

6.BÖLÜM

6. VERİ YAPILARI

6.1.Verî Yapılarıyla Çalışma:

Burada hemen hemen bellekteki bilgiyi düzenlemenin yolları düzenlenmiş bilginin çeşidi gibidir. Bu organizasyonel teknikler uygulamalarla birlikte değişir ve bu isimleri alırlar; diziler, stringler, arama tabloları. Bunlar veri yapılarının bütün farklı çeşitleridir.

Veri yapıları konusu birkaç bölüm olabilir. Bu yüzden biz burada bunu geniş olarak vermeyi düşünmüyoruz. Bunun yerine yoğunlaştırılmış olarak temel üç yapıyı vereceğiz: listeler, arama tabloları , metin dosyaları.Listeler element diye adlandırılan verilerin birimlerini bellekte sıralı olarak tutarlar (byte ya da Word). Elemanlar bitişik bellek bölgelerini kapladıysa ya da her element liste içinde bir sonrakine “işaretçi” içerdiğinde bağlıysa dizi ardışık olabilir. Bundan başka, elementler rasgele dizili olabilir ya da artan ya da azalan düzende olabilir.

Arama tabloları bilinen değere ilişkili olarak tanımlanmış bilgiyi (veri ya da adresten herhangi biri) tutan veri yapılarıdır. Telefon dizini bir arama tablosudur.; bir ismi bilerek, ilişkili bir telefon numarasını arayabilirsiniz.Metin dosyaları mektuplarda, raporlarda ve telefon listelerinde olduğu gibi sayısal olmayan bilgiden oluşur.

6.2.Düzenli Olmayan Listeler:

Bizim düzenli toplumumuzda, telefon listeleri alfabetik olarak düzenlenmişken, sokaktaki evlerin numaraları artıyor ya da azalırken, düzenli olmayan “hiçbir şey” her nasılsa can sıkıcı görünür. Hala, herşey bir şekilde düzenli olabilir, bu yüzden düzensiz listeler bazı uygulamalarda hayatın bir gerçeğini hatırlatır.Özellikle rastgele veri ya da zamanla değişen veri içerenler... Örneğin: Bilgisayara uyarlanmış istasyonları saat başı sıcaklığı düzensiz listelerden okuyabilir ve üreticiler statikleri bunlardan alarak günlük tutabilirler.

Bütün listeler bir element sayaç byte’ından (ya da word) ve bir ya da daha fazla veri elementinden oluşmuştur. Siz bir listeyle çalışırken , genellikle elementleri eklemek ya da çıkarmak ya da kesin kesin değerin birini aramak istersiniz. Bu işlemleri yapmak yeterince kolaydır.

1. Bir element eklemek için; bunu listenin sonuna sakla ve element sayacına 1 ekle.
2. Bir element silmek için; bellekte bir elementi yakarı doğru taşı, sonra element sayacından bir çıkar.
3. Bir değer aramak için; listede birinci elemandan başlayarak her elementi arama değeriyle karşılaştır.

6.3.Bir Düzensiz Listeye Bir Element Ekleme: Örnek.6.1’deki prosedür ADD_TO_UL, bir düzensiz liste yaratmak ya da var olan birine bir element eklemek için kullanılan bir çeşit programdır. Bu durumda, word değerlerini içerir (işaretli ya da işaretsiz).

ADD_TO_UL element sayacını CX'ten okur. Veri elementlerini değer için AX'ten tarar. Eğer bu değer listede zaten varsa (ZF'ni son değeri 0'dır), 286 başlangıç adresini DI'nın içinden geri çeker ve element sayacını bir bir arttırır.

Prosedürün çalışması ne kadar zaman alır? Bu elementleri sayısını ve aranan değer listenin içinde olup olmamasına bağlıdır. Birincil etken tarama String (SCASW) talimatını kaç defa yinelemiştir. SCASW (5+8N) kadar döngüde çalışır, N tekrarların sayısıdır. Gelin her durum için zamanlamaları yapalım – değer listenin içinde değildir ve değer listenin içindedir. – N tane data elementi bulunan bir liste için bunlar.

Örnek .6.1: Düzensiz listeye bir element eklemek:

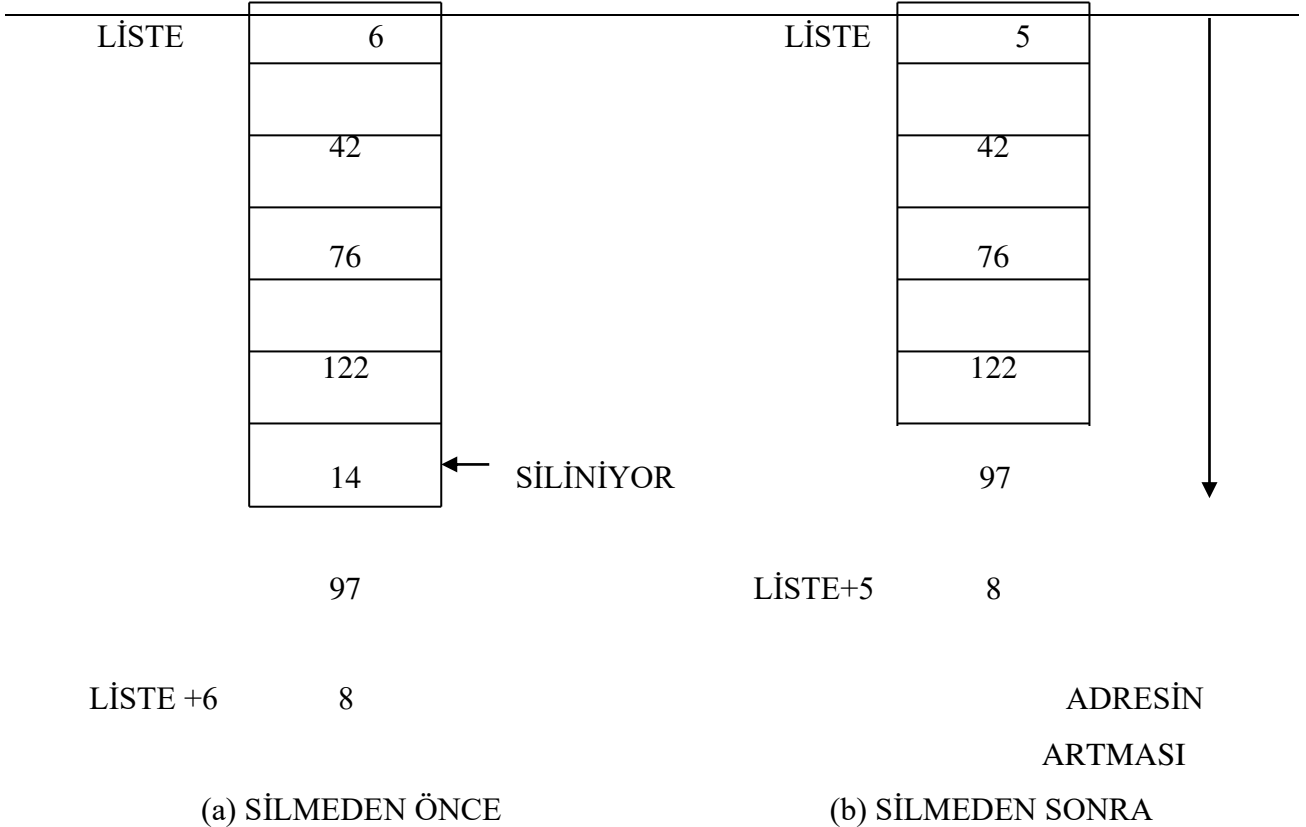
```
TITLE  ADD_2_UL - Düzensiz listeye ekle
; Düzensiz liste içine AX'ten bir değer ekler
; ekstra segment, bu değer listenin içinde değilse.
; Girdiler : DI = Listenin başlangıç adresi
      İlk bölge = Liste uzunluğu (word)
; Sonuçlar : Yok
; DI ve AX değiştirilmeden geri donmuştur.
; Assemble with: MASM ADD_2_UL;
; Link with: LINK callprog+ADD_2_UL;
      PUBLIC  ADD_TO_UL
CSEG   SEGMENT PARA 'CODE'
      ASSUME  CS:CSEG
ADD_TO_UL PROC FAR
      CLD          ; İleri yönde tarama için, DF=0 yap
      PUSH DI      ; Başlangıç adresini kaydet
      PUSH CX
      MOV  CX,ES:[DI] ; Word sayacını gidip getir
      ADD  DI,2      ; DI noktasını birinci veri elementi yap
      REPNE SCASW      ; Değer zaten listenin içinde mi?
      POP CX
      JNE  ADD_IT
      POP DI          ; Evet. Başlangıç adresini yerine koy
      RET             ; ve çık.
ADD_IT: MOV  ES:[DI],AX ; Hayır. Onu listenin sonuna ekle
      POP DI          ; sonra element sayacını güncelle
      INC  WORD PTR ES:[DI]
      RET             ; ve çık.
ADD_TO_UL ENDP
CSEG   ENDS
      END
```

Aranan değer listenin içinde değilse, SCASW talimatı N defa çalışır. Prosedürde kalan diğer talimatlar bir defa çalışır, 44 döngü olur. Böylelikle, Çalışma zamanı = $5 + 8N + 44 = 8N + 49$ döngüdür. Böylece, 100 elementli bir listeye bir element eklemek 849 döngü alır. <<849 döngü ya da 6Mhz’de 141.78μ.>>

Aranan değer listenin içindeyse, 286 yerleştirme yapmak için N/2 karşılaştırmalarının bir ortalamasını alır. Çünkü zamanın yüzde 50’sinde bir aranan değer listenin alt yarısında, ve zamanın diğer yüzde 50’sinde üst yarısındadır. Teorik olarak bu tarama $(5 + 8N)$ kadar döngü alır anlamına gelir. Prosedürde kalan diğer talimatlar ek olarak 29 döngü alır. Böylelikle, ortalama olarak, Çalışma Zamanı = $5 + 4N + 29 = 4N + 34$ döngü olup böylece, 100 elementli bir düzensiz listede bir element aramak ortalama 434 döngü alır, << ya da 6MHZ 72.48μ >>

6.4.Düzensiz Bir Listedden Bir Element Silmek:

Düzensiz bir listeden bir element silmek için, bulmanız ve sonra elementlerin bir yukarısına bir konum taşımanız gerekir. Bunu yaparken silmenin üzerine “hedef” yazarsınız. Element kaldırıldıktan itibaren, element sayacını bir bir azaltın (listenin birinci bölgesi). Göstermek için, Şekil.6.1.a, bellekteki byte’ların listesini gösterir. Listenin 6 elemente sahip olmasından itibaren, ilk bölge (LISTE) 6.değerini tutar. Şekil.6.1.b, listenin dördüncü elemanı (14) silindikten sonra nasıl görüldüğünü gösterir.



Şekil.6.1. Silme eylemi listeye etkisi

Liste bundan sonra 5 data elementine sahiptir ve 97 ve 8 değerleri bellekte bir yukarı taşınmıştır. Silinmiş değer yok edilmiştir. Örnek.6.2'deki DEL_UL prosedürü bu işlemi gerçekleştirir. AX saklayıcısını kullanarak silinecek değeri belirtir. Örnek 5-1'deki gibi DI listenin başlangıcını işaret eder. Talimatlar önce REPNE CX'in içindeki element sayacı ve DI'nın içindeki ilk data elementinin adresini yükler. Sonra listeyi aranan değer için tarar. Bu talimatlar Örnek 5-1'deki olanla özdeşdir. Aranan değer listenin içindeyse (ZF = 1), 286 DELETE kısmına atlar

DELETE kısmında, işlemci iki yoldan birini alır.. Eğer element silinmişse listenin en sonundadır (CX sıfır içerir). 286 listenin element sayacını basitçe azalttığında DEC_CNT kısmına atlar Eğer silme hedefi listenin hiçbir yerinde yoksa, NEXT_EL kısmındaki döngü bütün kalan elementleri bir yukarı konuma taşır, hedefin üzerine yazma. Element sayacı bir azalarak silme eylemini yansıtır.

Örnek.6.2: Düzensiz Listedden bir Element Silmek

```

TITLE    DEL_UL - Düzensiz listeden sil
; Düzensiz liste içindeki AX'ten bir değer siler
; ekstra segment, bu değer listenin içindeyse.
; Girdiler : DI = Listenin başlangıç adresi
        İlk bölge = Liste uzunluğu (word)
; Sonuçlar : Yok
; DI ve AX değiştirilmeden geri dönmüştür.
; Assemble with: MASM DEL_UL;
; Link with: LINK callprog+DEL_UL;

PUBLIC DEL_UL

CSEG    SEGMENT PARA 'CODE'
        ASSUME CS:CSEG
ADEL_UL PROC FAR
        CLD                ; İleri yönde tarama için, DF=0 yap
        PUSH BX            ; BX 'saklayıcısını kaydet
        PUSH DI            ; ve başlangıç adresini
        MOV CX,ES:[DI]     ; Element sayacını gidip getir
        ADD DI,2           ; DI noktasını birinci veri elementi yap
        REPNE SCASW        ; Değer zaten listenin içinde mi?
        JE DELETE         ; Eğer öyleyse, sile git.
        POP DI             ; Aksi halde, saklayıcıları yerine koy

```

```

        POP  BX
        RET          ; ve çık.
; Aşağıdaki talimatlarda listeden bir element silinir,
; bunları izleyerek:
; (1) Element listenin sonunda duruyorsa,
;     element sayacını 1 azaltarak elementi sil.
; (2) Aksi halde, elementi sonradan gelen tüm elementleri,
;     bir konum yukarı kaydırarak sil.
DELETE: JCXZ DEC_CNT    ; Eğer (CX) = 0, en son elementi sil.
NEXT_EL: MOV  BX,ES:[DI] ; Liste içinde bir element yukarı taşı
        MOV  ES:[DI-2],BX
        ADD  DI,2        ; Sonraki elementi işaretle
        LOOP NEXT_EL     ; Son element taşınana kadar don
DEC_CNT: POP  DI         ; Element sayacını 1 azalt
        DEC  WORD PTR ES:[DI]
        POP  BX          ; BX'in içeriğini yerine koy
        RET              ; ve çık.
DEL_UL ENDP
CSEG  ENDS
END

```

6.5.Düzensiz Listenin İçindeki Maksimum ve Minimum Değerler:

Bazen düzensiz listenin içindeki maksimum ve minimum değerleri bulmak isteyebilirsiniz. Kabul edilebilir yaklaşım ilk önce ilk elementi maksimum ve minimum değerlerin ikisi içinde aranan değer olarak bildirmektir. Sonra listedeki kalan elementleri bu değerlerle karşılaştırın. Eğer programınız en küçükken daha küçük bir değer buldursa, bu değeri yeni minimum yapar, Benzer şekilde, program en büyükten daha büyük bir değer bulursa, bu değeri yeni maksimum değer yapar.

Örnek.6.3' teki MINMAX prosedürü bu yöntemi uygulayarak bir düzensiz listenin DI'nın içindeki başlangıç adresindeki işaretli word değerlerini uygular. MINMAX AX ve BX'in içindeki maksimum ve minimum değer sırasıyla geri döner.

Örnek.6.3: Düzensiz listelerin içindeki maksimum ve minimum değerler.

```

TITLE  MINMAX - Düzensiz liste içinde maksimum ve minimum
; Ekstra segmentin içinden bir düzensiz listenin maksimum ve minimum
; Word'lerini bulur.
; Girdiler : ES:DI = Listenin başlangıç adresi

```

```

        İlk bölge = Liste uzunluğu (Word)
; Sonuçlar : AX = Maksimum
;          BX = Minimum
; DI değiştirilmemiştir.
; Assemble with: MASM MINMAX;
; Link with: LINK callprog+MINMAX;
        PUBLIC MINMAX
CSEG    SEGMENT PARA 'CODE'
        ASSUME CS:CSEG
MINMAX  PROC FAR
        PUSH CX
        PUSH DI      ; Başlangıç adresini kaydet
        MOV CX,ES:[DI] ; Element sayacını gidip getir
        DEC CX       ; sayaç-1 karşılaştırmaları için hazır ol
        ADD DI,2      ; İlk elementi işaretle
        MOV AX,BX     ; ve maksimum
CHKMIN: ADD DI,2      ; Sonraki elementi işaretle
        CMP ES:[DI],BX ; Elementi minimumla karşılaştı
        JAE CHKMAX    ; Yeni minimum bulundu mu?
        MOV BX,ES:[DI] ; Evet. BX'in içine koy
        JMP SHORT NEXTEL
CHKMAX: CMP ES:[DI],AX ; Elementi maksimumlar karşılaştı
        JBE NEXTEL    ; Yeni maksimum bulundu mu?
        MOV AX,ES:[DI] ; Evet. AX'in içine koy
NEXTEL: LOOP CHKMIN   ; Bütün listeyi denetle
        POP DI       ; Başlangıç adresini yerine koy
        POP CX
        RET          ; ve çık.
MINMAX  ENDP
CSEG    ENDS
        END.

```

Bu prosedür iki bölüm içerir. İlk bölüm karşılaştırmaların sayılarını hesaplayarak element sayacını -1 yapar ve ilk data elementini x maksimum ve minimum normlarına göre ayarlar. İkinci bölüm ise yeni minimum ve maksimumu arayarak liste boyuca döngüye girer. Bu bölüm BX'in

içine yeni minimumu AX'in içine yeni maksimumu kaydeder. Her ne kadar MINMAX işaretli Word değerlerinin listesini işlese de, siz işaretli Word değerlerinin listeleri içinde maksimum ve minimum aramak için onu değiştirebilirsiniz,; basitçe JAE CHKMAX ile JGE CHKMAX'ı ve JBE NEXTEL ile JLE NEXTEL'i değiştirirsiniz. (.....)

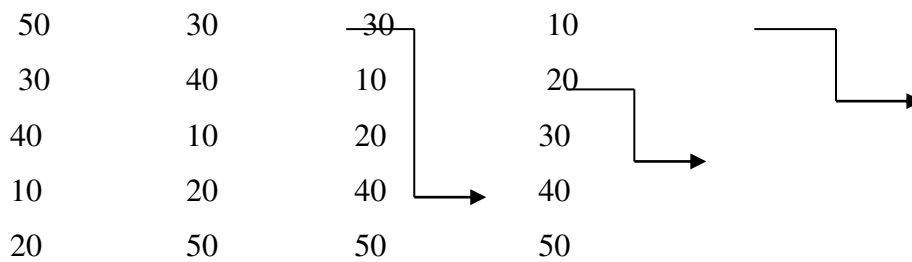
6.6. Düzensiz Verileri Sıralamak:

Eğer bilgiyi zamana karşı çiziyorsanız ya da metin işliyorsanız, bilgiyi düzensiz formada kabul edebilirsiniz. Ancak, bazı uygulamalarda artan ya da azalan düzende düzenlenmiş bilgi daha iyidir. Çünkü bu yolla analiz etmek daha kolaydır. Düzensiz verilerin listesini nasıl yeniden düzenleyebiliriz? Bu konuda literatürün değeri önemlidir. Ama biz bunu genel bir sıralama tekniği olan “bubble sort” ile yoğunlaştırarak vereceğiz.

6.6.1. Bubble Sort (Kabarcık Sıralama):

“Bubble Sort” tekniği adını şundan alır: Liste elementlerini bellek içinde yukarı doğru (daha yüksek numaralanmış adrese) “yükseltir” Tıpkı havaya yükselen çorba kabarcıkları gibi. Bir bubble sort bir listenin yolu boyunca sıralı olarak çalışır. Birinci elementten başlar ve listenin içindeki her elemanı bir sonrakiyle karşılaştırır.

Eğer bubble sort programı en yüksek numaralanmış adresin komşusunda daha büyük bir element bulursa, bu elementleri değiş tokuş eder. Sonra sıradaki iki elementi karşılaştırır. Eğer gerekliyse bunları da değiş tokuş eder ve böyle gider. 286 en son elemana geldiğinde en yüksek değerli element son liste pozisyonuna “yükseltilmiştir <bubbled-up>”



Şekil.6.2. Bir bubble sort en büyük sayıları en sona “yükseltir”.

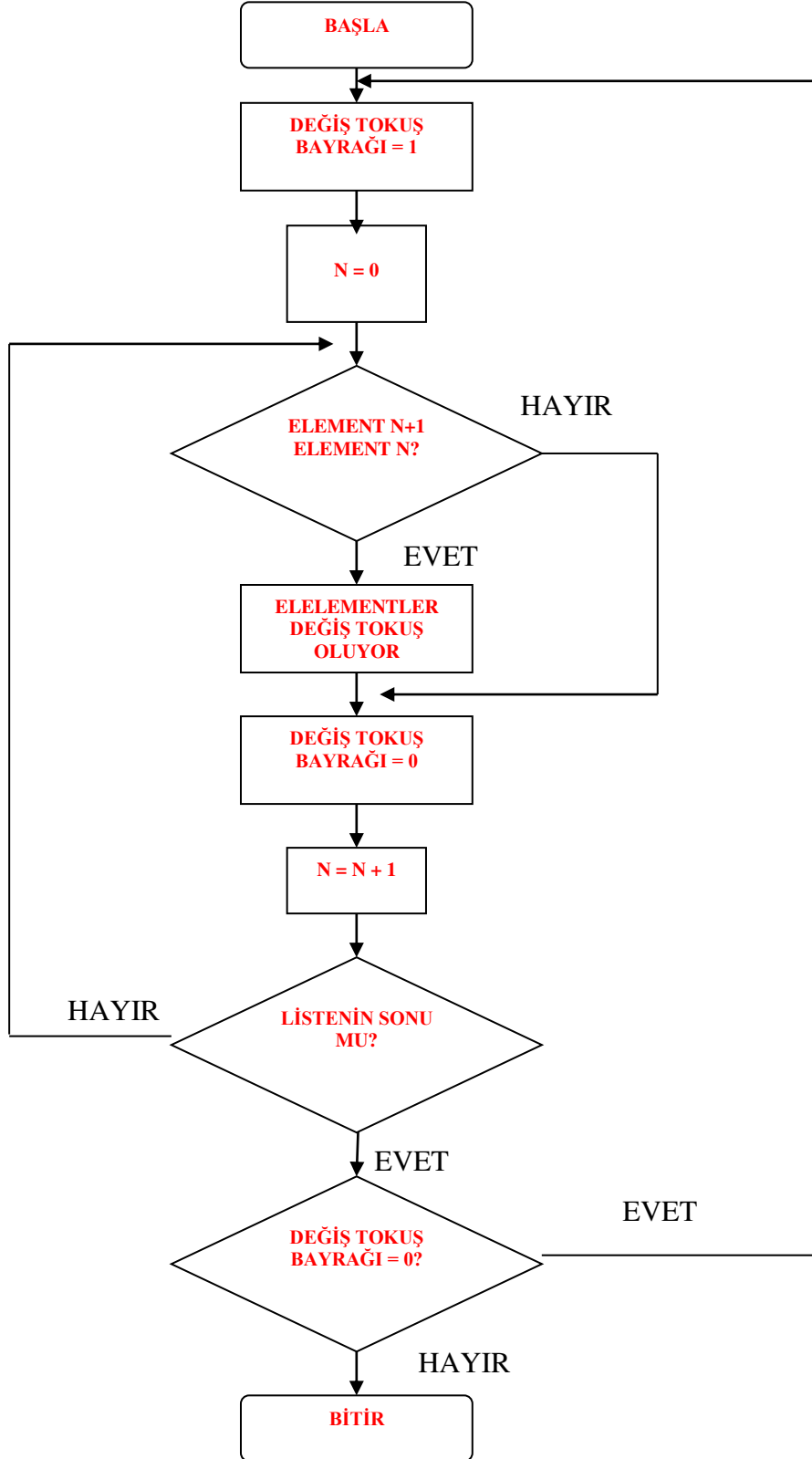
Bu algoritmayla sıralamada, işlemci genellikle şekil <5-2 deki basit örnekte gördüğünüz gibi> listenin içinde birkaç geçiş yapar. Burada ilk geçiş 50'yi listenin sonuna “yükseltir”. Ve sonraki iki geçiş 40 ve 30'u sonraki en yüksek konuma yükseltir. Böylelikle bu dikkate değer liste bu üç geçişle sıralanmış olur.

Listenin “olaylarını” şekil.6.2'deki gibi geçiş-geçiş görerek., bildiğiniz gibi bir liste tamamen sıralandığında bunu sizin için kolayca yapar, ancak bunu bilgisayar nasıl bilir? Siz özel geçiş değeri vermeden ya da bazı diğer yollar durduğunda söylemeden, bilgisayar rahat bir şekilde geçişten sonra geçişi çalıştırarak sona ulaşır. Sıralama geçişlerinin sayısı listenin başlangıçtaki düzenine bağlıdır. Bir program içinde tam geçiş sayacı desteklemenin bir yoluna sahip değiliz. Bir alternatif olarak, biz bilgisayarın sıralama durduğu zaman anlamak için kullanabileceği değiş tokuş bayrağı adında özel bir göstergeç ayarlayacağız.

Değiş tokuş bayrağı her sıralama geçişinden önce 1 olarak ayarlanır. Bir element değiş tokuş işlemi içeren herhangi bir sıralama geçişi bu elementi 0 a dönüştürür. Böylelikle, her geçişten sonra değiş tokuş bayrağının değeri bilgisayarın sıralamaya devam edip etmediğini söyler. Bir listenin içinde başka bir geçiş yapıldığını söyler. Bir 1 ise listenin sıralı olduğunu söyler, ve sıralamayı durdurur. Şekil.6.3, bubble sort algoritmasının akış diyagramını gösterir.

Görebildiğiniz gibi, Eğer ki bir liste zaten başlangıçtaki düzende ise, işlemciye bu gerçeği anlamak için bir geçiş yaptırır. Bir geçiş minimum ise, hangi geçişlerin maksimum sayısı sizi bekletebilir. Şekil.6.2'de 5 sayılı bir liste zaten sıralanmıştır.onu artan düzene sokmak için yalnızca üç sıralama geçişi yaptık. Bir sonraki geçiş listenin gerçekten sıralanmış olduğunu saptamaya gerek duyuyor. Diğer dört geçişte aynı şekilde bu saptamaya ihtiyaç duyuyor.

Eğer başta liste azalan düzende sıralanmış (en kötü durumda) ise, işlemci liste boyunca 5 geçiş yapacaktır. Dört geçiş veriyi sıralamak için, diğeri ise daha başka sıralama gerekmediğini belirlemek içindir.Bu gözlemde, N elementli liste sırlama için ortalama $(N+1)/2$ geçişle birden N'e kadar geçiş yapar.



Şekil.6.3.Bubble Sort Akış Diyagramı

6.6.2.Bubble Sort Program:

Bubble sort teorisinde geri plan önce gelerek ve bir neler yapılması gerektiğini gösteren akış diyagramı bir sıralama programı yazmak için hazırladık. Örnek.6.4, Word değerlerinin listesini sıralayan bir prosedürü (B_SORT) gösterir. (Kolayca byte'ların listesini sıralayacak şekilde değiştirebilirsiniz).

Alışlagelmiş olarak, liste ekstra segmentin içindedir ve başlangıç adresi DI'nın içindedir. B_SORT <BX>'i değiş tokuş bayrağını tutmak için kullanır. (her zaman 1 ya da 0 'dan birini içerir). Ax bir listedeki bir sonraki element ile karşılaştırılmış element değerini tutar.

B_SORT prosedürü data segmentin içinde iki Word değişkeni kullanır: SAVE_CNT karşılaştırma sayacını tutar (element sayacı -1) ve START_ADDR listenin başlangıç adresini tutar. B_SORT bu değişkenleri her yeni sıralama işleminin başlangıcında CX ve DI'yı yeniden başlangıç durumuna getirmek için kullanır.

Her ne kadar B_SORT prosedürü birçok talimat içerse de, oldukça kolaydır. SAVE_CNT ve START_ADDR için değerleri hesapladıktan sonra, prosedür değiş tokuş bayrağını (BX=1), karşılaştırma sayacı (CX), ve element işaretçisi (DI)'yı başlangıç durumuna getirir. NEXT etiketinde 286 elementi AX'in içine yükler, sonra bir bellekteki bir sonraki elementle karşılaştırır.

İkinci element birinci elementten daha küçükse, 286 bunu AX'in içine yükler ve bu iki elementi bellekte <ters düzende> saklar. Çünkü bir değiş tokuş yer almıştır. İşlemci BX'i 0 olarak temizler. CONT kısmındaki LOOP talimatı denetimi bütün liste işlenene kadar NEXT'e aktarır.

Bütün elementler karşılaştırıldığında, CMP talimatı değiş tokuş bayrağının (BX) sıfır olup olmadığını denetler. Eğer sıfırsa, en düşüğünde değiş tokuş sıralama geçişinin öncellenmesi sırasında yer aldı, bu yüzden 286 yeni bir geçiş yapmak için INIT kısmına geri dallandı. Aksi halde, sıralama geçişinden sonra BX hala 1 ise, liste en sonunda sıralanmıştır. Bu yüzden, 286 DI'yı START_ADDR'den ve BX ve AX'i stack'ten geri yükler, sonra döner.

6.6.3.Bubble Sort Programını Düzene Koymak:

Örnek 5-42teki Bubble Sort prosedürü bir zorluğa, ancak önemli bir eksikliğe sahiptir: bazı elementleri gereksiz yere sıralar. Özel olması için, her sıralama geçişi sırasında B_SORT listedeki her element çiftini karşılaştırır. Not: Ancak her geçişte “yükselmeler” listenin içinde bir elementten son konumadır. Bunda, ilk geçiş en yüksek değeri elementi listenin en sonuna yükseltir., ikinci geçiş bir sonraki en yüksek değerli elementi sonraki en son konuma yükseltir. Ve böyle gider.. Böylece, listenin sonundaki elementler boyunca elementlerin en son (sıralanmış) konumlarına ulaşılmış olur. Bunları sonradan gelen sıralama geçişlerinden dışlayabilirsiniz.

Örnek.6.4: Bubble sort prosedürü:

```
TITLE  B_SORT - Bubble Sort
; Ekstra segmentteki listenin 16-bit elementlerini düzenler
; segment artan düzendedir, bubble sort kullanılır.
; Girdiler : ES:DI = Listenin başlangıç adresi
        İlk bölge = Liste uzunluğu (Word)
; DI değiştirilmemiştir.
; Assemble with: MASM B_SORT;
; Link with: LINK callprog+B_SORT;
DSEG   SEGMENT PARA 'DATA'
SAVE_CNT DW  ?
START_ADDR DW  ?
DSEG   ENDS

        PUBLIC B_SORT
CSEG   SEGMENT PARA 'CODE'
        ASSUME CS:CSEG,DS:DESG
B_SORT  PROC FAR
        PUSH DS          ; Çağırın'ın registerlarını kaydet
        PUSH CX
        PUSH AX
        PUSH BX
        MOV  AX,DESG      ; DS'yi başlangıç durumuna getir
        MOV  DS,AX
        MOV  START_ADDR,DI ; Başlangıç adresini kaydet
        MOV  CX,ES:[DI]   ; Element sayacını gidip getir
        DEC  CX           ; sayaç-1 karşılaştırmaları için hazır ol
        MOV  SAVE_CNT,CX  ; Bu değeri belleğe kaydet
INIT:    MOV  BX,1         ; Değiş tokuş bayrağı (BX) = 1
        MOV  CX,SAVE_CNT  ; Bu sayacı CX'e yükle
        MOV  DI,START_ADDR ; Başlangıç adresini DI'ya yükle
NEXT:    ADD  DI,2         ; Adres bir veri elementini adresle
        MOV  AX,ES:[DI]   ; ve bunu AX'e yükle
        CMP  ES:[DI+2],AX ; Sonraki element < bu element?
        JAE  CONT         ; Hayır. Yeni çifti denetlemeye git
```

```

XCHG ES:[DI+2],AX ; Evet. Bu elementleri deęiş tokuř et
MOV ES:[DI],AX
SUB BX,BX ; ve deęiş tokuř bayraęını 0 yap
CONT: LOOP NEXT ; Bütün listeyi isle
CMP BX,0 ; deęiş tokuřlar yapıldı mı?
JE INIT ; Eęer yapıldıysa, listeyi yeniden isle
MOV DI,START_ADDR ; Yapılmadıysa, saklayıcıları yerine koy
POP BX
POP AX
POP CX
POP DS
RET ; ve çık.
B_SORT ENDP
CSEG ENDS
END

```

Sıralanmış elementleri dışlamak için, liste boyunca her geçiř kendi atasında en az bir karşılaştırma içerecektir. Biz bu olayı Kendi prosedürümüzü deęiřtirerek yaptık. Ve böylece SAVE_CNT'nin içindeki deęer her yeni geçiřten önce azalır.

Bunu yapmak için, INIT etiketi altındaki azalma sayacı işlemini getirmek için prosedür içindeki altıncı, yedinci ve sekizinci talimatları basitçe deęiřtirmek gerekir. Siz aynı zamanda azalmadan sonra SAVE_CNT 0 ise işlemciyi çıkar yapmak için bir başka talimat eklemelisiniz. Ařaęıda bu deęiřiklięin özeti vardır.

<u>Aslı</u>	<u>Düzenlenmiş</u> ***
DEC CX	MOV SAVE_CNT,CX
MOV SAVE_CNT,CX	INIT: MOV BX,1
INIT: MOV BX,1	DEC SAVE_CNT
JZ SORTED	

Burada, SORTED MOV talimatı üzerinde START_ADDR'den DI'nın içerięini geri yükleyen etikettir. Örnek 5-5 bu deęiřiklięe sahip yeni bir bubble sort prosedürünü (BUBBLE) gösterir.

Herhangi bir verilen liste için, BUBBLE Örnek.6.4'teki B_SORT prosedüründeki gibi sıralama geçiřlerinin aynı sayılarını işler. Ancak çünkü BUBBLE karşılařtırmaları yaklaşık yarısı kadarını yapar, bu B_SORT'tan daha hızlı çalışır.

Örneğin: 100 elementi azalan düzende düzenlenmiş olarak sırlamak için;B_SORT 100 geçiş yapar, ve her geçiş için 99 karşılaştırma – toplam 9,900 karşılaştırma yapar. Tam aksine, BUBBLE 50 karşılaştırmanın ortalaması için 99 karşılaştırmayla ilk geçişi yapar ve bir de son geçişi yapar, - toplam 5,500 karşılaştırma.

B_SORT ile BUBBLE’ı karşılaştırmak için, IBM PC AT’de bu iki prosedürü de kullanarak 16-bit elementlerin iki listesini sıraladım. Her iki liste de başlangıçta azalan düzende düzenlendi. 500 elementli ilk liste, B_SORT ile 2,5 saniyede; BUBBLE ile 1,5 saniyede sıralandı. 1000 elementli ikinci listenin sırlanması B_SORT ile 10,0 saniye BUBBLE ile 5,0 saniye aldı. Bu testlere dayanarak, BUBBLE bir listeyi B_SORT’tan yüzde 40-50 daha hızlı sıralıyor.

Not: Sıralama zamanı uzun listeleri ele aldığımızda üzücü bir şekilde artıyor. Bizim bubble sort prosedürümüz 32K Word’leri sırlayabiliyor, ‘‘ancak listeniz bu uzunluğun herhangi bir yerindeyse, daha iyisini hazırlamak için bekleyeceksiniz.’’ ‘’’ éé ‘’’ Aslında, en kötü durumda 2000 çift kelimenin listesinin sırlanması BUBBLE ile 20 saniye alır. Bu 1000 kelimelik bir listenin sırlanmasının dört katıdır.

Örnek.6.5: Daha iyi bir Bubble Sort Prosedürü:

```
TITLE    B_SORT - Daha iyi Bubble Sort
; Ekstra segmentteki listenin 16-bit elementlerini düzenler
; segment artan düzendedir, bubble sort kullanılır.
; Girdiler : ES:DI = Listenin başlangıç adresi
          İlk bölge = Liste uzunluğu (Word)
; DI değiştirilmemiştir.
; Assemble with: MASM BUBBLE;
; Link with: LINK callprog+BUBBLE;
DSEG     SEGMENT PARA 'DATA'
SAVE_CNT DW    ?
START_ADDR DW  ?
DSEG     ENDS

PUBLIC BUBBLE

CSEG     SEGMENT PARA 'CODE'
          ASSUME CS:CSEG,DS:DESG
BUBBLE   PROC FAR
          PUSH DS          ; Çağıranın registerlarını kaydet
          PUSH CX
          PUSH AX
```

```

PUSH BX
MOV AX,DESG      ; DS'yi başlangıç durumuna getir
MOV DS,AX
MOV START_ADDR,DI
MOV CX,ES:[DI]   ; Element sayacını gidip getir
MOV SAVE_CNT,CX  ; Bu değeri belleğe kaydet
INIT:  MOV BX,1    ; Değiş tokuş bayrağı (BX) = 1
      MOV CX,SAVE_CNT ; Bu sayacı CX'e yükle
      MOV DI,START_ADDR ; Başlangıç adresini DI'ya yükle
NEXT:  ADD DI,2    ; Adres bir veri elementini adresle
      MOV AX,ES:[DI] ; ve bunu AX'e yükle
      CMP ES:[DI+2],AX ; Sonraki element < bu element?
      JAE CONT     ; Hayır. Yeni çifti denetlemeye git
      XCHG ES:[DI+2],AX ; Evet. Bu elementleri Değiş tokuş et
      MOV ES:[DI],AX
      SUB BX,BX    ; ve Değiş tokuş bayrağını 0 yap
CONT:  LOOP NEXT   ; Bütün listeyi isle
      CMP BX,0    ; Değiş tokuşlar yapıldı mı?
      JE INIT     ; Eğer yapıldıysa, listeyi yeniden isle
SORTED: MOV DI,START_ADDR ; Yapılmadıysa, saklayıcıları yerine koy
      POP BX
      POP AX
      POP CX
      POP DS
      RET        ; ve çık.
B_SORT ENDP
CSEG  ENDS
      END

```

6.7.Düzenli Listeler:

Şimdi bir listenin nasıl sıralanacağı biliyorsunuz. Gelin özel bir değer nasıl aranacağını tartışalım ve elementleri ekleyeceğimizi ve sileceğimizi görelim.

6.7.1.Düzenli Bir Listede Arama Yapma:

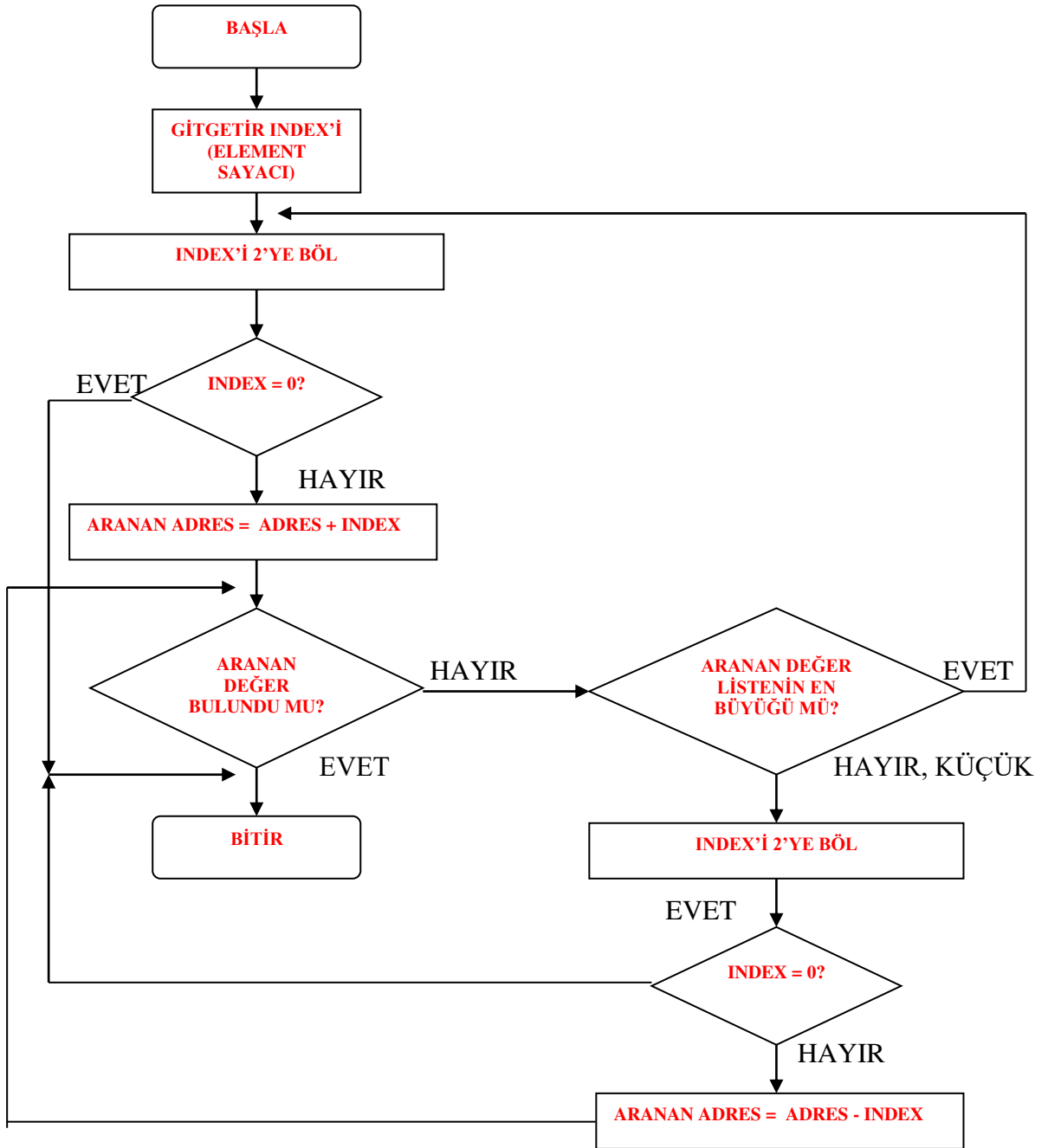
Düzensiz listeleri anlatırken düzensiz listenin gerektirdiği arama boyunca elementleri element element nasıl yerleştirildiğini öğrendik. Bu N-elementli bir liste için ortalama $N/2$ karşılaştırma alıyordu. Bir liste düzenliyse, ancak, siz bir değeri bulmak için birkaç arama tekniklerinin herhangi birini kullanabilirsiniz. Hepsi için ancak en kısa listeler için, bütün bu teknikle sıralı aramadan daha hızlı ve daha verimlidir.

6.7.2.Binary(İkili) Arama:

Düzenli listelerlerde arama için bir yaygın teknik binary arama (binary search) diye adlandırılır. Bu isim şu gerçeği yansıtır: bu arama listeyi sürekli olarak ikiye bölündüğünü ve sonunda bir bir elementin içinde saklandığını yansıtır. Binary arama listenin ortasından başlar ve aranan değerin bu noktanın üstünde ya da altında olup olmadığını bildirir. Bundan sonra listenin yarısını alır ve bunu ikiye böler, ve böyle sürüp gider.

Şekil.6.4' deki akış diyagramı düzenli liste üzerinde sonuca benzer bir adres üretmek nasıl binary arama yapacağımızı gösterir. Aranan değer listenin içindeyse, adres bu eşleme elementinindir. Eğer aranan değer listenin içinde değilse, adres karşılaştırılanın en son yeridir. Kuşkusuz, bu aramayı gerçekleştiren program aynı zamanda adresin başarılı ya da başarısız aramayı yansıtıp yansıtmadığını söyleyen birkaç çeşit göstergesi de döndürür.

Örnek.6.6, işaretli Word'lerin düzenli bir listesinde arama yapmakta kullanabileceğiniz bir prosedürü (B_SEARCH) gösterir. gerçekte bu prosedürle Wordlerin yerine byteleri gerektirmesi istenirse tamal algoritmamızda birkaç değişiklik yapmamız gerekir. Bir şey için, çünkü Word'ler bellekte iki byte'lık yer ayırır. biz her zaman çift değerler olan ikinin çarpanların indekslemeyi garanti eden talimatlar kullanmalıyız. Aynı nedenle, indeks 2'ye azaldığında (0 yerine) aramayı sonlandırırız ve başarısız olarak tanımlarız.



Şekil.6.4. Binary Arama Algoritması

Örnek.6.6: 16-bit binary arama prosedürü:

```
TITLE      B_SEARCH - Binary Arama Prosedürü
; ekstra segmentte bir düzenli listede AX'in içindeki
; Word değeri için arama yapar.
; Girdiler : ES:DI = Listenin Başlangıç adresi
           İlk bölge = Listenin uzunluğu (Word)
; Sonuçlar: Değer listenin içindeyse,
           CF = 0
           SI = Eşleme elementinin ofseti
Değer listenin içinde değilse,
           CF = 1
           SI = Son karşılaştırılan elementin ofseti
; AX ve DI etkilenmemiştir.
; Assemble with: MASM B_SEARCH;
; Link with: LINK callprog+B_SEARCH;
DSEG  SEGMENT PARA 'DATA'
START_ADDR DW ?
DSEG  ENDS

PUBLIC B_SEARCH
CSEG  SEGMENT PARA 'CODE'
      ASSUME CS:CSEG,DS:DSEG
B_SEARCH PROC FAR
      PUSH  DS      ; Çağıranın DŞsaklayıcılarını kaydet
      PUSH  AX
      MOV   AX,DESG  ; DS'yi Başlangıç durumuna getir
      MOV   DS,AX
      POP   AX
; Eğer AX listenin sınırları ötesinde bulunduyorsa.
      CMP  AX,ES:[DI+2] ; Aranılan Değer < or = ilk element?
      JA   CHK_LAST    ; Hayır. En son elementini denetlemeye git
      LEA  SI,ES:[DI+2] ; Evet. İlk elementin adresini gidip getir.
      JE   EXIT        ; Eğer Değer = ilk element ise, çık
      STC                     ; Eğer Değer < ilk element ise, CF'yi ayarla
      JMP  EXIT        ; ve sonra çık
```

```

CHK_LAST: MOV SI,ES:[DI]    ; En son elementi işaretle
          SHL SI,1
          ADD SI,DI
          CMP AX,ES:[SI]    ; Aranan Değer en son elementten büyük mu?
          JB SEARCH        ; Hayır. Arama listesine git
          JE EXIT          ; Evet. Değer en son elemente eşitse çık
          STC              ; Değer > en son element, set CF
          JMP EXIT          ; and then exit
; Listenin içindeki değeri arama
SEARCH: MOV START_ADDR,DI  ; Başlangıç adresini belleğe kaydet
          MOV SI,ES:[DI]    ; Indeks'i hafızadan oku
EVEN_IDX: TEST SI,1        ; Indeks'i bir çift değere it
          JZ ADD_IDX
          INC SI
ADD_IDX: ADD DI,SI          ; Bir sonraki arama adresini hesapla
COMPARE: CMP AX,ES:[DI]    ; Aranan Değer bulundu mu?
          JE ALL_DONE      ; bulunduysa, çık
          JA HIGHER        ; Aksi halde, doğru yarıyı bul
; Bu talimatlar listenin içinde aranan Değer daha küçükse çalışır
          CMP SI,2          ; Index= 2?
          JNE IDX_OK
NO_MATCH: STC              ; Öyleyse, CF'yi ayarla
          JE ALL_DONE      ; ve çık
IDX_OK: SHR SI,1           ; değilse, index'i ikiye bol
          TEST SI,1         ; Indeks'i bir çift değere it
          JZ SUB_IDX
          INC SI
SUB_IDX: SUB DI,SI         ; Bir sonraki adresi hesapla
          JMP SHORT COMPARE ; Bu elementi denetlemeye git
; Bu talimatlar listenin içinde aranan Değer daha büyükse çalışır
HIGHER: CMP SI,2          ; Indeks = 2?
          JE NO_MATCH      ; Öyleyse, CF'yi ayarla ve çık
          SHR SI,1          ; Değilse, index'i ikiye bol
          JMP SHORT EVEN_IDX ; Bir sonraki elementi denetlemeye git

```

```

; Aşağıdaki talimatlar çıkış talimatlarıdır
ALL_DONE: MOV SI,DI      ; Karşılaştırma adresini SI'nin içine taşı
          MOV DI,START_ADDR ; Başlangıç adresini yerine koy
EXIT: POP DS
      RET              ; ve çık
B_SEARCH ENDP
CSEG ENDS
END

```

Bu prosedürde, aranan değer AX'in içinde ve listenin başlangıç adresi DI'nın içindedir. B_SEARCH sonuç adresini SI'nın içine geri döndürür ve CF değerini bulunup (CF=0) bulunmadığı (CF=1) söyler.

B_SEARCH prosedürü temel binary arama algoritmasının göstermediği bir adımla başlar. (Şekil 5-4). Bu aranan değeri listenin ilk ve son elementleriyle karşılaştırır. Aranan değer ilk elementten daha küçükse ya da son elementten daha büyükse ya da bu elementlerden birine eşitse; prosedür daha fazla gürültü patırtı yapmadan sonlanır. Eğer bu başlangıç denetimleri başarısız olursa ancak, 286 arama işlemini SEARCH'ten başlatarak ilerletir.

DI'yı belleğe kaydettikten sonra, 286 indeksi (Word sayacı) SI'nın içindeki listenin ilk yerinden kopyalar ve onu çift değere iter. Bu indeks listenin ortasındaki elementin adresinden DI'ya eklenmiştir. Başlangıç değeri binary arama içindir. Sonra 286 aranan değeri ortadaki element ile karşılaştırır. Ve değer eşitse ALL_DONE kısmına dallanır. Eşitliğin olmaması durumunda, 286 listenin üst yarısında (HIGHER kısmında başlayan talimatları kullanarak) ya da alt yarısında aramanın sürüp sürmediğini belirler.

Bu yollar aşağıdakileri yaptığınızda benzerdir.

1. Indexin 2'ye eşit olup olmadığını denetle. Eğer öyleyse, 286 CF'yi 1'e ayarlar (Bir eşitsizlik gösterir), sonra ALL_DONE'a aktarır ve çıkar.
2. Index'i bir konum sağa kaydırarak ikiye böl.
3. Kaydırılmış index'i çift değere doğru it.

Ancak, listenin düşüğü aramak için, 286 index'i (SI) güncel adresten (DI) çıkarır; yüksek bulmak için index'i güncel adrese ekler.

Bu işlem indeks 2'ye kadar azalana ya da aranana değer bulunana kadar yinelenir. Başka yolla, DI SI'ya taşındığında ve DI'nın asıl içeriği bellekten geri alındığında prosedür ALL_DONE'da biter.

Binary arama düz sıralı element element aramadan ne kadar verimli – Örnek.6.1 ve 6.2 ‘deki çeşidiyle? Kuşkusuz, siz doğal olarak bir listeyi düzende bir değer için elementi eşleyerek bir şeyler yapmak için ararsınız. Tipik olarak, listeye bir değer eklemek ya da listeden bir değer çıkarmak istersiniz. Şimdi bu işlemleri nasıl gerçekleştirildiğini görelim.

6.7.2.1.Düzenli Bir Listeye Bir Element Ekleme:

Düzenli bir listeye bu dört adımla bir element ekleyebilirsiniz.

1. Değerin nereye eklenmiş olduğunu bul.
2. Tüm daha yüksek değerli elementleri bir konum aşağı taşıyarak giriş için bölgeyi temizleyin.
3. Yeni boşaltılmış element konumuna girişi ekleyin.
4. Eklemeyi yansıtmak için, element sayacına 1 ekleyin.

B_SEARCH prosedürü önceden geliştirilip bize verilmiş iyi bir ipucu olan girişin nereye yapılabileceğini içerir: arama bittiğinde elementin adresi geri döner. Adım 1’i tamamlamak için, son aranan elementin hemen önce ya da hemen sonra girilip girilmediğini belirlememiz gerekir. Bunu giriş değerini son aranan element ile karşılaştırarak belirleriz.

Örnek.6.7, bizim listelediğimiz dört adımı gerçekleştiren (ADD_TO_OL) prosedürünü gösterir. Bu B_SEARCH’ü çağırarak giriş değerinin zaten listenin içinde olup olmadığını bulmak için başlar. B_SEARCH’ı yeniden çağırma SI’nın içine bir adres ve Elde Bayrağına’da (CF) bulundu/bulunamadı göstergeci döndürür.

B_SEARCH’tan dönen, ADD_TO_OL prosedürü CF’yi sorgular, ve CF 0 ise çıkar (bu girişin zaten listenin içinde olduğu anlamına gelir). CF 1 ise ancak, prosedür en son aranan adresi BX’in içine kaydeder ve en son elementin adresini (CX’in içindeki) hesaplar. SI’nın içeriğinin bu adresten çıkarılması ekleme için bir oda yapmak için bellek içinde daha yükseğe taşınmış olması gereken byte’ların sayısını verir. Bu sonucu 2’ye bölme (sağa kaydırarak) size ne kadar Word’ün taşındığını söyler. Eğer giriş değeri en son karşılaştırılan elementten küçükse, bu element zaten taşınmış olmalıdır, bu yüzden taşıma sayacını (CX) 1 arttırırız.

Örnek.6.7: Düzenli Bir Listeye Bir Element Ekleme:

TITLE ADD_2_OL - Düzenli listeye girdi ekler

; Düzenli liste içine AX’ten bir değer ekler

; ekstra segment, bu değer listenin içinde Değilse.

; Girdiler : DI = Listenin Başlangıç adresi

 İlk bölge = Liste uzunluğu (Word)

; Sonuçlar : Yok

; DI ve AX değiştirilmemiştir

```

; B_SEARCH prosedürü (Örnek 5-6) aramayı sağlamak için kullanıldı
; Assemble with: MASM ADD_2_OL;
; Link with: LINK callprog+ADD_2_OL+B_SEARCH;
    EXTRN B_SEARCH:FAR
    PUBLIC ADD_TO_OL
CSEG    SEGMENT PARA 'CODE'
    ASSUME CS:CSEG
ADD_TO_OL PROC FAR
    PUSH CX        ; saklayıcıları Kaydet
    PUSH SI
    PUSH BX
    CALL B_SEARCH   ; değer lisenin içinde mi?
    JNC GOODBYE     ; Öyleyse çık
    MOV BX,SI       ; Değilse, Karsılaştırma adresini BX'e kopyala
    MOV CX,ES:[DI]  ; En son elementin adresini bul
    SHL CX,1
    ADD CX,DI       ; ve bunu CX'in içine koy
    PUSH CX        ; Bu adresi stack'e kaydet
    SUB CX,SI       ; Tasınmış Word'lerin sayısını hesapla
    SHR CX,1
    CMP AX,ES:[SI]  ; Tasınmış elementler karşılaştırılacak mı?
    JA EXCLUDE
    INC CX          ; Evet. Taşıma sayacı 1 arttır
    JNZ CHECK_CNT
EXCLUDE: ADD BX,2    ; Hayır. Girdi göstergesini ayarla
CHECK_CNT: CMP CX,0 ; Taşıma sayacı =0?
    JNE MOVE_ELS
    POP SI          ; Öyleyse, değeri listenin sonuna sakla
    MOV ES:[SI+2],AX
    JMP SHORT INC_CNT ; sonra element sayacı arttırmaya git
MOVE_ELS: POP SI    ; Taşıma için Başlangıç adresini yükle
    PUSH BX         ; Girdi adresini stack'e kaydet
MOVE_ONE: MOV BX,ES[SI] ; Listede bir element aşağı taşı
    MOV ES:[SI+2],BX

```

```

SUB SI,2      ; Sonraki elementi işaretle
LOOP MOVE_ONE ; Hepsi taşınana kadar yinele
POP BX       ; Girdi adresini geri al
MOV ES:[BX],AX ; Listede AX'i gir
INC_CNT: INC WORD PTR ES:[DI] ; Element sayacına 1 ekle
GOODBYE: POP BX      ; saklayıcıları yerine koy
POP SI
POP CX
RET          ; ve çık.
ADD_TO_OL ENDP
CSEG ENDS
END

```

CHECK_CNT’de, 286 taşıma sayacını denetler. Eğer sıfırsa, girişi basitçe listenin sonuna çivilenmiştir. Aksi halde, bu değer listenin içine eklenmiş olmalıdır; bu tüm sonradan gelen elementleri Word konumunun aşağısına taşınmasını gerektirir.

Bu talimatlar MOVE_ELS’de başlar ve elementleri bir bir en son listenin içindeki Word’dan başlayarak aşağı taşımaya gerekli elementler taşınmış olduğunda, 286 girişi (AX) yeni boşaltılmış yuvaya ekler, sonra element sayacını bir artırır.

6.7.2.2.Düzenli Bir Listedden Bir Element Silmek:

Düzenli bir listeden bir element silmek eklemekten daha kolaydır. Bütün bu yapmak zorunda olduklarımız uygun elementi bulmak, tüm sıradaki elementleri bir konum yukarı taşımak ve element sayacını azaltmak için.

Örnek.6.8, B_SEARCH (Örnek.6.6) kullanarak silme eğilimindeki “hedefi” yerleştiren tipik bir silme prosedürü (DEL_OL)’u gösterir. Alışlagelmiş olarak, listenin başlangıç adresi DI’nın içindedir ve silinmiş olan değer AX’in içindedir.

B_SEARCH listenin içine giriş değerini yerleştirir. DEL_OL bunun adresini kullanır, ve listenin sonunun adresi ne kadar Word yukarı taşındığını belirlemek içindir. MOVEM’deki dört-talimatlı döngü taşıma işlemini gerçekleştirir. 286 bütün Word’leri taşıdığında, listenin element sayacını silme eylemini yansıtmak için azaltır.

Örnek .6.8: Düzenli Bir Listedden Bir Element Silme:

```
TITLE  DEL_OL - Düzenli listeden element siler
; Ekstra segmentteki düzenli liste içinden AX'e bir değer siler
; bu değer listenin içinde ise .
; Girdiler : DI = Listenin Başlangıç adresi
        İlk bölge = Liste uzunluğu (Word)
; Sonuçlar : Yok
; DI ve AX değiştirilmemiştir
; B_SEARCH prosedürü (Örnek 5-6) aramayı sağlamak için kullanıldı
; Assemble with: MASM DEL_OL;
; Link with: LINK callprog+DEL_OL+B_SEARCH;

        EXTRN B_SEARCH:FAR

        PUBLIC DEL_OL

CSEG    SEGMENT PARA 'CODE'
ADD_TO_OL PROC FAR
        ASSUME CS:CSEG
        PUSH CX      ; saklayıcıları Kaydet
        PUSH SI
        PUSH BX
        CALL B_SEARCH ; değer listenin içinde mi?
        JNC ADIOS     ; Öyleyse çık
        MOV CX,ES:[DI] ; Öyleyse en son elementin adresini bul
        SHL CX,1
        ADD CX,DI      ; ve bunu CX'in içine koy
        CMP CX,SI      ; En son element silinmiş mi?
        JE  CNT_M1     ; Evet. Element sayacını azaltmaya git
        SUB CX,SI      ; Hayır. Taşınma sayacını hesapla
        SHR CX,1
MOVEM:  MOV BX,ES:[SI+2] ; Listede bir element yukarı taşı
        MOV ES:[SI],BX
        ADD SI,2        ; Sonraki elementi işaretle
        LOOP MOVEM      ; Hepsi taşınana kadar yinele
CNT_M1: DEC WORD PTR ES:[DI] ; Element sayacını 1 azalt
ADIOS:  POP BX         ; saklayıcıları yerine koy
```

```
POP SI
POP CX
RET          ; ve çık.
ADEL_OL ENDP
CSEG  ENDS
END
```

6.9.Arama Tabloları:

Bazı programlar işlemeye gerek duydukları değerleri tutmak için tabloları kullanırlar. Bazen bu tablolar hesaplaması çok zaman alan sayıları tutar, <sinüs ve açılar gibi>. Diğer zamanlarda, bu tablolar program girdisiyle ilişkili olarak tanımlanmış bazı parametreleri tutar, ancak hesaplamadan tutar. Bu durum için, bilgisayarın isme karşılık gelen telefon numarasını hesaplamasını bekleyemezsiniz.

Bunun gibi uygulamalar “arama tabloları” diye adlandırılır. bu ismin ima ettiği gibi, bir arama tablosu bilinen bir değere dayanan (bir işlev) bilginin (bir parça) bir ögesini elde eder.

Arama tabloları karmaşık ya da zaman tüketen dönüşüm işlemlerini kaldırabilir. Sayının kare kökünü ya da küp kökünü alması ya da açının (sinüs, kosinüs gibi) türetmek gibi işlevler olabilir. arama tabloları genellikle argümanların küçük aralık kapsadığı bir işlev olduğunda verimlidir. (örneğin: sayıları 1 ile 20 arasında olan küpler). Bir arama tablosu kullanarak, bir değere gereksinim duyduğu her zaman karmaşık hesaplamalar gerçekleştirmeye gerek duymaz.

Genelde, arama tabloları hepsinde zaman kazandırır ancak en basit durumlarda, (Örneğin, her zaman işlevin değerini iki kez argümanları saklayan bir arama tablosu kullanamayacaksınız). Ancak arama tabloları boyunca genellikle belleğin saklama boşluğunun değerlerini büyük alır. Bunlar uygulamalarda bellek boşluğunu yürütme hızı için kurban etmek istediğinizde en verimlidir.

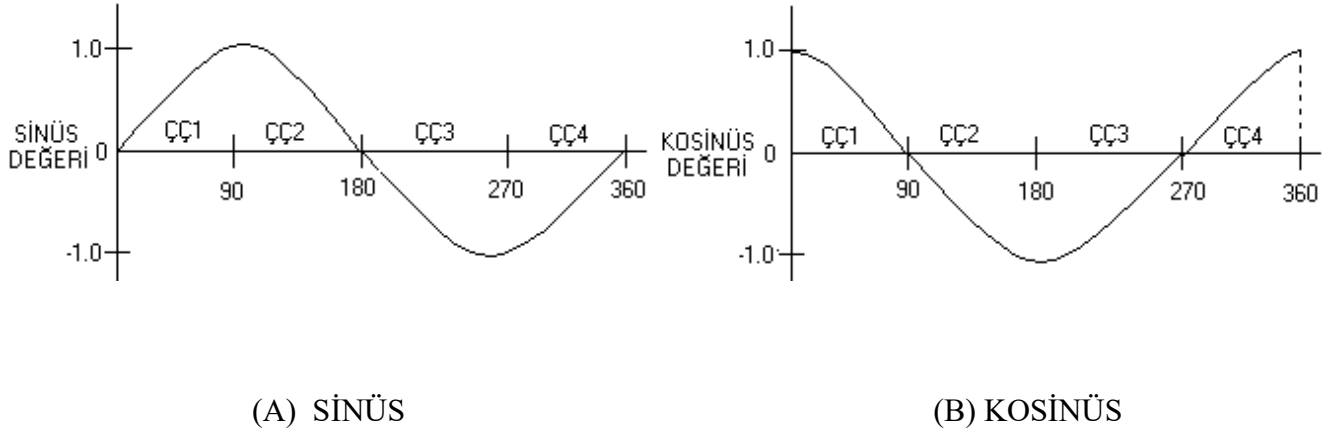
Çünkü arama uygulamaları çok yaygındır, 286 bu amaç için Translate <dönüştür> (XLAT) adında özel bir talimata sahiptir. XLAT bir byte tablosunda BX’in içeriğini temel adres gibi kullanarak ve AL’nin içeriğini bir indeks gibi kullanarak bir değeri arar ve bunu AL’nin içine geri döndürür. Bu bölüm byte tabloları ve Word tablolarının her ikisi için arama işlemlerinin örneklerini içerir.

Arama tablolarında karmaşık eşitlikleri destekleyerek işlem zamanından ve geliştirme zamanından kazanabilirsiniz. Örneğin:, bir arama tablosu bir açı için sinüs ve kosinüsü destekleyebilir.

6.9.1.Bir Açının Sinüsü:

Yüksek okuldaki trigonometriden anımsayabileceğiniz gibi, bütün açıların sinüsü 0° ve 360° formunda eğri Şekil.6.5.A’ da gösterilmiştir. Matematiksel olarak, yaklaşık değeri bu denklemlle bulabilirsiniz.

$$\sin(X) = X - \frac{X^3}{3!} + \frac{X^5}{5!} - \frac{X^7}{7!} + \frac{X^9}{9!} \dots\dots\dots$$



ÇÇ=ÇEYREKÇEMBER

Şekil.6.5. 0 ile 360 derece arasındaki açıların sinüs ve kosinüsleri.

Olası yazılan program bu hesaplamayı yapar, ancak program çalışmak için birkaç milisaniyeye gerek duyacaktır. Eğer sizin uygulamanız çok fazla sinüs eğrisi içeriyorsa, böyle bir program yazmakta zorlanabilirsiniz. Ancak bütün uygulamalar açıdan sinüse arama tablosuyla daha iyi servis yapar.

Bir uygulama herhangi bir açının 0° ile 360° arasındaki tamsayı değerlerinde açının sinüsünü gerektiriyorsa, tabloda kaç tane sinüs değeri olmalıdır – 360 mı? Hayır, bu tabloyu biz yalnızca 91 açının sinüs değerleri ile oluşturabiliriz – 0° ile 90° arasındaki her açı için bir tane.

Bunun nasıl böyle olduğunu anlamak için, Şekil 5-5A’ya yeniden bakın. Eğer çizimin en solundaki çeyreğine (0° ile 90° arasındaki açılar) Çeyrekçember1 dedik, ve bunu gördük:

1. Çeyrek çember 2 içindekilerin sinüsü (91° ile 180° arası) Çeyrekçember1’in “ayna imgesi” dir
2. Çeyrek çember 3 içindekilerin sinüsü (181° ile 270° arası) Çeyrekçember1’in “negatif tersidir” dir

3. Çeyrek çember 4 içindeki sinüsü (271° ile 360° arası) bu Çeyrekçember1'in "negatif tersi - ayna imgesi" dir

Bu şu anlama gelir: Çeyrek çember 2,3,4 teki sinüsler basit ilişkilerle Çeyrek çember 1'in içindeki sinüslere taşınabilir.

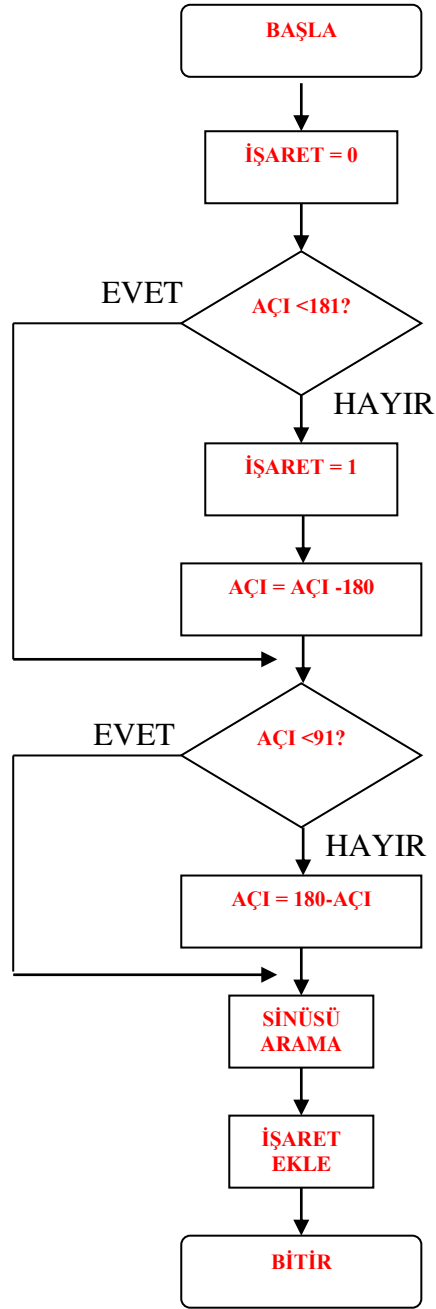
Böylelikle, bir arama tablosunda değerleri Çeyrek çember 1'in içinde sakladıysanız, programınız açıların sinüsünü herhangi bir Çeyrek çemberde bu dönüşümleri yaparak bulabilir.

<u>Açı X bunların arasındaysa:</u>	<u>değer Al</u>
0° ve 90°	$\sin(X)$
91° ve 180°	$\sin(180 - X)$
181° ve 270°	$\sin(X - 180)$
271° ve 360°	$\sin(360 - X)$

Bu ilişkiler bize bir açı-sinüs dönüşüm programı için bir akış çizelgesi inşa etmeye izin verir. Bu çizelge, Şekil.6.6'da gösterilmiştir. Burada sinüs işaret ve büyüklük değeri olarak türetilmiştir. Burada, sonucun yüksek düzendeki biti sinüsün işaretini verir (0=pozitif, 1=negatif) ve kalan bitler sinüsün büyüklüğünü verir, bu tam değerdir.

Örnek.6.9, FIND_SINE adında açıları AX'eten 0° ile 360° arasında alan ve BX'e bir 16-bit işaret ve büyüklük sinüs değeri geri döndüren bir açıdan-sinüse arama prosedürü vermektedir. Sinüsler SINES arama tablosunun içinde tamsayı olarak saklıdır. Bunları operand gibi kullanmak için 10,000' e bölmelisiniz.

Başlamak için, 286 açının boyutunu denetler. Eğer 181° 'den daha küçükse, işlemci SIN_POS'a atlar: aksi halde, işaret saklayıcısının en belirgin bitini (CX) 1 olarak ayarlar. Çünkü sinüs 180° 'nin üzerinde negatiftir – ve açıdan 180 çıkarılır.bu çıkarmayı SUB 180, AX talimatıyla gerçekleştirebilirsiniz. Ancak 286 bu formda sunuş yapmaz. Oluşan durumda, biz çıkarma işlemini AX'i reddederek yaptık sonra sonuca 180 ekledik. GET_SIN'deki 4 talimat BX'e açığı yükler, bir Word indeks formunda bunları çiftler, aramada sinüs SINE'nin içindedir, ve bunu CX'in içindeki işaret ile OR işlemine tabi tutar.



Şekil.6.6. açıdan sinüse dönüştürme programı akış çizelgesi.

Örnek.6.9: Açının sinüsünün aranması (Look up)

TITLE SINE - Açının sinüsü

; Açının sinüsünü dondurur

; Girdiler : AX : Açı (0 - 360)

; Sonuçlar : BX : sinüs, işaret ve büyüklük biçiminde

```

; AX değiştirilmemiştir
; Assemble with: MASM SINE;
; Link with: LINK callprog+SINE;
DSEG  SEGMENT PARA 'DATA'
SINES  DW  0,175,349,523,698,872    ; 0-5
        DW  1045,1219,1392,1564,1736 ; 6-10
        DW  1908,2079,2250,2419,2588 ; 11-15
        DW  2756,2924,3090,3256,3420 ; 16-20
        DW  3584,3746,3907,4067,4226 ; 21-25
        DW  4384,4540,4695,4848,5000 ; 26-30
        DW  5150,5299,5456,5592,5736 ; 31-35
        DW  5878,6018,6157,6293,6428 ; 36-40
        DW  6561,6691,6820,6947,7071 ; 41-45
        DW  7193,7313,7431,7547,7660 ; 46-50
        DW  7771,7880,7986,8090,8191 ; 51-55
        DW  8290,8387,8480,8572,8660 ; 56-60
        DW  8746,8829,8910,8988,9063 ; 61-65
        DW  9135,9205,9272,9336,9397 ; 66-70
        DW  9455,9511,9563,9816,9848 ; 71-75
        DW  9703,9744,9781,9816,9848 ; 76-80
        DW  9877,9903,9926,9945,9962 ; 81-85
        DW  9976,9986,9994,9998,10000 ; 36-40
DSEG  ENDS

PUBLIC FIND_SINE
CSEG  SEGMENT PARA 'CODE'
        ASSUME CS:CSEG,DS:DSEG
FIND_SINE PROC FAR
        PUSH DS          ; saklayıcıları kaydet
        PUSH AX
        PUSH CX
        MOV  BX,DSEG      ; DS'yi Başlangıç durumuna getir
        MOV  DS,BX
        SUB  CX,CX        ; İşareti 0'la
        CMP  AX,181       ; Açı < 181 derece?

```

```

JB SIN_POS      ; Evet. İşaret 0'la devam et
MOV CX,8000H    ; Hayır. İşareti 1 yap
SUB AX,180      ; Açıdan 180 çıkar
SIN_POS: CMP AX,91      ; Açı < 91 derece?
JB GET_SIN      ; Evet. sinüs aramaya git
NEG AX          ; Hayır. Açıdan 180 çıkar
ADD AX,180
GET_SIN: MOV BX,AX      ; Açıyı bir Word index'i yap
SHL BX,1
MOV BX,SINES[BX] ; ve sinüs değerini ara
OR BX,CX        ; sinüsü işaret bit'iyle birleştir
POP CX
POP AX
POP DS
RET             ; ve çık
FIND_SINE ENDP
CSEG ENDS
END

```

6.9.2.Açının Kosinüsü:

Şekil.6.5.B'de gösterildiği gibi kosinüs eğrisi sinüs eğrisinin bir çeyrek çember sola kaydırılmasından başka bir şey değil. Böylelikle, herhangi bir verilen açının kosinüsü 90° daha büyük açının sinüsüne eşittir. Böyle:

$$\cos(X) = \sin(X+90)$$

bunu bilerek, Örnek 5-9'deki SINES tablosunu bir açının kosinüsünü bulmak için sinüsünü bulmak kadar iyi kullanabiliriz. Örnek 5-10 bunu yapan bir prosedürü gösterir. FIND_SINE'deki gibi, FIND_COS'un sonucunu 10,000'e bölmelisiniz.

Tesadüfi olarak, sinüs ve kosinüs eğrileri dikey ekseninde simetrikler; böylece, negatif açıların sinüs ve kosinüsleri pozitif karşıtlarıyla aynı büyüklüklere sahiptir. Örneğin:, -25° 'in sinüsü ve kosinüsü $+25^\circ$ 'ninkiyle aynı büyüklüktedir. Bu şu anlama gelir; -1° ile -360° arasındaki açılar için eğer 'AX'in içindeki kedin değerleri destekliyorsa FIND_SINE ve FIND_COS'u zaten kullanabiliyorsunuz.

Örnek.6.10: Açının kosinüsünü arama (Look up)

```
TITLE    COSINE - Açının kosinüsü
; Açının kosinüsünü dondurur
; Girdiler : AX : Açı (0 - 360)
; Sonuçlar : BX : kosinüs, işaret ve büyüklük biçiminde
; AX değiştirilmemiştir
; FIND_SINE (Örnek 5-9) prosedürü çağrılır
; Assemble with: MASM COSINE;
; Link with: LINK callprog+COSINE+SINE;
        EXTRN FIND_SINE:FAR
        PUBLIC FIND_COS
CSEG    SEGMENT PARA 'CODE'
        ASSUME CS:CSEG,
FIND_COS PROC FAR
        PUSH AX
        ADD  AX,90      ; FIND_SINE ile kullanmak için 90 ekle
        CMP  AX,360     ; Sonuç > 360?
        JNA  GET_COS
        SUB  AX,360     ; Öyleyse, 360 çıkar
GET_COS: CALL FIND_SINE ; Kosinüs arama
        POP  AX
        RET
FIND_COS ENDP
CSEG    ENDS
        END
```

6.9.3.Arama Tabloları Kod Dönüşümlerini Gerçekleştirme:

Arama tabloları görüntü kodları gibi data kodlarını, printer kodlarını ve mesajları da tutabilir. Örnek.6.11, çoklu arama gerçekleştiren bir prosedürü göstermektedir. Bu AL'deki hexadesimal sayıyı ASCII, BCD, EBCDIC eşitine çevirir ve bunları sırasıyla CH,CL ve AH'in içine geri döndürür. Veriyi bilgisayar ve sistemdeki yazıcı, görüntü aygıtı ya da bazı diğer çevre aygıtları arasında veri gönderdiğinizde, bunarı ASCII (American Standart Code for Information Interchange) formunda alır. EBCDIC (Extended Binary-Coded Decimal Interchange Code) veri işleme ve haberleşme sistemleri için bir iletim protokolüdür.

Örnek.6.11: Bir hex sayıyı ASCII, BCD ve EBCDIC 'ye çevirir.

```
TITLE  CONV_HEX - Hex'i ASCII, BCD, EBCDIC'çevirir
; Bir hexadesimal sayıyı onun ASCII, BCD, EBCDIC eşitine çevirir
; Girdiler : AL : Hex sayı
; Sonuçlar : CH = ASCII karakter
           CL = BCD sayı
           AH = EBCDIC karakter
; AL değiştirilmemiştir
; Assemble with: MASM CONV_HEX;
; Link with: LINK callprog+CONV_HEX;
DSEG  SEGMENT PARA 'DATA'
ASCII  DB  '0123456789ABCDEF'
BCD    DB  0,1,2,3,4,5,6,7,8,9,10H,11H,12H,13H,14H,15H
EBCDIC DB  0F0H,0F1H,0F2H,0F3H,0F4H,0F5H,0F6H,0F7H
        DB  0F8H,0F9H,0C1H,0C2H,0C3H,0C4H,0C5H,0C6H
DSEG  ENDS

PUBLIC CONV_HEX

CSEG  SEGMENT PARA 'CODE'
        ASSUME CS:CSEG,DS:DSEG
CONV_HEX PROC FAR
    PUSH DS        ; Sklayicilari kaydet
    PUSH BX
    PUSH DX
    MOV BX,DSEG    ; DS'yi Başlangıç durumuna getir
    MOV DS,BX
    MOV DL,AL      ; Girdi değerini DL'nin içine kaydet
    LEA BX,ASCII   ; ASCII değeri ara
    XLAT ASCII
    MOV CH,AL      ; ve bunu CH'in içine yükle
    MOV AL,DL
    LEA BX,BCD     ; BCD değeri ara
    XLAT BCD
```

```

MOV CL,AL      ; ve bunu CL'in içine yükle
MOV AL,DL
LEA BX,EBCDIC  ; EBCDIC değeri ara
XLAT EBCDIC    ; ve bunu AH'in içine yükle
MOV AH,AL
MOV AL,DL      ; ve bunu AH'in içine yükle
POP DX         ; saklayıcıları yerine koy
POP BX
POP DS
RET            ; ve çık
CONV_HEX ENDP
CSEG ENDS
END

```

6.9.4.Dallanma Tabloları:

Bazı arama tabloları veri değerlerinden daha çok adresleri içerirler. Örneğin: bir hata rutini olası mesaj setinin içinde dikkate değer bir mesajın adresini bulmak için bir arama tablosu kullanabilir. Benzer bir şekilde, bir kesme servis rutini dikkate değer bir aygıt gerektiğinde servisin tipine bağlı olarak birkaç kesme tutucu programlarında birini çağırmak için bir arama tablosu kullanabilir. Bir başka rutin bir kullanıcının menüdeki hangi seçeneği seçtiğine bağlı olarak birkaç altprogramdan birini çağırmak için bir arama tablosu kullanabilir. Bütün bu uygulamalarda (ve daha fazlasında), arama tablosunun tuttuğu adres dallanma tablosu diye adlandırılır.

Örnek.6.12, bir bilgisayar yardımcı eğitim programı içinde görebileceğiniz dallanma tablosu uygulamasını gösterir. Bu prosedür, SEL OPT, ADDITION, SUBTRACTION, MULTIPLICATION ya da DIVISION rutinlerini bir öğrencinin menüden 0,1,2 ya da 3 seçip seçmediğine bağlı olarak aktarım yapar.

SEL_OPT girilen kodun geçerli olup olmadığını denetler, ve eğer 3'ten büyükse hata raporlama rutinine dallanır. A SEL_OPT'un uygun rutine dolaylı dallanma yapmak için kullandığı bir indeks olursa geçerli bir koddur. Kullanıcı bitirdiğinde, rutinin RET talimatı denetimi programın SEL_OPT kısmına geri döndürür. Bu durumda bir menü görüntülenir.

Örnek.6.12: Çoklu Seçenek Seçin Prosedürü:

- ; Bu prosedür kullanıcının seçtiği AL içindeki koda göre dört
- ; yordamdan (routine) birini çalıştırır.
- ; AX ve DI'nin içeriği değiştirilmiştir.

; Bu adres tablosunu data segmentin içine sakla

CHOICE DW ADDITION,SUBTRACTION,MULTIPLICATION,DIVISION

; Secim prosedürü burada

SEL_OPT PROC FAR

CMP AL,3 ; Geçersiz secim?

JA ERROR

CBW ; Hayır. Kodu Word'e çevir,

MOV DI,AX ; sonra kodu DI'nin içine taşı

SHL DI,1 ; ve bir index'e dönüştür

JMP CHOICE[DI] ; Seçilen yordama dallan

ERROR: ...

...

RET

ADDITION:

...

...

RET

SUBTRACTION:

...

...

RET

MULTIPLICATION

...

...

RET

DIVISION

...

...

RET

SEL_OPT ENDP

6.10. Metin Dosyaları:

Önceki bölümde veri yapılarının içerdiği sayılarla çalışmıştık. Ancak, kelime işlemci ve diğer uygulamalar, sayısal olmayan bilginin işlenmesini içerir – öncelikle netin dosyalarını.

Metin dosyaları ASCII karakterler stringlerini listeler. Örneğin: Bir şirket için Personel bilgilerini tutan bir metin dosyası her personel için bir element ya da kayıt içerecektir. Döngü içinde, her kayıt çalışanların isimlerini, kimlik numaralarını, değişimi, maaş oranını ve bunun gibi şeyleri listeleyen birkaç alt kayda ya da alana sahiptir. String talimatları dikkate değer bir şekilde metin dosyaları üretmek için uygundur.

Metin dosyalarını sayısal dosyalar için uyguladığınız aynı basit tekniklerle üretebilirsiniz. Ancak, yapının çoklu kaydından dolayı, metin dosyalarını işleyen programlar basit byte ya da Word işleyenlerden biraz farklı olabilir.

Örneğin, Bir metin dosyası aramak genellikle her girişin bir kısmını (belki de yalnız isim kısmını) aranan değerle karşılaştırmayı gerektirir. Bu tün girişi karşılaştırmaktan daha iyidir. Benzer bir şekilde, kabarcık sırlamada bir metin dosyası bitişik girişlerin tek alanını karşılaştırır, ancak bir değiş tokuş gerektiğinde tüm girişler taşınır.

Metin dosyası operasyonunun basit bir örneğinde olduğu gibi, isimlerin ve telefon numaralarının bir listesini sıralayan bir programa bakalım. Listenin ilk bölgesi girişin sayacı olan bir “iki-byte” tutar.

Listedeki her giriş 42-byte uzunluğundadır ve üç alana bölünmüştür: bir 15-byte’lın sayısını, bir 15-byte’lık ilk/orta isim ve 12-byte’lık telefon numarası. Bir alanda bazı kullanılmayan byte’lar ASCII ‘boş (blank)’ karakter içerdiği varsayılır. Böylece, Listede tipik bir giriş buna benzer:

DB ‘CORNELL’,8 DUP (‘ ’)

DB ‘RAY’,12 DUP (‘ ’)

DB ‘728-732-8437’

(Kuşkusuz, siz doğal olarak klavye yardımıyla yazarak inşa ediyorsunuz. Şimdi ise, bellekte zaten var olan bir dosyayı varsayalım.)

Örnek.6.13, ekstra segmentin içinde sıralı telefon listesini bubble-sort ile sıralayan PHONE_NOS prosedürünü göstermektedir. Bunun yapısı Örnek.6.5’deki BUBBLE prosedürüne benzemektedir. (NOT: PUSHA ve POPA’nın kullanımı genel saklayıcıları korumak içindir. Bu talimatlar 80286 için tek olduğu sürece, bu prosedürü IBM PC’de ya da diğer 8086 ya da 8086 tabanlı bilgisayarda çalıştırmak için bu talimatları PUSH’lara ve POP’lara dönüştürmelisiniz.)

Örnek.6.13: Bir Telefon Listenin Sıralanması:

```
TITLE  PHONE_# - Bir telefon listesini sırala
; Bu prosedür bir telefon listesini alfabetik olarak sıralar
; Liste kişisel girişleri izleyen giriş sayaç Word'lerinden oluşur
; Her giriş 42 byte uzunluğundadır ve uç alana bölünmüştür:
; soyad (15 byte), iki ad (15 byte) ve telefon numarası (12 byte).
; Input: ES:DI = Girdi sayaç Word'ünün adresi
; Assemble with: MASM PHONE_#;
; Link with: LINK callprog+PHONE_#;
.286c

DSEG  SEGMENT PARA 'DATA'
SAVE_CNT DW  ?
FIRST_ENT DW  ?
DSEG  ENDS

      PUBLIC PHONE_NOS

CSEG  SEGMENT PARA 'CODE'
      ASSUME CS:CSEG,DS:DESG
PHONE_NOS PROC FAR
      PUSHA
      MOV AX,DSEG      ; DS'yi Başlangıç durumuna getir
      MOV DS,AX
      CLD
      MOV CX,ES:[DI]   ; Element sayacını hafızadan oku
      MOV SAVE_CNT,CX  ; Bu değeri belleğe kaydet
      ADD DI,2         ; İlk girdinin adresini al
      MOV FIRST_ENT,DX ; ve bu değeri belleğe kaydet
INIT:  MOV BX,1        ; Değiş tokuş bayrağı (BX) = 1
      DEC SAVE_CNT     ; sayaç-1 karşılaştırması için hazır ol
      JZ SORTED        ; SAVE_CNT 0 ise çık
      MOV CX,SAVE_CNT  ; Karşılaştırma sayacını CX'e yükle
      MOV BP,FIRST_ENT ; ve ilk girdi adresi BP'nin içinde
NEXT:  MOV DI,BP        ; bir girdi DI ile adresleniyor
      MOV SI,BP         ; ve sonraki girdi SI ile
```

```

ADD SI,42
PUSH CX      ; Güncel Karsılařtırma sayacını kaydet
MOV CX,15    ; 15-byte soyad'ı karşılařtır
REPE CMPS ES:BYTE PTR[SI],ES:[DI]
                ; sonraki soyad < bu soyad?
JAE CONT     ; Hayır. Sonraki çifti denetlemeye git
MOV CX,42    ; Evet. Bu girdileri Deęiş tokuř et
SWAPEM: MOV AL,ES:[BP]
XCHG ES:[BP+42],AL
INC ES:[BP],AL
INC BP
LOOP SWAPEM
SUB BX,BX    ; Deęiş tokuř bayraęını = 0 yap
CONT: POP CX    ; Karsılařtırma sayacını geri yükle
LOOP NEXT    ; sonraki iki adi karsılařtırmaya git
CMP BX,0     ; Deęiş tokuřlar yapıldı mı?
JE INIT      ; Eęer yapıldıysa, bir dięer geçiř yap
SORTED: POPA    ; Yapılmadıysa, saklayıcıları yükle
RET          ; ve çık
PHONE_NOS ENDP
CSEG ENDS
END

```

Başlamak için, prosedür giriş sayacını ve iki deęişkenin içinden ilk girişin adresini okur, SAVE_CNT ve FIRST_ENT. Talimat NEXT ve SWAPEM arasında CMPS işlemini ayarlar ve çalıştırır. Burada,bu alanlar belleęin içinde ayrı olarak 42-byte yer kapladığı yerde, DI noktaları bir girişin soy isim alanına, SI noktaları yeni girişin soy isim alanına yüklenir.

SWAPEM'deki döngü gerektiğinde iki giriři deęiş tokuř eder. Giriřler 42-byte uzunluęunda olduğundan bu yana, döngü sayacı CX 42'ye başlangıç durumuna getirilmiştir. Prosedürün kalanı Örnek 5-5 le benzerdir.

Not: PHONE_NOS çift soy isimleri hesaplayamaz. Birinin bulunması üzerine, ilk olarak bu girişlerin isim alanlarına bakar ve ikinci olarak bunları alfabetik düzende sıralar. Örnek 5-132ü bunu yaparak deęiřtirebilirsiniz.