıç adresini DS:DX'ten, karakter sayacını (en yüksek 7) CX'ten alır. Rastlantısız olarak bunlar READ_KEYS prosedürü parametreleri ASCII_BIN 16-bit değeri AX'in içine döndürür. Rakamların sayısı ondalık ayıracından sonra DX'in içindedir ve DI'nın içindeki ilk dönüştürülemez karakterin adresidir.



Şekil.1.3. Bir ASCII stringi binary'ye çeviren algoritma



Şekil.1.4. Klavyeden bir string okuyan prosedür.

```
 TITLE  READKEYS – 50 tuş darbesini okur

; Bu prosedür 50 tuşu okur
; Girdi:Yok
; Sonuçlar: DS:DX = String adresi
;             CX = Karakter sayacı
;
; Assemble with: MASM READKEYS;
; Link with: LINK callprog+READKEYS;

     PUBLIC  READ_KEYS
DSEG   SEGMENT PARA PUBLIC 'DATA'
USER_STRING  DB  51,51  DUP(?)
DSEG   ENDS

CSEG   SEGMENT PARA PUBLIC 'CODE'
     ASSUME  CS:CSEG,DS:DSEG
READ_KEYS  PROC FAR
        PUSH  AX
        MOV   AX,DSEG   ; DS'yi başlangıç durumuna getir
        MOV   DS,AX
        LEA   DX,USER_STRING  ; String oku
        MOV   AH,0AH
        INT   21H
        SUB   CH,CH     ; CX'ten karakter sayacını oku
        MOV   CL,USER_STRING+1
        ADD   DX,2      ; DX noktasını text yap
        POP   AX        ; AX'i yerine koy
        RET             ; Ana programa geri dön
READ_KEYS  ENDP
CSEG   ENDS
     END
```

Örnek.1.1..ASCII stringi binary'ye dönüştüren prosedür.

```
 TITLE    ASC_BIN – ASCII'den binary'ye dönüştürür

; bir ASCII stringi onun 16-bit, ikinin bileşeni
; binary eşitine dönüştürür.
; Girdiler: DS:DX = Stringin başlangıç adresi
;           CX = Karakter sayacı
; Sonuçlar:   CF = 0 hata olmadığını bildirir
;             AX = ikilik değer
;             DX = Ondalık ayıracından sonra sayaç
;                  rakamları
;             DI = 0FFH
;           CF = 1 hata bildirir
;             DS:DI = dönüştürülemez karakterin adresi
; DX ve CX meyilsiz
;
; Assemble with: MASM ASC_BIN
; Link with: LINK callprog+ASC_BIN

      PUBLIC  ASCII_BIN
.286C
CSEG    SEGMENT PARA 'CODE'
      ASSUME  CS:CSEG
ASCII_BIN  PROC FAR
        PUSH  BX      ; BX ve CX'i kaydet
        PUSH  CX
        MOV   BX,DX   ; BX'in içine ofseti koy
        SUB   AX,AX   ; Başla, sonuç = 0
        SUB   DX,DX   ; ondalık sayaç = 0
        MOV   DI,00FH ; hatalı karakter olmadığını varsay
        CMP   CX,7    ; String çok uzun mu?
        JA    NO_GOOD ; Öyleyse, CF'yi CF yi ayarlamaya
```

```
                    ; git ve çık
BLANKS:   CMP   BYTE_PTR [BX],'' ; Geçmiş önceki boşlukları
                        ; tara
        JNE   CHK_NEG
        INC   BX
        LOOP  BLANKS
CHK_NEG:   CMP   BYTE PTR [BX],'-' ; Negatif sayı mi?
        JNE   CHK_POS
        INC   BX        ; Öyleyse, göstergeci arttır
        DEC   CX        ; sayacı azalt,
        CALL  CONV_AB    ; stringi dönüştür
        JC    THRU
        CMP   AX, 32768  ; sayı çok mu küçük?
        JA    NO_GOOD
        NEG   AX        ; Hayır. Sonucu tamamla
        JS    GOOD
CHK_POS:   CMP   BYTE_PTR [BX],'+'   ; Pozitif sayı mi?
        JNE   GO_CONV
        INC   BX        ; Öyleyse göstergeci arttır
        DEC   CX        ; sayacı azalt
GO_CONV:   CALL  CONV_AB    ; Stringi dönüştür
        JC    THRU
        CMP   AX,32767   ; sayı çok mu büyük?
        JA    NO_GOOD
GOOD:     CLC
        JNC   THRU
NO_GOOD:   STC             ; öyleyse, Elde Bayrağını ayarla
THRU:     POP   CX        ; register'ları yerine koy
        POP   BX
        RET             ; ve çık
ASCII_BIN  ENDP

; Bu prosedür asıl dönüşümü gerçekleştirir.
```

```
CONV_AB    PROC
        PUSH  BP        ; gelişi güzel register'ları
        PUSH  BX        ; kaydeder
        MOV   BP,BX     ; BP'nin içine göstergeci koy
        SUB   BX,BX     ; ve BX'i temizle
CHK_PT:    CMP   DX,0       ; Ondalık ayıracı bulunmuş mu?
        JNZ   RANGE     ; öyleyse, izleyen denetimi atla
        CMP   BYTE PTR DS:[BP],'.'  : Ondalık ayıracı mi?
        JNE   RANGE
        DEC   CX        ; öyleyse, sayacı azalt,
        MOV   DX,CX     ; ve sayacı DX'in içine kaydet
        JZ    END_CONV  ; CX=0'sa çık
        INC   BP        ; göstergeci arttır
RANGE:     CMP   BYTE PTR DS:[BP],'0'  ; Karakter bir rakam
        JB    NON_DIG             ; değilse...
        CMP   BYTE PTR DS:[BP],'9'
        JBE   DIGIT
NON_DIG:   MOV   DI,BP     ; adresini DI'nın içine koy
        STC             ; Elde Bayrağını ayarla
        JC    END_CONV  ; ve çık
NON_DIGIT: IMUL  AX,10     ; AX'in içindeki rakamı 10'la çarp
        MOV   BL,DS:[BP] ; ASCII kodunu çıkar
        AND   BX,0FH    ; yalnızca yüksek bitleri kaydet,
        ADD   AX,BX     ; kısmi Sonucu güncelle
        JC    END_CONV  ; sonuç çok büyükse çık
        INC   BP        ; aksi halde, BP'yi arttır
        LOOP  CHK_PT    ; ve devam et
        CLC             ; bittiğinde, Elde Bayrağını temizle
END_CONV:  POP   BX       ; register'ları yerine koy
        POP   BP
        RET             ; ana programa geri don
CONV_AB    ENDP
CSEG       ENDS
        END
```

DX'in içindeki değer sonucun büyüklüğünü gösterir. Bu size karışık boyutlarda dönüştürülmüş sayıları işletirseniz hangi "ölçek etkeni"ni uygulayacağınızı söyler. DX'in içeriği 0 (sonuç bir tamsayı) ile 5 (sonuç bir basit kesir) arasında olabilir. Örneğin: AX 1000H (ondalık 4096) içerir ve DX 2 içerirse, sonucunuz ondalık 40,96 değerini gösterir.
Bu Sonucu DX'in içinden 3 olarak dönen önceki değer olan sonuca eklemek için, bir önceki Sonucu öncelikle 10'a bölmeniz gerekiyor. Benzer bir şekilde 40.96'yı DX'in içinden 0 olarak dönen bir önceki sonuca eklemek için yeni değeri öncelikle 100'e bölmeniz gerekir.
Elde Bayrağı (Carry Flag - CF) dönüşüm sırasında hata meydana gelip gelmediğini gösterir. Eğer CF 0'sa, sonuçlar geçerli, eğer CF 1'se ASCII_BIN aşağıdaki hatalardan birini bulmuştur.
String yedi karakterden büyükse (CX>7) AX ve DX 0 tutar ve DI 00FFH tutar.
ASCII_BIN geçersiz karakter buldu, DI bunun ofset değerini tutar.
Sayı aralığın dışındaysa (-32768'den daha küçük, 32767'den daha büyük) AX sıfırsızdır ve DI 00FFH'ı tutar.

Yanıtın geçerliliğini denetlemek için ASCII_BIN'i bu yazımda çağırın...

```
        CALL  ASCII_BIN    ; Dönüşüm prosedürünü çağır
        JNC   VALID        ; yanıt geçerli mi?
        OR    DI,DI        ; hayır. Hata durumu bulundu
        JNZ   INV_CHAR
        OR    AX,AX
        JNZ   RANGE_ER
        ..         ; string çok büyüktü
        ..
RANGE_ER:  ..          ; sayı aralık dışında
        ..
INV_CHAR:  ..          ; geçersiz karakter
        ..
VALID:     ..          : yanıt geçerli
        ..
```

1.4.2.Binary Sayıları Stringe Çevirmek

Bir sonuç yazdırmak ya da bunu ekranda göstermek için, onu ASCII'ye çevirmeniz gerekir. İyi ki, bunu yapmak kolay. Bir 16-bit binary sayıyı ASCII'ye dönüştürmek için sayının içerdiği 1, 10, 100 ,1000, 10000'leri bildiren ve bu sayaçların her birini bir ASCII karaktere dönüştüren bir programa ihtiyacınız var. ASCII karakterleri hesaplandığı gibi çıktı verebilir ya da onları bellekte bir string olarak saklayabilir ve başka bir programla daha sonra çıktı olarak verebilirsiniz.
Örnek 6.6 AX içindeki bir 16 bit binary sayıyı bellekte bir ASCII stringe çeviren BIN_ASCII adında bir prosedürdür. Çeşitli sayaçlar türetmek için, BIN_ASCII AX'in içeriğini ardı sıra 10'a böler ve her bölme işleminin kalanını string inşa etmek için kullanır. BIN_ASCII dönüştürülmüş stringlerin adresini DS:DX2in içine ve karakter sayacını CX'in içine geri döndürür.

Örnek.1.2. Binary sayıyı stringe çevirme

```
TITLE    BIN_ASC – Binary'yi ASCII'ye çevirme

; Data segmentteki Işaretli binary sayıyı altı-byte'li ASCII
; stringe (artı işaretli dört rakam) çevirir.
; Girdiler: AX = Dönüştürülmüş sayı
;          DS:DX = string tamponunun başlangıç adresi
; Sonuçlar:  DS:DX = stringin başlangıç adresi
           CX = Karakter sayacı
; Diğer register'lar saklı.

; Assemble with: MASM BIN_ASC;
; Link with: LINK callprog+BIN_ASC;

      PUBLIC BIN_ASCII
CSEG    SEGMENT PARA PUBLIC 'CODE'
      ASSUME  CS:CSEG
BIN_ASCII PROC FAR
      PUSH  DX    ; ana programın register'larini sakla
      PUSH  BX
      PUSH  SI
```

```
        PUSH  AX
        MOV   BX,DX   ; ofseti BX'in içine koy
        MOV   CX,6    ; tamponu boşluklarla doldur
FILL_BUFF:  MOV  BYTE PTR [BX],''
        INC   BX
        LOOP  FILL_BUFF
        MOV   SI,10   ; hazırla -*- 10'a bol
        OR    AX,AX   ; i değeri negatifse,
        JNS   CLR_DVD
        NEG   AX      ; sayıyı pozitif yap
CLR_DVD:  SUB  DX,DX  ; bölümün üst yarısını temizle
        DIV   SI      ; AX'i 10'a böl
        ADD   DX,'0'  ; kalanı ASCII rakamına dönüştür
        DEC   BX      ; buffer yardımıyla yedekle
        MOV   [BX],DL ; karakteri stringin içine sakla
        INC   CX      ; dönüştürülen karakterleri say
        OR    AX,AX   ; hepsi bitti mi?
        JNZ   CLR_DVD ; değilse, yeni rakamı al
        POP   AX      ; evet, asıl değeri al
        OR    AX,AX   ; negatif miydi?
        JNS   NO_MORE
        DEC   BX      ; evet. İşaretini sakla
        MOV   BYTE PTR [BX],'-'
        INC   CX      ; ve karakter sayacını arttır
NO_MORE: POP SI       ; register'ları yerine koy
        POP   BX
        POP   DX
        RET           ; ve çık
BIN_ASCII  ENDP
CSEG    ENDS
        END
```