İSTANBUL TEKNİK ÜNİVERSİTESİ * FEN BİLİMLERİ ENSTİTÜSÜ

IBM 4381 VM/SP ORTAMINDA KULLANICILAR ARASI İLETİŞİM ARABİRİMİYLE BİR VERİ TABANI UYGULAMASI

YÜKSEK LİSANS TEZİ

Müh. Faruk KOCABIYIK

Ana Bilim Dalı : KONTROL VE BİLGİSAYAR MÜHENDİSLİĞİ

Programı : KONTROL VE BİLGİSAYAR

TEMMUZ 1991

ISTANBUL TECHNICAL UNIVERSITY * INSTITUTE OF SCIENCE AND TECHNOLOGY

A DATABASE APPLICATION ON INTER USER COMMUNICATION VEHICLE (IUCV) IN VM/SP ENVIRONMENT FOR IBM 4381

M.S.THESIS

Faruk KOCABIYIK, B.S.

Date of Submission : 10 June 1991
Date of Approval : 5 July 1991

Examination Committee
Supervisor : Assistant Prof.Dr.Nadia ERDOĞAN
Member : Prof.Dr.Emre HARMANCI
Member : Prof.Dr.Nadir YÜCEL

JULY , 1991

PREFACE

I express my indebtedness to Dr. Nadia Erdoğan, my thesis supervisor, from her specialized knowledge I have benefited.

I wish to acknowledge interests showed by the staff of Data Processing Center in various ways.

TABLE OF CONTENTS

SUMMARY

A DATABASE APPLICATION ON INTER USER COMMUNICATION VEHICLE IN VM/SP ENVIRONMENT FOR IBM 4381

In this study, a database application is implemented using Inter User Communication Vehicle facility.

The Inter User Communication Vehicle (IUCV) is a communications facility that allows a program running in a virtual machine to communicate with other virtual machines, with a CP system service, and with itself.

My application program mainly involves the recording and searching of publications. It allows the user to record his publication or search for any item in the database of ITU Control and Computer Engineering Department. Based on IUCV, the program was written in REXX Language in VM/SP (Virtual Machine/System Product) environment.

In this paper, Parallel Processing and the IUCV facility are explained. Also the REXX Language and CMS Window support are introduced and some aspects are reflected to the reader. Finally the database program is listed.

ÖZET

IBM 4381 VM/SP ORTAMINDA KULLANICILAR ARASI İLETİŞİM ARABİRİMİYLE BİR VERİ TABANI UYGULAMASI

Kullanıcılar arası iletişim arabirimi bir görüntü makinada işleyen bir programın kendisi ve diğer görüntü makinalarla bir CP (Kontrol Programı) sistem hizmet programı yardımıyla haberleşebilmesini sağlayan bir iletişim programıdır.

Bir IUCV iletişimi, bir kaynak ve bir hedef iletişimci arasında yer alır. İletişim önceden tanımlanan bir bağlantı yolu üzerinde olur. Her bir iletişimci birçok yola sahip olabilir ve aynı yol üzerinde eşzamanlı olarak birçok mesaj gönderebilir veya alabilir.

IUCV, yine IUCV adlı makro komutuyla aşağıdaki fonksiyonları sağlar:

- Haberleşme yolları oluşturmak ve bozmak (ayırmak),
- Mesajlar yollamak ve mesajlara karşılık vermek,
- Mesajları kabul etmek veya reddetmek,
- IUCV olaylarının sırasını kontrol etmek.

İletişimciler IUCV olayları hakkındaki bilgiyi, IUCV dış kesmelerini ele alarak sağlar.

VM/SP İŞLETİM SİSTEMİ

VM/SP (Virtual Machine/System Pruduct): Görüntü Makina/Sistem Ürünü anlamını taşır. VM/SP, eldeki bilgisayar sisteminin daha verimli olması için kaynak paylaşımı sağlayan bir işletim sistemidir.

Real Machine (Gerçek Makina): VM/SP ile çalışırken ihtiyaç duyulan gerçek işlemci, kanallar (diğer bir deyimle Giriş/Çıkış İşlemcileri), bellek ve Giriş/Çıkış Cihazlarıdır.

Virtual Machine (Görüntü Makina): Bir kullanıcının terminalinden kontrol edebildiği, bir bilgisayar ve bağlı cihazlarının fonksiyonel bir benzetimi (simulasyonu).

Userid (Kullanıcı Kodu): Bir kullanıcıyı sisteme tanımlayan, önceden belirlenmiş maximum 8 uzunluktaki karakter takımı.

Directory: VM/SP içinde her bir görüntü makinanın normal konfigürasyonunu tanımlayan bir CP (Kontrol Programı) disk dosyası. Konfigürasyon şunlardan oluşur: Kullanıcı Kodu, Şifre, Görüntü Bellek kapasitesi, izin verilen CP özel komut sınıfı, işlemciye erişme (Dispatching) önceliği, kullanılacak mantıksal yazma sembolleri, vs...

Auxiliary Storage (Yardımcı Bellek): Ana bellekten ayrı olan bellek birimi. VM/SP'de bu birim genellikle, erişim zamanı yerden bağımsız olan doğrudan erişimli bir cihazdır.

Time Sharing (Zaman Paylaşımı): Birden fazla kullanıcının programlarını bilgisayarda -görünürde- paralel olarak çalıştırmaları ve program boyunca programlarıyla etkileşimde bulunabilmeleri.

Virtual Storage (Görüntü Bellek): Görüntü adreslerin gerçek adreslere dönüştürüldüğü (Mapping) bir bilgisayar sisteminin kullanıcılarınca adreslenebilen ana bellek olarak görülebilen bellek alanı. Görüntü adresler bu alanı ya da görüntü giriş/çıkış cihazlarını adresleyebilirler. Görüntü bellek büyüklüğü bilgisayar sisteminin adresleme olanağı ve eldeki yardımcı bellek miktarı ile sınırlanır; yoksa ana belleğin büyüklüğüyle değil.

VM/SP'nin Özellikleri

VM/SP ile her bir kullanıcı bir bilgisayar sisteminin fonksiyonel eşdeğerine sahiptir. Görüntü makinalar gerçek değildir ama gerçek bir sistem gibi iş yapabilirler. VM/SP, Dosya yönetimi sağlayan, İngilizce çağrışımlı bir komut dili olan ve Tam Ekran (Full Screen) editör imkanı veren bir sistemdir.

Sistemde görüntü makinaların kullanılmasıyla zaman paylaşımı ve çoklu programlama bütün sisteme yayılır. VM/SP her bir görüntü makinanın yaptığı işi düzenler ve kontrol eder, sistem kaynakları ve zamanı yönetir. Her bir görüntü makina gerçek makinanınkine benzer fonksiyonlar sağlar ve böylece gerçek makinanın güçlÜ performansını paylaşabilir.

VM/SP görüntü makinaları
- Değişik konfigürasyonda olabilir,
- Değişik işletim sistemleriyle çalışabilir,
- Etkileşimli (İnteraktif) ya da yığın (Batch) işlem yapabilir,
- Diğer görüntü makinalarla -görünürde- paralel olarak çalışabilirler.

VM/SP'yi kullanarak bir görüntü makinada çalışmak üç adımda olur:

1) İşlem: Kullanıcı, bilgisayara doğrudan ya da telefon hattı ile bağlı bir terminali kullanarak LOGON işlemi ile görüntü makinasını açar.

Sonuç: Sistem o anki zamanı ve tarihi gösterir. Bu mesaj kullanıcının VM/SP ile başarılı bir şekilde iletişim kurduğunu ve bir terminal döneminin başladığını gösterir.

2) İşlem: Kullanıcı arzu edilen işletim sistemini görüntü makinasına IPL (Initial Program Loading:İlk Program yükleme) işlemiyle yükler. VM/SP'nin IPL işlemini otomatikman yapması da sağlanabilir.

Sonuç: Sistem artık kullanıcının istediği işi yapması için hazırdır.

3) İşlem: İşini bitiren kullanıcı terminal dönemini bir LOGOFF işlemiyle kapatır.

Sonuç: Sistem zamanı, tarihi ve diğer bazı bilgileri göstererek dönemi sonlandırır.

VM/SP işletim sistemi bir tek gerçek bilgisayarda, başka bir deyişle fiziksel donanım üzerinde birden fazla görüntü bilgisayar gösteren ve yöneten bir işletim sistemidir. Yani VM/SP işletim sistemi bir tek gerçek bilgisayarı dışarıya birçok bilgisayar varmış görünümü vererek yönetir.

VM/SP işletim sistemi, büyük ana bellek, hızlı ve büyük kapasiteli yardımcı bellek ve kontrol ettiği diğer çevre cihazlarını bir ahenk içinde yöneterek koordinasyon sağlar. İşletim sistemi belleğin optimum kullanmını da hedefler.

VM/SP işletim sistemi hızlı ve yüksek kapasiteli yardımcı bellek cihazı sayesinde, aslında yardımcı bellek üzerindeki bölgeler olarak tanımlanmış bulunan, bir bilgisayarın normal konfigurasyonunu haiz, görüntü makina olarak adlandırılan görüntü bilgisayarlar yaratır ve yönetir. Her görüntü makinanın yine yardımcı bellek üzerinde belirlenmiş alanlardan ibaret olan, gerçek çevre cihazları imajındaki görüntü çevre cihazlarını da oluşturur. Bu görüntü makinaların gerçek çevre cihazlarına olan ihtiyaçlarını da istek halinde belli bir hiyerarşiye göre zaman paylaşmalı olarak karşılar.

Görüntü Makina Konfigürasyonu, CP ve CMS

Görüntü Makina (VM), VM/SP işletim sisteminin getirdiği görüntü bilgisayar sistemidir; her görüntü makinanın sistem kaynaklarından ne şekilde yaralanabileceği ve ona ait görüntü çevre cihazları Directory'sinde belirlenmiştir.

Bilgisayar açıldığında bu makinada çalışabilmek için önce gerçek belleğe CP (Kontrol Programı) yüklenir. Bu program işletim sisteminin simulasyon kısmını oluşturur. Altındaki programlara servis verir, yani donanım erişimini sağlar.

CMS'e VM'in işletim sistemi gözüyle bakılabilir. Fakat CP altında çalışmak zorundadır. İşletim sisteminin Yazılım ilişkilerini sağlar.

CP bir yönetici (Supervisor) programdır. Tek başına bir işletim sistemi değildir. Fakat bir işletim sistemiyle (CMS) birlikte çalışır.

Görüntü Makinanın kullanıcıya hizmet vermesi CMS'in yüklenmesiyle mümkün olur. Bu işlem IPL ile sağlanır.

CP (Kontrol Programı) sistem kaynaklarını yönetir ve işletim sistemlerinin çalıştırabildiği görüntü makinalar yaratır. CP, görüntü makinalar kadar, gerçek işlemciye normal olarak kontrolünde bulunduran işletim sistemlerini de normal olarak destekler (Bunlar VM/SP, DOS/VSE ve OS/VS gibi sistemlerdir).

CP komutları kullanıcılara VM/SP sistemini ve görüntü makinaları kontrol etme imkanını verir. CP komutları öncelik sınıflarına ayrılabilir. Her bir kullanıcı bir ya da daha çok öncelik sınıfına atanabilir. Bu sınıflar bir kullanıcının çeşitli fonksiyonlara erişimini sınırlandırır. CP aşağıda belirtilen fonksiyonları gerçekler:

- Görüntü makina ve gerçek makina adreslerini oluşturmak,
- Görüntü makinaları işlemciyer eriştirmek (Dispatching),
- Gerçek makinayı kontrol etmek,
- Görüntü makinalara disk alanı atamak,
- Kullanıcının diske giriş ve çıkış işlemlerini yönetmek.

CMS (Conversational Monitor System:Görüşmeli Monitor Sistemi), VM/SP için oluşturulmuş bir işletim sistemidir. CMS, CP'ye gereksinim duyar. CMS kullanıcılara programlarını bir terminal aracılığıyla etkileşimli olarak çalıştırabilme imkanını verir. CMS komutlarıyla, doğrudan erişimli bellek cihazlarında (DASD) saklı CMS verilerini işlemek mümkün olur.

CMS kendi dosyalarını yönettiği için, kullanıcıların dosyaların yalnız ismiyle ilgilenmeleri yeterli olmaktadır. Kullanıcıların tek tek dosya alanlarıyla ilgilenmelerine gerek yoktur; CMS, kullanıcıya ayrılan blok alanlarını dinamik olarak düzenler.

EXEC PROSEDÜRLERİ

VM/SP'nin üç adet komut dili vardır. Bir dildeki komutları bir dosyada toplayıp tek bir komutla yürütmemiz kullanım açısından kolaylık sağlar. Belli kurallarla hazırlanmış, bu tür dosyalar 'EXEC' prosedürleri, veya kısaca EXEC ler olarak bilinirler.

Bir EXEC, herhangi bir adla bir CMS minidiskinde yer alan ve dosya tipi EXEC olan bir dosyadır. Çeşitli CP, CMS ve o exec dilinin komutlarından oluşur. Bu komutlar EXEC'in dosya adının girilmesiyle yürütülürler.

EXEC prosedürlerinin yazılabileceği üç ayrı EXEC dili vardır. Bir exec'in hangi dilde yazılmış olduğu EXEC'in ilk satırı ile anlaşılır. Exec'lerin yazılabileceği diller :

1)EXEC: İlk satırda CONTROL komutu bulunur.
2)EXEC2: İlk satırda TRACE komutu bulunur. EXEC dilinin daha geliştirilmiş şeklidir.
3)REXX: İlk satırı bir açıklama ile başlar. Diğerlerine göre en gelişmiş olanıdır.

REXX Dilinde bir Exec'in Yazılması

Her REXX cümleciği sırasıyla şu şekilde işlem görür :

1. Büyük harfe çevrilir, açıklamalar atlanır, katarlar değiştirilmez.
2. Komutların yerine koyma işlemi yapılır.
3. Komutlar ve atamalar yürütülür.

Temel Kurallar

* Her cümlecik bir satıra yazılır.

* Aynı sıraya birden fazla cümlecik yazmak gerekirse bunlar noktalı virgül ile ayrılmalıdırlar.

* Alt satıra devam etmek gerekiyorsa, bulunulan satırın sonuna virgül konulmalıdır.

* Özel anlamı olan karakterlere (= + / ( ) %) dikkat edilmelidir.

Açıklamalar /*..... */ işaretleri arasında yer alır. Programın her yerinde kullanılabilirler.

REXX dili ile yazılmış bir EXEC prosedürünün ilk satırı açıklama olmak zorundadır.

Boş satırlar programın bölümlere ayrılması ve kolay takibi için kullanılır. REXX çalışırken esnasında bunları atlar.

Etiketler, programımızın sapmasını istediğimiz noktaları belirler. Etiketleri ':' takibeder.

Örnek ==> Döngü:

Komutlar

REXX komutları aşağıda verilmektedir. Bunların dışında tırnak içinde olmak üzere bütün CP ve CMS komutları da kullanılabilir.

Katarlar, tırnak içine yazılan bilgilerdir. REXX kataın içeriğine dokunmaz.

Atamalar

sembol = değer

Burada sembol bir değişkenin adıdır. Atama yapılırken uyulması gereken kurallar:

1. Özel karakterler içermemelidir.
2. Sayısal bir karakter ile başlamamalıdır.
3. Değişken bir eki ifade eden '.' (nokta) içerebilir.

EXECIO Komutu

Bu CMS komutunun işlevleri şunlardır :

* Disk veya varsayılan okuyucudan satırları STACK'e okumak,
* STACK'daki satırları CMS disk kütüğüne, varsayılan delgi veya yazıcıya yazmak.

EXECIO'nun Kullanım Alanları

* EXEC'lerin içinden basit bir şekilde dosya yönetimi.
* Bir CMS kütüğünde doğrudan erişimle güncelleştirme yapmada.
* Sistemdeki önemli EXEC'lerin yürütülme kaydını tutmak.
* CP komutlarından gelen yanıtları stack'e yazmak.

Pencereler

Bu çalışmada kurulan veri tabanını oluşturmada ve yayın tarama olayında EXECIO komutundan büyük ölçüde yararlanılmıştır. Yararlanılan bir diğer imkan da VM/SP Rel.5 yazılımının bir yeniliği olan pencerelerdir.

Üzerinde çalışılan ekranı masa, üstünde işlem yapılan kağıdı pencere olarak düşünürsek VM/SP kullanıcıya şimdiye kadar bir masa ve bir çalışma kağıdı sağlıyordu. Ancak VM/SP Rel.5'nin getirdiği yenilik kullanıcıya birden fazla masa ve her masanın üstünde istediği şekilde yerleştirip, istediği şekilde işleyebileceği birden fazla kağıt (pencere) tanımlama olanağı vermektedir. İstendiği takdirde masa değiştirilebilir. Ancak bir anda bir masada çalışılabilir. Bir masada çalışırken, öndeki çeşitli kağıtlar üstünde, aynı masa ile ilgili değişik işler yapılabilir.

Varsayılan Ekranlar

Bir varsayılan ekran gerçek bir ekranın terminalinizin kısıtlamalarından etkilenmeden ekranımızda yaratılması olarak düşünülebilir.

Örneğin, renksiz bir terminal olan 3278 de 100 kolona, 300 satırlık bir pencere tanımlayıp programımızın çıkışını buraya renk nitelikleri vererek yazmasını sağlayabiliriz. Pencere yöneticisi bunu gerçekleştirecektir.

Pencere varsayılan ekranın görüntülendiği, buradaki bilginin üzerinde değişiklik yapılabildiği gerçek ekran üzerinde bir alandır. Bir kerede en çok 255 pencere tanımlanabilir, ve en fazla çalışılan ekranın büyüklüğünde olabilir.

Ekranın herhangi bir yerinde bulunabilir, kısmen veya tümüyle üst üste gelebilir, ve de kullanıcının istediği sırada görüntülenebilirler.

Yayın Veri Tabanı

Bu çalışmada REXX exec dili kullanılmış ve IUCV arabirimini execlerde de kullanabilmeyi sağlayan IUCV module'unden yararlanılmıştır.

Kurulan veri tabanının pilot olarak Kontrol ve Bilgisayar Bölümünün yayınlarını kapsaması öngörülmüş; tutulan yolun performans değerlendirilmesinin daha kolay yapılması için bu gerekli görülmüştür.

Kullanıcı programı çağırdıktan sonra ekranda çıkan pencerede kullanıcıya yayın girmek mi yoksa yayın taramak mı istediği sorulmaktadır. Yayın girme seçilirse yayın girişi genel penceresi ekranda belirmekte ve kullanıcıdan yayın adı, yazar, ünvan, yayın cinsi ve dili gibi birtakım bilgiler istenmektedir. Bunlardan ünvan, yayın cinsi, dili vb. gibi bir takım bilgilerin ekranda verilen kodlara göre girilmesi program ve bellek tasarrufu açısından gerekli görülmüştür. Kodların yanlış girilmesi durumunda kullanıcı uyarılmakta ve bilgileri doğru girmesi istenmektedir. Daha sonra kullanıcının girdiği yayın tipinin (Bilimsel Makale, Yayınlanmış Bildiri, Kitap, Ders Notu, Yayınlanmış Rapor, Y.Lisans veya Doktora tezi) gerektirdiği bilgileri isteyen pencereler ekrana gelmekte ve kullanıcı bu alanları da doldurmaktadır. Kullanıcı yayını hakkındaki gerekli bilgileri girip bu bilgileri onayladıktan sonra program ekrandan bilgileri okuyup veri tabanı görüntü makinasına yollamaktadır. Veri tabanı görüntü makinası ilk gelen veri parçasından bunun veri kaydı isteği olduğunu anlamakta ve bundan sonra o kullanıcıdan gelen verileri sırasına uygun şekilde alıp veri tabanına kaydetmektedir.

Eğer kullanıcıdan gelen istek yayın taramaysa, gelen bilgiler tarama kriteri olarak algılanmakta ve tarama sonucu bulunabilecek bilgiler yine IUCV (Kullancılar Arası İletişim Arabirimi) yoluyla kullanıcı makinasına aktarılmaktadır. Kullanıcı programı da bu bilgileri yine pencereler aracılığıyla ekranda ekranda göstermektedir.

Veri Tabanı programında ilk olarak IUCV yüklenmekte ve IUCV'nin bütün kullanıcılardan gelebilecek isteklere açık olması için de 'IUCV CONNECT *MSG' deyimi kullanılmıştır. Bu deyimin icrasının -herhangi bir sebeple- olumlu sonuçlanmaması durumunda 'FARUK' görüntü makinasına 'IUCV ye bağlanılamadı' şeklinde bir mesaj yollanmakta ve program yazarı böylece uyarılmaktadır.

Eğer IUCV yüklenmiş ve '*MSG' sistem servisine bağlanılmışsa, program

```
File.1 = 'ARTICLE DATA A'
    .
    .
File.7 = 'THESIS DATA A'
```

gibi deyimlerle icraya devam etmektedir. Bu tanımlamaların amacı gelen yayın kaydı isteklerini yayın tipine göre sınıflandırmak ve her yayını kendi sınıfına ait dosyada saklamaktır. Yapıyı bu şekilde kurmanın getirdiği yarar, yayın taraması yapılırken tarama süresinin kısalmasını sağlamasıdır. Bundan sonra program program temel olarak

```
Do forever
        Pull msg
        Select
                When msg.1 = 'RECORD' then call RECORD
                When msg.1 = 'SEARCH' then call SEARCH
                Otherwise nop
        End
End
```

sonsuz döngüsünde kalmaktadır. Yani kullanıcıdan gelen verinin ilk sözcüğü 'RECORD' ise RECORD adlı alt program çağrılmakta ve bu alt program kayıt işlemini yapmaktadır. Eğer ilk sözcük 'SEARCH' ise bundan bir sonraki sözcük tarama kriteri olarak alınmakta ve istenen taramayı veri tabanında yapıp sonuçları kullanıcıya aktaran uygun alt program çağırılmaktadır.

Gelen ilk sözcüğün bu iki sözcük dışında herhangi bir sözcük olması durumunda ise istek anlaşılamadığından hiç bir işlen yapılmamaktadır. Kurulan döngü sonsuza kadar bu şekilde sürüp gitmektedir.

RECORD (Kayıt) Alt Programı

Bu alt programda ilk önce 'FORM NO A' dosyasının varlığı varlığı test edilmektedir. Çünkü ilk kez kayıt yapıldığı anda bu dosya var olmayacaktır. Bunu anlamayı sağlayan CMS komutu ise

'STATE FORM NO A'

komutudur. Eğer bu deyimin getirdiği dönüş kodu değişkeni RC'nin değeri 0 ise dosya var, aksi halde dosya yok demektir. Dosya yoksa form numarası sıfırlanmakta, dosya varsa bu değer dosyadan okunmaktadır. Daha sonra bu sayı 1 arttırılmakta ve elde edilen bu yeni sayı güncel form numarası olarak bir değişkene atanmakta ve 'FORM NO A' dosyasına yazılmaktadır.

Bu aşamada gelen yayın bilgileri IUCV'den çekilir. Bu bilgileri veri tabanında saklamak için CMS'in dosya yönetim imkanlarından büyük ölçüde yararlanılmıştır. Gelen bilgileri dosyaya yazmak için CMS'ın EXECIO komutu kullanılmıştır. EXECIO komutuyla bir dosyaya normalde değişken formatta yazılmaktadır. Bu da disk tasarrufu açısından önemlidir. Çünkü değişken kayıt uzunluklu dosyalar, sabıt formatlı dosyalara göre diskte daha az yer kaplamaktadır.

Değişken formatta bir dosyaya EXECIO ile yazılırken en fazla 255 karakter uzunluğunda bir kayıt eklenebilmektedir. Bu yüzden her bir kayıt için toplam karakter uzunluğu 255'i geçmeyecek şekilde bir yapı kurulmuştur.

Her bir kaydın maximum 255 byte uzunluğunda olabilmesi, bazı bilgilerin birbirine eklenerek saklanmasının disk tasarrufu sağlayacağını göstermiştir. Bunu sağlarken bilgileri birbirinden ayırdetmek için araya satırbaşı karakteri ('15'X) konulmuştur.

SEARCH (Arama) Alt Programı

Kullanıcıdan gelen istek yayın taramaksa bu alt program çağrılmaktadır. Bu yordamla yayın tarama kriterinin gerektirdiği işler yapılmaktadır. Bir örnek olarak "Yazar Soyadı"na göre tarama işlevinin nasıl gerçekleştiğini görelim:

Kullanıcıdan gelen mesajın ilk sözcüğü 'SEARCH', ikinci sözcüğü 'SURNAME' ise SRCH_SUR adlı alt program çağırılmaktadır. Bu alt program gelen mesajın 3. ve son kısmını soyadı argümanı olarak almaktadır. Bundan sonra yayın dosyaları bu soyadı için EXECIO komutunun LOCATE seçeneği kullanılarak taranmaktadır. Eşleşme durumunda ilgili yayın bilgisi kullanıcı makinasına IUCV yoluyla aktarılmaktadır. Eğer hiç bir eşleşme olmazsa kullanıcı görüntü makinasına 'NO_DATA' bilgisi yollanmakta ve kullanıcı programı da bunu uygun şekilde değerlendirmektedir.

Bütün bu işlemlerden sonra ana döngüye geri dönülmekte ve çevrim böylece sürmektedir.

Kullanıcı Programı

Kullanıcı programı veri tabanı görüntü makinasıyla haberleşen ve kullanıcının isteklerinin gerçekleşmesini sağlayan programdır. Programdaki 'SIGNAL ON ERROR' deyimi herhangi bir hata durumunda ERROR adlı etikete dallanmayı sağlamaktadır. Program içindeki herhangi bir hatalı durum karşısında sistem 0'dan farklı bir dönüş kodu vermektedir. ERROR etiketli yerdeki deyimler dönüş kodunu test ederek ilgili işlemleri yapmaktadır. Örneğin veri tabanını sağlayan görüntü makinayla bağlantı kurulamamışsa bu, dönüş kodu ile anlaşılmakta ve operatör'e veri tabanı görüntü makinasını açması için mesaj gönderilmektedir. Bu hata yordamının sonunda da program sonlanmaktadır. Kullanıcı programında veri tabanı makinasını tanımlamayı sağlayan

Dbase_vm = 'KBBYAYIN'

deyimidir. Bu görüntü makinada veri tabanı programı çalışacaktır. Daha sonra

'IUCV CONNECT' dbase_vm

deyimiyle bu görüntü makinayla bağlantı kurulmaktadır. Kullanıcıdan vermesi gereken bilgileri istemek ve yazılan bilgileri okumak için CMS Rel.5'ın getirdiği pencereleme imkanlarından faydalanılmıştır. Programda bu aşamada pencerelerde gösterilecek sabit bilgiler tanımlanmıştır.

Program çalıştırılınca ekranda ilk olarak kullanıcıdan 'Yayın Kaydı' mı yoksa 'Yayın Tarama' mı yapmak istediğini soran bir ana menü belirmektedir.

Eğer yayın kaydı isteği seçilirse ekrana bir ortak yayın kaydı penceresi gelmekte kullanıcıdan yayın tipinden bağımsız olan orta bilgileri girmesi istenmektedir. Bu bilgiler arasında yayın adı, tipi ve dili, yazar ünvanı, adı, soyadı ve kurumu sayılabilir. Yayın dili ve yayın tipi gibi bir takım bilgiler kodlanmıştır. Kullanıcının bu kodlara göre bilgi girmesi gerekmektedir. Seçeneklerde olmayan bir kod girilirse kullanıcı doğru kodu girmesi için uyarılmaktadır.

Bütün girilen bilgiler doğruysa bir sonraki aşamaya geçilmektedir. Bu aşamada TYPE değişkeni

```
Select
        When type = 1 then signal ARTICLE
                .
                .
        When type = 5 then signal REPORT
        Otherwise signal THESIS
End
```

deyimleriyle kontrol edilmekte ve ilgili etikete dallanma sağlanmaktadır. Yayın tipleri Bilimsel Makale, Yayınlamış Bildiri, Kitap, Ders Notu, Yayınlanmış Rapor, Y.Lisans veya Doktora Tezi olarak sınıflandırılmaktadır

Yayın tipinin gerektirdiği etikete dallanıldıktan sonraq ekranda o yayın tipiyle ilgili bilgileri isteyen bir pencere açılmakta ve kullanıcıdan yine bu bilgileri eksiksiz ve hatasız doldurması beklenilmektedir. Bu esnada ilk pencereye, yani ana kayıt sayfasına dönüp burada yazılan bilgileri değiştirmek mümkündür. Bu sayfa da gerektiği şekilde doldurulup ENTER'a basılınca ekrana yine ana menü çıkmaktadır. Bu arada kullanıcının verdiği bilgiler program tarafından okunmakta ve veri tabanı görüntü makinasına uygun şekilde aktarılmaktadır. Bizim örneğimizde ilk yollanan bilgi parçası 'RECORD' olacaktır. Çünkü tarama isteğiyle karışmamalıdır.

Eğer kullanıcıdan gelen istek yayın tarama yönündeyse ekranda yayın taramayla ilgili bir pencere açılmakta ve kullanıcıdan yayın tarama isteği sorulmaktadır. Bu istek kodlarla belirlenmiş durumdadır. Örneğin "1" girilmişse bu, yazar soyadına göre tarama anlamındadır. Bu kodlar ekranda açıklanmıştır.

Yazar soyadına göre tarama istenmişse yazarın soyadı ekrandan sorulmakta ve girilen bilgi <surnamq> değişkenine atanmaktadır. Daha sonra veri tabanı görüntü makinasına sırasıyla 'SEARCH', 'SURNAME', surnamq bilgilerini yollanmaktadır. Bir sonraki aşamada 'IUCV WAIT' komutuyla veri tabanından gelecek bilgiler beklenmektedir. Eğer 'NO_DATA' şeklinde bir bilgi gelirse bu, tarama sonunda böyle bir yazar adıan rastlanmadığı şeklinde yorumlanıp kullanıcı bilgilendirilmektedir. Eğer gelen ilk bilgi 'NO_DATA' değilse sorulan yazarın yayınlarına ait bilgiler olarak değerlendirilmekte ve ilgili alt program çağrılmaktadır.

İlgili alt program gelen kayıttaki bilgileri '15'X karakterlerini ayıklayarak çıkarmakta ve uygun değişkenlere atamaktadır. Eğer gelen kayıt sadece '15'X ise bu gelen yayın bilgisinin sonu anlamına gelmekte ve bundan sonraki veri yeni bir yayın bilgisi olarak alınarak ilgili değerlendirme alt programı çağrılmaktadır.

CHAPTER 1
INTRODUCTION

1.1 Parallelism

Many of today's advanced research problems need greater computing power at high speeds. Examples of this kind include artificial intelligence, robotics, signal processing, fluid mechanics, weather forecasting, high energy physics, molecular physics and space sciences.

A simple-minded approach to gain speed, as well as power, in computing is through parallelism; here many computers would work together, all simultaneously executing some portions of a procedure used for solving a problem. Such an approach rests on the following assumptions:

(1) The availability of many low-cost, high speed computers that can be put together to work in unison, as in a concert;

(2) The existence of a strategy to partition a problem into smaller problems, such that most of these can be solved simultaneously, and from which we can easily construct the solution to the entire problem: this is popularly known as the 'divide-and-conquer strategy'.

1.2 Communication

When a large problem is broken into smaller tasks, it is necessary to set up a coordination between these tasks. To do this coordination we need communication links between the different tasks. The larger the number of pieces a problem is split into, the more communication links the resulting algorithm requires. In fact, the number of directed two-way communication links between any two tasks among n tasks could be n*(n-1) and so the communication complexity grows quadratically. The communication problem introduces a new dimension to parallel programming and parallel architecture design. This means that in addition

to the computing time and memory space requirements of an algorithm, we must also consider the communication costs and set a bound on the complexity of communication. This would imply that communication between tasks which are not close enough (closeness being measured by a suitably defined criterion for an architecture or for a program) should be avoided; that is, the communication should preferably be confined to only those process that are very close neighbors.

When a large number of processors are assigned to carry out the split tasks, it is possible that simultaneous request or access to certain data or tasks may create a conflict or collision. This could slow down the anticipated speed advantage resulting from the multiple processors, or may even lead to a state of inactivity or standstill when two processes or processors wait for each other indefinitely (deadlock), or may delay some process indefinitely (lockout). Since the computational speeds for different tasks are unpredictable (non-deterministic), the different processes may become unsynchronized, leading to a total breakdown of the tasks.

The communication problem is therefore concerned with the minimization of communication complexity, prevention of deadlocks and improved coordination.

In recent years, the models and techniques employed to solve the three basic issues, namely decomposability, complexity and communication, have grown into major interdisciplinary science of parallel processing. This science deals with both the theoretical studies and the practical aspects of design to achieve the best results.

1.3 Practical Realization of Parallelism

In the first section, the notion of parallelism was introduced by the informal statement 'many computers are put together to work, all simultaneously executing some portions of a procedure'. This statement is imprecise and requires further elaboration on the different practical ways in which several processors can be coupled together and the nature of the tasks that are handled.

1.4 Nature of the Tasks

There are two major categories in which computational tasks can be classified:

- Mutually independent or non-interacting tasks;
- Mutually dependent and interacting tasks.

In the case of mutually independent tasks the input consists of many non-interacting or disjoint sets of data; on this data a common (or a different) set of operations is to be performed. To carry out these operations each processors works independently by dividing the problem suitably. The following are examples:

(1) elementwise addition of several arrays;

(2)
```
A := B + C
D := E - F
G := B * H + F
```

When these tasks are mutually dependent, different tasks may be simultaneously executable, sometimes on common data. However, it is possible that a succeeding task is dependent on the outputs generated from one or more of the preceding tasks. In such a case there is a need for a mechanism to coordinate the arrival of the inputs with the succeeding tasks from the preceding tasks. For instance, one or more of the several required inputs may arrive at different time intervals, some earlier and some later; then the succeeding tasks have to wait before any action; otherwise, the computed input-output relations would not be the desired relations. To coordinate such a dynamic or time-varying situation, a synchronization mechanism of some kind has to be introduced. Such a mechanism should have the following properties.

(1) The ability to detect actions performed by the interacting processes;

(2) The ability to permit the data transfer at the appropriate time.

These two properties require that the communication between dependent processes and their synchronization go hand in hand and are, in general, inseparable.

The design of a parallel machine with many processors and the design of a program with many processes are critically dependent on the manner in which synchronization and communication take place. Several methods are available for the design of parallel machine and for the design of parallel languages. The methods are essentially dependent on the methods used for coupling the processors, the nature of the tasks and the types of the parallelism involved.

1.5 Multiprogramming (Scheduling)

When two or more programs share the resources of a single CPU, we call in multiprogramming. When two or more processors are involved it is called multiprocessing. Thus multiprocessing is a hardware concept while multiprogramming is a software concept. Multiprogramming increases the CPU utilization by always having something for the CPU to execute by properly scheduling the jobs or suitably interleaving them. Essentially such a concept evolved for the design of multiuser operating systems. Thus multiprogramming handles a queue of jobs and schedules them to maximize the performance of the computer system.

When several processors are loosely coupled in a network, it is obviously necessary to improve the performance and to balance the load in each processor by properly routing the jobs to the various processors. Multiprogramming techniques are therefore useful for the design and optimization of systems that work in parallel.

1.6 Synchronization Methods

When a set of processes are non-sequential, the processes may have several simultaneous inputs or outputs and they may need to pass data to one another or to communicate. Since the speed at which each process works is not determinable, non-determinism arises in linking the processes. For instance, even if several processes begin together they may not all end simultaneously. Therefore we say there is a race condition and the parts of computation containing such processes are time critical. A need for therefore arises to introduce a mechanism to coordinate and control the temporal order in which processes are executed to realize a given non-

sequential algorithm. Such a mechanism is called a synchronization mechanism.

The synchronization mechanism is essentially meant for handling the non-determinism that arises in the execution of different process by providing choice and delay among the processes to coordinate them.

Two distinctly different methods are used for synchronization among processes.

(1) **Shared-Variable Method**

In this method, synchronization and communication among the processes are achieved using shared mechanisms under a centralized control.

(2) **Message Passing Method**

In this method the processes are autonomously controlled in sending and receiving messages, without sharing data; they are coordinated by using delay and wait operations among them.

These two methods have contributed to the development of new styles of programming languages for concurrent programming.

The design of methods for the description of parallel processes and the synchronization and protection mechanisms has been a very active area of research during recent years. As a result of these studies, new concepts and analytic models have been developed for the programming, complexity and analysis of parallel and concurrent process.

1.7 Message-Passing Synchronization (Handshaking) Mode

The message-passing mode of synchronization provides a greater flexibility for a large number of interacting processes - called distributed processes. This mode of synchronization is also known as handshaking since it involves the exchange of information.

The basic operations needed for message passing are:

(1) Send (message);

(2) Receive message.

For any two processes to communicate using these operations the following the following basic issues are to be considered:

(1) Naming of processes so that the processes know each other's identity.

(2) Size of the message - fixed or variable?

(3) Capacity of links - is there a buffer storage?

(4) Symmetric or asymmetric - can two processes communicate with each other symmetrically or only one way?

(5) Introduction of non-determinism: to deal with a set of events whose order cannot be predicted in advance.

(6) Termination: although each process communicates with others for a finite unbounded number of times, the program terminates.

(7) Deadlock absence: no two processes should wait for each other for communication.

(8) Chattering free: two processes do not communicate infinitely, ignoring other processes.

(9) Fairness - there are three types of fairness:

- process fairness: each process will infinitely often communicate with others and conversely;
- channel fairness: each pair of processes is mutually willing (symmetric) to communicate;
- communication fairness: for specific communication between two processes there is no restriction.

(10) Starvation free: no process is locked out as a result of the absence of any of the three possibilities described in (9). In particular, we must ensure that several of the processes do not conspire against some of the other processes do not conspire against some of the other processes to lock them out.

When two concurrent processes want to communicate they must go through two basic steps:

(1) Synchronization: Each process must contain in its program a statement that expresses its intention to exchange information with another process. If one of the processes arrives at its statement of intention earlier and the other arrives at its statement of intention later, the former has to wait until the latter reaches its proper point to synchronize.

(2) Communication: Once the processes are synchronized as above, they exchange messages.[1]

CHAPTER 2
INTER-USER COMMUNICATIONS VEHICLE (IUCV)

2.1 General Description

The Inter User Communication Vehicle (IUCV) is a communications facility that allows a program running in a virtual machine to communicate with other virtual machines, with a CP system service, and with itself.

An IUCV communication takes place between a source communicator and a target communicator. The communication takes place over a predefined linkage called a path. Each communicator can have multiple paths, and can receive or send multiple messages on the same paths simultaneously.

IUCV provides functions, through the IUCV macro instruction, to

- Create and dismantle paths
- Send and reply to messages
- Receive or reject messages
- Control the sequence of IUCV events.

Communicators receive information about IUCV events by handling IUCV external interrupts.

2.2 IUCV Paths

The IUCV directory control statement authorizes the establishment of paths between virtual machines, or between a virtual machine and a CP system service. If the maximum number of paths is not specified by the MAXCONN keyword of the OPTION statement in the user's directory, a communicator can establish a maximum of four paths.

Once authorized, users establish a path when the source communicator invokes the CONNECT function and the target communicator invokes the ACCEPT function. Either communicator can terminate an established path via the sever function. The target communicator can also prevent the establishment of a path bu invoking the SEVER function instead of the ACCEPT function. In addition, communication over a path can be temporarily suspended when a communicator invokes the QUIESCE function. The quiesced path can be reactivated when a communicator invokes the RESUME function.

A single communicator can have multiple paths defined, and virtual machines may have multiple paths between them. The communicator could be a source communicator on some of its defined paths, a target communicator on other paths, and both a source and a target communicator on still other paths. Communication over any and all paths can occur simultaneously.

Every path has two ends: the source communicator's end and the target communicator's end. The source communicator has a description of the path from the source's perspective and the target communicator has a description of the same path from the target's perspective.

Each path description has a path id that is unique for each communicator. IUCV assigns path ids when communicators invoke the CONNECT and ACCEPT functions. When invoking IUCV functions, the source communicator identifies the path by using the source's path id. The target communicator identifies the same path to IUCV by using the target's path id. A pathid is IUCV's method of distinguishing among the paths available to a communicator.

2.3 IUCV Messages

An IUCV communication is called a message. The source communicator invoking the SEND function initiates communication and creates a message. The target communicator obtains the message by invoking the RECEIVE function.

The target communicator can optionally request information about messages sent to it by invoking the DESCRIBE function, and can refuse a message sent to it by invoking the REJECT function. The target communicator can respond to a message via the REPLY function.

Communication is terminated and the message is destroyed when the source communicator issues the TEST COMPLETION function or handles an IUCV message complete external interrupt.

2.3.1 Message Data Transfer

When the target communicator issues the RECEIVE function, IUCV moves the message data from the source communicator's SEND virtual address space to the target communicator's RECEIVE virtual address space. When the target communicator issues the REPLY function during a two-way communication, IUCV moves data from the target communicator's REPLY virtual address space to the source communicator's ANSWER virtual address space.

Figure 2.1 illustrates the movement of message data during an IUCV two-way communication.

The source communicator's SEND and ANSWER areas may overlap. Similarly, the target communicator's RECEIVE and REPLY areas may overlap.

CP (Control Program) performs storage protection checking for all data moved during an IUCV communication.

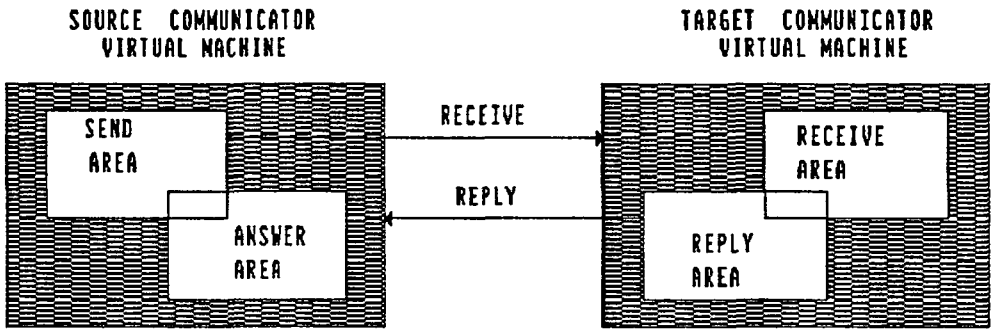

Figure 2.1 IUCV Two-Way Data Transfer

2.3.2 Message Identification

A message is fully identified to a virtual machine by three values. IUCV functions allow one or more these values to be specified to selectively process messages.

- **Message id**

IUCV assigns a message id when the source communicator invokes the SEND function. The message id is generated by a sequential counter value and is unique for the system IPL.

- **Message Class**

The source communicator identifies a message by using the source message class and target communicator identifies a message by using the target message class. The message classes are arbitrary values that the source communicator specifies when invoking the SEND function. The meaning of message classes is agreed to in advance by the two communicators. IUCV places no restrictions on the values specified for

message class. The communicators can use the message class to handle messages selectively.

- **Path id**

IUCV assigns the path id when a path is established with the CONNECT function.

There is no defined relationship between the values of the source and target path ids IUCV assigns, or between the message classes the source and the target communicators use. None of these values need to be the same although they refer to the same message.

The message id always has the same value for both target and source communicators.

When invoking IUCV functions, the source communicator may refer to a message by a combination of its source path id, source message class, and message id. The target communicator may refer to the same message by a combination of its target path id, target message class, and message id.

The message tag information may optionally be used by the source communicator to further identify a message. Since IUCV presents the tag to the source communicator when the message completes, the tag may be used to tie the completed message to the original SEND request.

Since a message can be identified as a priority message, the source communicator may also use this as an indication to the target communicator that special handling is required. IUCV queues a priority message ahead of any nonpriority messages and behind any earlier priority messages. A communicator must be authorized to handle priority messages in the IUCV directory control statement.[2]

2.4 IUCV Module:

The 'IUCV' command is used to manage IUCV communications with other virtual machines or with CP services. The format of the 'IUCV' command is:

```
IUCV LOAD
     Connect
     Sever   userid
     Quiesce userid
     Resume  userid
     Send    userid text
     Sendp   userid text
     Wait
     Grab
     Type
     Stem    stemname
```

where:

LOAD

causes 'IUCV' to load itself as a nucleus extension with the SYSTEM and SERVICE attributes. 'IUCV' must be loaded before any other functions can be used.

CONNECT userid

Sets up an IUCV connection to the virtual machine or CP system service that is specified by userid. If the other virtual machine accepts connections that allow priority IUCV messages, then the connection will be authorized for priority IUCV messages.

SEVER userid

Severs an existing IUCV connection to the virtual machine or CP system service that is specified by userid.

QUIESCE userid

Quiesces the IUCV connection to the virtual machine or CP system service that is specified by userid. When you quiesce an IUCV connection, you may send messages to the other user, but they may not

send messages to you. The other user will receive an IUCV interrupt that indicates you have quiesced the connection.

RESUME userid

Resumes a previously quiesced IUCV connection to the virtual machine or CP system service that is specified by userid. When you resume the IUCV connection, the other user will again be able to send messages to you via IUCV. The other user will receive an IUCV interrupt that indicates you have resumed the connection.

SEND userid text

Sends a non-priority IUCV message to the virtual machine or CP system service specified by userid. The message consists of the value of text. The message cannot be a null string, but may consist entirely of blanks.

SENDP userid text

Is similar to the SEND function, except that the message is sent as a priority IUCV message. SENDP will only work if the other virtual machine has allowed you to send priority IUCV messages to it.

WAIT

Waits for an IUCV message to arrive from another virtual machine or from a CP system service. If there are already messages waiting then the WAIT function will return immediately.

GRAB

Puts all the messages that have arrived since the last GRAB, TYPE or STEM function on the console stack. See Note 1

TYPE

Types all the messages that have arrived since the last GRAB, TYPE or STEM function on the console. See Note 1.

STEM stemname

Places all the messages that have arrived since the last GRAB, TYPE or STEM function into REXX or EXEC2 variables. See Notes 1 and 2.

2.4.1 General Information

The 'IUCV LOAD' function must be issued before any of the other 'IUCV' functions can be used. After it has been issued the first time, the 'IUCV LOAD' function will generate a return code of 1, which indicates that 'IUCV' is already loaded. Virtual machines must be authorized in their CP directory entries for IUCV transactions in order for a authorized connection to be made.

The 'IUCV' command manages IUCV connections via the CMSIUCV service of CMS. It will not conflict with other programs that use CMSIUCV except under limited circumstances. If two programs are both trying to communicate with the same CP system service, then whichever program establishes a connection first will be the one that is able to communicate.

The 'IUCV CONNECT' function specifies the name 'IUCV' in the IPUSER field of the CONNECT parameter list. If the other virtual machine is also using CMSIUCV, this will establish a connection to the program in the other virtual machine that has identified itself to CMSIUCV as 'IUCV'.

When another virtual machine wishes to connect to yours, it must specify the name 'IUCV' in the IPUSER field when it issues the CONNECT function. Then CMSIUCV in your virtual machine will pass the CONNECT PENDING interrupt to the 'IUCV' command. The 'IUCV' command will then accept the connection.

When a connection is initiated by another virtual machine, you may communicate with the other virtual machine the same as if you had issued an 'IUCV CONNECT' to that virtual machine.

When the other virtual machine desires a connection that is enabled for priority messages, the 'IUCV' command will allow this if possible. One of the two virtual machines must be authorized in its CP directory entry for priority IUCV transactions in order for a priority authorized connection to be made. If neither virtual machine is authorized, the IUCV connection will still be made, but it will not be able to transfer priority messages.

The error "Invalid path id specified" can only occur if some other program has severed the path that the 'IUCV' command established. This could occur if a non-CMSIUCV program issued a SEVER ALL.

Programs that do not use the CMSIUCV service can cause problems for programs that do when both are being used in the same virtual machine. If the program that does not use CMSIUCV usurps the IUCV interrupt buffer from CMS, then all CMS IUCV communications are disabled. Furthermore, CMS will abend with a program exception when it attempts to clean up its own IUCV management. For these reasons, use of such programs should be avoided if other IUCV programs are in use.

2.4.2 Notes

1. The GRAB, TYPE and STEM functions are used to retrieve incoming IUCV messages. These functions output the messages by different methods, but all of them format the messages in the same way. The format of the information produced is as follows:

```
columns          1-8          9
                 userid   message text
```

All of the messages received from all of the existing IUCV connections are output. The messages are grouped by the userid or CP system service that they came from.

The GRAB function is limited to a total length of 256 characters of information per line. The TYPE function is limited to 1760 characters. The STEM function has no limit other than the storage available in the virtual machine.

2. The STEM function places messages in REXX or EXEC2 variables. The specified stem is concatenated with numeric values to make a series of indexed variables. For example, if there are 5 messages waiting:

```
IUCV STEM OUTPUT.
```

will set the following values:

```
OUTPUT.0 = 5
OUTPUT.1 = first message
.
.
.
OUTPUT.5 = fifth message
```

When you specify the stem name on the command, the case of the stem name is preserved. This is to say, the stem name is not translated to upper case before it is used. Since REXX stem names are upper case, you must specify the stem name in upper case if it is a simple stem like the example above. There is no limit imposed on the length of the stem name. More complex stem assignments are possible. You can use multiple variable substitutions in constructing the stem name so that it contains lower case characters. For example:

```
IUCV STEM OUTPUT.lower.
```

will set the variables:

```
OUTPUT.lower.0

OUTPUT.lower.n          [3]
```

CHAPTER 3
REXX LANGUAGE

3.1 Introduction

Most home computers use a language called BASIC that has very few rules. However, if you want to write an intricate or involved program in BASIC, you will find yourself writing great many lines. Other languages like PL/1, APL, and Pascal, which are much more "powerful," allow you to do more in fewer lines, even though these languages have more rules. The Restructures Extended Executor language, abbreviated REXX, is a language that fits nicely in between.

When you write REXX program, you can include commands to CP (Control Program) and CMS (Conversational Monitoring System) in the "list of things to be done." You write these just as you would if you were typing them on the CMS command line. REXX is preferred to the older exec languages for nearly all programs that need to issue CMS commands.

On the other hand, REXX somewhat resembles PL/1. There are a number of differences, but the main difference is that a REXX program is "interpreted" (the interpreter operates on the program directly as it executes). In PL/1 the program is compiled (translated into machine language) first, then executed. So, although REXX is easier to use, it does use more computer time.

3.2 Rules

The System Product Interpreter works on your REXX program, line by line and word by word, doing what you have written. The rules of REXX are quite simple.

To indicate comments in REXX , use:

/* to mark the start of a comment

*/ to mark the end of a comment

The first line of a REXX program must start with a comment. Why? Historically, there are three languages that can be used for writing EXECs for VM/SP. The oldest is called CMS exec; the next is EXEC 2; and the latest is REXX. For technical reasons, they all have a filetype of EXEC. Because each type of EXEC requires its own special processing, CMS must be able to distinguish one type from another. It does this by looking at the first line of the EXEC file. So, to tell CMS that your program is written in REXX, the first line of the file must be a comment.

```
/* This is a REXX program. */
```

Strings:

When the interpreter sees a quote (either " or ') it stops interpreting and just goes along looking for the matching quote. The string of characters inside the quotes is used just as it is. Example:

```
'Final Result:'
```

If you want to use a quotation mark within a string you should use a pair of quotes in it.

Clauses:

Your REXX program consists of a number of clauses. A clause can be:

1. An instruction that tells the interpreter to do something; like,

```
say 'the word'
```

In this case, the interpreter will display the word on the user's screen.

2. An assignment; like,

```
Message = "Take care!"
```

3. A label, which is a name followed by a colon; like,

MYSUB:

4. A null clause, such as a completely blank line, or

;

Anything that is not one of these is taken to be:

5. A command; like,

erase hello exec

Commands are passed to CMS (or other environments.)

When Does a Clause End?

It is sometimes useful to be able to write more than one clause on a line, or to extend a clause over many lines. The rules are:

* Normally, each clause occupies one line.
* If you want to put more than one clause on a line you must use a semicolon (;) to separate the clauses.
* If you want to a clause to span more than one line you must put a comma (,) at the end of the line to indicate that the clause continues on the next line

Tidying Up

When a line is being interpreted, everything that is not in quotes is translated to uppercase. In other words the letters:

a,b,c,...z

get changed to

A,B,C,...Z

3.3 REXX Commands and Functions

ADDRESS environment (expression) ;

where, environment is a single symbol or string, which is taken to be a constant. This instruction is used to effect a temporary or permanent change to the destination of command(s).

To send a single command to a specified environment, an environment name followed by an expression is given. expression is evaluated, and the resulting command string is routed to environment. After execution of the command, environment will be set back to whatever it was before, thus giving a temporary change of destination for a single command. Example:

```
Address CMS 'STATE PROFILE EXEC'
```

If only environment is specified, a lasting change of destination occurs: all following commands (expressions not preceded by a REXX keyword) will be routed to the given command environment, until the next ADDRESS instruction is executed. The previously selected environment is saved. Example:

```
address CMS
'STATE PROFILE EXEC'
if rc=0 then 'COPY PROFILE EXEC A TEMP = ='
address XEDIT
```

ARG variablename

ARG instruction assigns information you enter after the exec name to a variable.

EXAMPLE: Your exec named PRT contains this arg instruction:

```
arg num
```

If you enter PRT 2, the variable 'num' is set to 2.

CALL name (expression)(,(expression))...;

CALL is used to invoke a routine. The routine may be an internal routine, an external routine or program, or a built-in function. Name, given in the CALL instruction, must be a valid symbol, which is treated literally, or a string. If a string is used for name (that is, name is specified in quotes) the search for internal labels is bypassed, and only a built-in function or an external function will be invoked. Note that the names of built-in functions (and generally the names of external routines too) are in uppercase, and hence the name in the literal string should be in uppercase. The invoked routine may optionally return a result upon its completion, which is functionally identical to the clause:

result=name((expression)(,(expression))...);

where the variable RESULT will become uninitialized if no result is returned by the routine invoked. Up to ten expressions, separated by commas, may be specified. These are evaluated in order from left to right, and form the argument string(s) during execution of the routine. Any ARG or PARSE ARG instructions, or ARG built-in function in the called routine will access these strings, rather than those previously active in the calling program. Expressions may be omitted if desired. The CALL then causes a branch to the routine called name using exactly the same mechanism as function calls. The order in which these are searched for is described in the section on functions but briefly is as follows:

Internal routines: (unless the routine name is specified in quotes) These are sequences of instructions inside the same program, starting at the label that matches name in the CALL instruction.

Built-in routines: These are routines built in to the interpreter for providing various functions. They always return a string containing the result of the function.

External routines: Users may write or make use of routines that are external to the interpreter and the calling program. An external routine may be written in any language, including REXX, which supports the system dependent interfaces used by the interpreter. A REXX program

may be invoked as a subroutine by the CALL instruction, and in this case may be passed more than one argument string. These may be retrieved using the ARG or PARSE ARG instructions, or the ARG built-in function.

During execution of an internal routine, all variables previously known are normally accessible. However, the PROCEDURE instruction may be used to set up a local variables environment to protect the subroutine and caller from each other. The EXPOSE option on the PROCEDURE instruction may further be used to expose selected variables to a routine. Calling an external program as a subroutine is similar to calling an internal routine. The external routine is however an implicit PROCEDURE in that all the caller's variables are always hidden, and the status of internal values (NUMERIC settings, etc.) start with their defaults (rather than inheriting those of the caller).

When control reaches the internal routine, the line number of the CALL instruction is available in the variable SIGL (in the caller's variable environment). This may be used as a debug aid, as it is therefore possible to find out how control reached a routine.

Eventually the subroutine should execute a RETURN instruction, and at that point control will return to the clause following the original CALL. If the RETURN instruction specified an expression, the variable RESULT will be set to the value of that expression. Otherwise, the variable RESULT is dropped (becomes uninitialized).

Internal routines may include calls to other internal routines, including itself. Example:

```
 /* Recursive subroutine execution... */
arg x
call factorial x
say x'! =' result
exit

factorial: procedure      /* calculate factorial by.. */
  arg n                    /* .. recursive invocation. */
  if n=0 then return 1
  call factorial n-1
  return  result * n
```

Implementation maximum: The total nesting of control structures, which includes internal routine calls, may not exceed a depth of 250.

CENTER(string,length(,pad)) or CENTRE(string,length(,pad))

returns a string of length length with string centered in it, with pad characters added as necessary to make up length. The default pad character is blank. If the string is longer than length, it will be truncated at both ends to fit. If an odd number of characters are truncated or added, the right hand end loses or gains one more character than the left hand end. Here are some examples:

```
CENTER(abc,7)              ->    ' ABC '
CENTER(abc,8,'-')          ->   '--ABC---'
CENTRE('The blue sky',8)   ->   'e blue s'
CENTRE('The blue sky',7)   ->   'e blue '
```

Note: This function may be called either CENTRE or CENTER, which avoids errors due to the difference between the British and American spellings.

COMPARE(string1,string2(,pad))

returns 0 if the strings, string1 and string2, are identical. If they are not, the returned number is non-zero and is the position of the

first character that does not match. The shorter string is padded on the right with pad if necessary. The default pad character is a blank. Here are some examples:

```
COMPARE('abc','abc')          ->   0
COMPARE('abc','ak')           ->   2
COMPARE('ab ','ab')           ->   0
COMPARE('ab ','ab',' ')       ->   0
COMPARE('ab ','ab','x')       ->   3
COMPARE('ab-- ','ab','-')     ->   5
```

COPIES(string,n)

The COPIES function produces 'n' concatenated copies of a string.

EXAMPLE: say copies ('crowded',2) displays: "crowdedcrowded"

say copies ('testing ',1) displays: "testing "
say copies ('2 spaces ',2) displays: "2 spaces 2 spaces "

C2D(string(,n))

Character to Decimal. Returns the decimal value of the binary representation of string. Here are some examples:

C2D('09'x)	->	9
C2D('81'x)	->	129
C2D('a')	->	129
C2D('FF81'x)	->	65409

C2X(string)

Character to Hexadecimal. Converts a character string to its hexadecimal representation (unpacks). The data to be unpacked may be of any length. Here are some examples:

C2X('72s')	->	'F7F2A2'
C2X('0123'x)	->	'0123'

DATATYPE(data)

The DATATYPE function determines whether data is numeric or alphabetic and returns a result of NUM or CHAR.

EXAMPLE: The variable A = 'ABCD'. To find out if variable A is numeric or alphabetic, include in your program: datatype(A)

Here are some examples:

DATATYPE(' 12 ')	->	NUM
DATATYPE('')	->	CHAR
DATATYPE('123*')	->	CHAR

DELSTR(string,n(,length))

deletes the substring of string that begins at the nth character, and is of length length. If length is not specified, the rest of string is deleted. If n is greater than the length of string, the string is returned unchanged. n must be a positive whole number. Here are some examples:

```
DELSTR('abcd',3)       ->    'ab'
DELSTR('abcde',3,2)    ->    'abe'
DELSTR('abcde',6)      ->    'abcde'
```

DELWORD(string,n(,length))

deletes the substring of string that starts at the nth word. The length option refers to the number of blank-delimited words. If length is omitted, it defaults to be the remaining words in string. n must be a positive whole number. If n is greater than the number of words in string, string is returned unchanged. The string deleted includes any blanks following the final word involved. Here are some examples:

```
DELWORD('Now is the  time',2,2)  ->  'Now time'
DELWORD('Now is the time ',3)    ->  'Now is '
DELWORD('Now is the  time',5)    ->  'Now is the  time'
```

DO

```
DO   name=expri (TO exprt) (BY exprb)     WHILE exprw  ;
               (FOR exprf)               UNTIL expru
     FOREVER                             +           +
     exprr
   +                                  +

   +             +
     instruction
          .
          .
          .
END (symbol);
-------------------
```

Or, to present the instruction more generally:

```
DO (repetitor) (conditional);
     instruction
          .
          .
END (symbol);
```

Where, repetitor is one of:

```
name = expri (TO exprt) (BY exprb) (FOR exprf)
  FOREVER
  exprr
```

conditional is one of:

```
    WHILE exprw
    UNTIL expru
```

DO is used to group instructions together and optionally to execute them repetitively. During repetitive execution, a control variable (name) may be stepped through some range of values.

DROP name (name) (name)...;

Where, name is a symbol, separated from any other names by one or more blanks.

DROP is used to "unassign" variables; that is, to restore them to their original uninitialized state. Each variable specified will be dropped from the list of known variables. The variables are dropped in sequence from left to right. It is not an error to specify a name more than once, or to DROP a variable that is not known. If an EXPOSEd variable is named (see the PROCEDURE instruction), the variable itself in the older generation will be dropped. Example:

```
 j=4
Drop  a x.3 x.j
/* would reset the variables: "A", "X.3", and "X.4" */
/* so that reference to them returns their name.     */
```

If a stem is specified (that is, a symbol that contains only one period, as the last character), all variables starting with that stem are dropped. Example:

```
Drop  x.
/* would reset all with names starting with "X." */
```

D2C(whole-number(,n))

Decimal to Character. Returns a character string of length as needed, which is the binary representation of the decimal number. Length may also be specified by n. Here are some examples:

```
D2C(9)        ->    '09'x
D2C(129)      ->    '81'x
```

D2X(whole-number(,n))

Decimal to Hexadecimal. Returns a string of hexadecimal characters of length as needed, which is the hexadecimal (unpacked) representation of the decimal number. whole-number must be a non-negative number unless n is specified, or an error will result. If n is not specified, the result is returned such that there are no leading 0 characters. If n is specified, it is the length of the final result in characters; that is, after conversion the input string will be sign-extended to the required length. If the number is too big to fit into n characters, it will be truncated on the left. Here are some examples:

```
D2X(9)        ->    '9'
D2X(129)      ->    '81'
D2X(129,1)    ->    '1'
D2X(129,2)    ->    '81'
D2X(129,4)    ->    '0081'
D2X(257,2)    ->    '01'
D2X(-127,2)   ->    '81'
D2X(-127,4)   ->    'FF81'
```

EXIT (expression);

EXIT is used to unconditionally leave a program, and optionally return a data string to the caller. The program is terminated immediately, even if an internal routine is currently being executed. If no internal routine is active, RETURN and EXIT have the same function. If expression is given, it is evaluated and the string resulting from the evaluation is then passed back to the caller when the program terminates.

Example:

```
j=3
Exit j*4
/* Would exit with the string '12' */
```

If expression is not given, no data is passed back to the caller. If the program was called as an external function, this will be detected as an error - either immediately (if RETURN was used), or on return to the caller (if EXIT was used).

"Running off the end" of the program is always equivalent to the instruction EXIT, in that it terminates the whole program and returns no result string.

Note: The interpreter does not distinguish between invocation as a command on the one hand, and invocation as a subroutine or function on the other. If in fact the program was invoked via the more primitive command interface (which only allows a numeric return code), an attempt is made to convert the returned value to a return code acceptable by the host. The returned string must then be a whole number whose value will fit in a S/370 register (that is, must be in the range -2**31 through 2**31-1). If the conversion fails, it is deemed to be a failure of the host interface and is thus not subject to trapping by SIGNAL ON SYNTAX. Note also that only the last five digits of the return code (four digits for a negative return code) will be displayed by the standard CMS ready message.

EXTERNALS( )

returns the number of elements in the terminal input buffer (system external event queue), that is, the number of logical typed-ahead lines, if any. Here is an example:

```
EXTERNALS()    ->    0    /* Usually */
```

FIND(string,phrase)

searches string for the first occurrence of the sequence of blank-delimited words phrase, and returns the word number of the first word of

phrase in string. Multiple blanks between words are treated as a single blank for the comparison. Returns 0 if phrase is not found. Here are some examples:

```
FIND('now is the time','is the time')    ->    2
FIND('now is  the time','is     the')    ->    2
FIND('now is  the time','is  time ')     ->    0
```

IF

```
IF expression(;) THEN(;) instruction
                 (ELSE(;) instruction)
```

The IF construct is used to conditionally execute an instruction or group of instructions - depending on the evaluation of the expression. The instruction after the THEN is executed only if the result of the evaluation was 1. If an ELSE was given, the instruction after the ELSE is executed only if the result of the evaluation was 0. Example:

```
if answer='YES' then say 'OK!'
                else say 'Why not?'
```

Remember that if the ELSE clause is on the same line as the last clause of the THEN part, you need a semicolon to terminate that clause. Example:

```
if answer='YES' then say 'OK!'; else say 'Why not?'
```

The ELSE binds to the nearest IF at the same level. Example:

```
if answer='YES' then if name='FRED' then say 'OK, Fred.'
                                    else nop
                else say 'Why not?'
```

Notes:

1. instruction includes all the more complex constructs such as DO groups and SELECT groups, as well as the simpler ones and the IF instruction itself. A null clause is not an instruction; so putting an extra semicolon after the THEN or ELSE is not equivalent to putting a dummy instruction (as it would be in PL/I). The NOP instruction is provided for this purpose.

2. A variable called THEN cannot be used within expression, because the keyword THEN is treated differently, in that it need not start a clause. This allows the expression on the IF clause to be terminated by the THEN, without a ";" being required - were this not so, people used to other computer languages would experience considerable difficulties.

INDEX(haystack,needle(,start))

returns the character position of one string, needle, in another, haystack (see also the POS function). If the string needle is not found, 0 is returned. By default the search starts at the first character of haystack (start is of the value 1). This can be overridden by giving a different start point, which must be a positive whole number. Here are some examples:

```
INDEX('abcdef','cd')      ->   3
INDEX('abcdef','xd')      ->   0
INDEX('abcdef','bc',3)    ->   0
INDEX('abcabc','bc',3)    ->   5
INDEX('abcabc','bc',6)    ->   0
```

INSERT(new,target(,(n)(,(length)(,pad))))

inserts the string new, padded to length length, into the string target after the nth character. length and n must be non-negative. If n is greater than the length of the target string, padding is added there also. The default pad character is a blank. The default value for n is 0, which means insert before the beginning of the string. Here are some examples:

```
INSERT(' ','abcdef',3)          ->   'abc def'
INSERT('123','abc',5,6)         ->   'abc  123  '
INSERT('123','abc',5,6,'+')     ->   'abc++123+++'
INSERT('123','abc')             ->   '123abc'
INSERT('123','abc',,5,'-')      ->   '123--abc'
```

ITERATE (name);

ITERATE alters the flow within a repetitive DO loop (that is, any DO construct other than that with a simple DO).

Execution of the group of instructions stops, and control is passed to the DO instruction just as though the bottom of the group of instructions had been reached. The UNTIL expression (if any) is tested, the control variable (if any) is incremented and tested, and the WHILE expression (if any) is tested. If these tests indicate that conditions of the loop have not yet been satisfied, the group of instructions is executed again (iterated), beginning at the top. If name is not specified, ITERATE will step the innermost active repetitive loop. If name is specified, it must be the name of the control variable of a currently active loop (which may be the innermost), and this is the loop that is stepped. Any active loops inside the one selected for iteration are terminated (as though by a LEAVE instruction). Example:

```
do i=1 to 4
 if i=2 then iterate
 say i
 end
/* Would display the numbers:   1, 3, 4  */
```

LEAVE (name);

LEAVE causes immediate exit from one or more repetitive DO loops (that is, any DO construct other than that with a simple DO). Execution of the group of instructions is terminated, and control is passed to the instruction following the END clause, just as though the END clause had been encountered and the termination condition had been met normally. However, on exit, the control variable (if any) will contain the value it had when the LEAVE instruction was executed. If name is not specified, LEAVE will terminate the innermost active repetitive loop. If name is specified, it must be the name of the control variable of a currently active loop (which may be the innermost), and that loop (and any active loops inside it) is then terminated. Control then passes to the clause following the END that matches the DO clause of the selected loop.

Example:

```
do i=1 to 5
 say i
 if i=3 then leave
 end
/* Would display the numbers:   1, 2, 3  */
```

Notes:

1. name, if specified, must match that on the DO instruction. No substitution for compound variables is carried out when the comparison is made.

2. A loop is active if it is currently being executed. If a subroutine is called (or an INTERPRET instruction is executed) during execution of a loop, the loop becomes inactive until the subroutine has returned or the INTERPRET instruction has completed. LEAVE cannot be used to terminate an inactive loop.

3. If more than one active loop uses the same control variable, the innermost will be the one selected by the LEAVE.

LEFT(string,length(,pad))

returns a string of length length containing the left-most length characters of string. The string returned is padded with pad characters (or truncated) on the right as needed. The default pad character is a blank. length must be non-negative. The LEFT function is exactly equivalent to SUBSTR(string,1,length(,pad)). Here are some examples:

```
LEFT('abc d',8)        ->    'abc d   '
LEFT('abc d',8,'.')    ->    'abc d...'
LEFT('abc  def',7)     ->    'abc  de'
```

LENGTH(string)

returns the length of string. Here are some examples:

```
LENGTH('abcdefgh')    ->    8
LENGTH('abc defg')    ->    8
LENGTH('')            ->    0
```

LINESIZE()

returns the current terminal line width (the point at which the interpreter will break lines displayed using the SAY instruction). If this is indeterminate, 0 will be returned.

Note: This is the terminal width as set by the CP TERM LINESIZE command (but is limited to the CMS maximum of 130); 0 implies that the virtual machine is DISCONNected or that CP TERMINAL LINESIZE OFF has been issued.

MAX(number(,number)...)

returns the largest number from the list specified, formatted according to the current setting of NUMERIC DIGITS. Up to ten numbers may be specified, although calls to MAX may be nested if more are needed. Here are some examples:

```
MAX(12,6,7,9)                              ->   12
MAX(17.3,19,17.03)                         ->   19
MAX(-7,-3,-4.3)                            ->   -3
MAX(1,2,3,4,5,6,7,8,9,MAX(10,11,12,13))    ->   13
```

MIN(number(,number)...)

returns the smallest number from the list specified, formatted according to the current setting of NUMERIC DIGITS. Up to ten numbers may be specified, although calls to MIN may be nested if more are needed. Here are some examples:

```
MIN(12,6,7,9)          ->   6
MIN(17.3,19,17.03)     ->   17.03
MIN(-7,-3,-4.3)        ->   -7
```

NOP

NOP is a dummy instruction that has no effect. It can be useful as the target of a when or if clause.

Example:

```
Select
   when a=b then nop            /* Do nothing */
   when a>b then say 'A > B'
   otherwise     say 'A < B'
end
```

Note: Putting an extra semicolon instead of the NOP would merely insert a null clause, which would be ignored. The second WHEN clause would be seen as the first instruction expected after the THEN, and hence would be treated as a syntax error. NOP is an instruction thatis a valid target for the THEN clause.

OVERLAY(new,target(,(n)(,(length)(,pad))))

overlays the string target, padded or truncated to length length, with the string new starting at the nth character. If length is specified it must be positive or zero. If n is greater than the length of the target string, padding is added there also. The default pad character is a blank, and the default value for n is 1. n must be greater than 0. Here are some examples:

```
OVERLAY(' ','abcdef',3)          ->    'ab def'
OVERLAY('.','abcdef',3,2)        ->    'ab. ef'
OVERLAY('qq','abcd')             ->    'qqcd'
OVERLAY('qq','abcd',4)           ->    'abcqq'
OVERLAY('123','abc',5,6,'+')     ->    'abc+123+++'
```

PARSE

```
PARSE (UPPER) < ARG                    > (template);
              < EXTERNAL               >
              < NUMERIC                >
              < PULL                   >
              < SOURCE                 >
              < VALUE (expression) WITH >
              < VAR name               >
              < VERSION                >
```

Where, template is a list of symbols separated by blanks and/or "patterns." The PARSE instruction is used to assign data (from various sources) to one or more variables according to some rules. If the UPPER option is specified, the data to be parsed is first translated to uppercase.

Otherwise, no uppercase translation takes place during the parsing.If template is not specified, no variables will be set but action will be taken to get the data ready for parsing if necessary. Thus for PARSE EXTERNAL and PARSE PULL, a data string will be removed from the appropriate queue; and for PARSE VALUE, expression will be evaluated. The data used for each variant of the PARSE instruction is:

PARSE ARG

The string(s) passed to the program, subroutine, or function as the input parameter list are parsed. (See the ARG instruction for details.)

Note: The argument string(s) to a REXX program or internal routine may also be retrieved or checked by using the ARG built-in function.

PARSE EXTERNAL

The next string from the terminal input buffer (system external event queue) is parsed. This queue may contain data that is the result of external asynchronous events - such as user console input, or messages. If that queue is empty, a console read results. Note that this mechanism should not be used for "normal" console input, for which PULL is more general, but rather it could be used for special applications (such as debugging) when the program stack cannot be disturbed. The number of lines currently in the queue may be found with the EXTERNALS built-in function.

PARSE PULL

The next string from the program stack (system-provided data queue) is parsed (see note). This queue can save a series of data strings. Data can be added to the beginning or end of the queue using the PUSH and QUEUE instructions respectively. The queue can also be altered by other programs in the system, and can be used as a means of communication between programs. The number of lines currently in the queue may be found with the QUEUED built-in function.

Note: PULL and PARSE PULL read from the program stack. If that is empty, they read from the terminal input buffer; and if that too is empty, a console read results. (See the PULL instruction for further details.)

PARSE SOURCE

The data parsed describes the source of the program being executed. The source string contains the characters CMS, followed by either COMMAND, FUNCTION, or SUBROUTINE depending on whether the program was invoked as some kind of host command (for example, EXEC or macro), or from a function call in an expression, or via the CALL instruction. These two tokens are followed by the program filename, filetype, and filemode; each separated from the previous token by one or more blanks. (The filetype and filemode may be unknown if the program is being executed from storage, in which case the SOURCE string will have one * for each unknown value.) Following the filemode is the name by which the program was invoked (due to synonyms, this may not be the same as the filename). It may be in mixed case and will be truncated to 8 characters if necessary. (If it cannot be determined, "?" is used as a placeholder). The final word is the initial (default) address for commands. If the interpreter was called from a program that set up a subcommand environment, the filetype is usually the name of the default address for commands. The string parsed might therefore look like this:

CMS COMMAND REXTRY XEDIT * rext XEDIT

PARSE VALUE

expression is evaluated, and the result is the data that is parsed. Note that WITH is a keyword in this context and so cannot be used as a symbol within expression. Thus, for example:

PARSE VALUE time() WITH hours ':' mins ':' secs

will get the current time and split it up into its constituent parts.

PARSE VAR name

The value of the variable specified by name is parsed. name must be a symbol that is valid as a variable name (that is, it may not start with a period or a digit). Note that the variable name may be included in the template, so that for example:

PARSE VAR string word1 string

will remove the first word from string and put it in the variable word1, and

PARSE UPPER VAR string word1 string

will also translate the data from string to uppercase before it is parsed.

POS(needle,haystack(,start))

returns the position of one string, needle, in another, haystack. If the string needle is not found, 0 is returned. By default the search starts at the first character of haystack (that is start is of the value 1). This may be overridden by specifying start (which must be a positive whole number), the point at which to start the search. Here are some examples:

```
POS('day','Saturday')      ->   6
POS('x','abc def ghi')     ->   0
POS(' ','abc def ghi')     ->   4
POS(' ','abc def ghi',5)   ->   8
```

PROCEDURE (EXPOSE name (name) (name)...);

Where, name is a symbol, separated from any other names by one or more blanks. The PROCEDURE instruction may be used within an internal routine (subroutine or function) to protect all the existing variables by making them unknown to the following instructions. On executing a RETURN instruction, the original variables' environment is restored and any variables used in the routine are dropped.

The EXPOSE option modifies this, in that the variables specified by names are exposed, so that any references to them (including setting them and dropping them) refer to the variables' environment owned by the caller. If the EXPOSE option is used, at least one name must be specified. Any variables not specified by name on a PROCEDURE EXPOSE instruction are still protected. Hence, some limited set of the caller's variables can be made accessible, and these variables may be changed (or new variables in this set may be created). All these changes will be visible to the caller upon RETURN from the routine.

The variables are exposed in sequence from left to right. It is not an error to specify a name more than once, or to specify a name that has not been used as a variable by the caller. Example:

```
 /* This is main program */
j=1; x.1='a'
call toft
say j k m        /* would display "1 7 M" */
exit

toft: procedure expose j k x.j
   say j k x.j  /* would display "1 K a"    */
   k=7; m=3      /* note "M" is not exposed */
   return
```

Note that if X.J in the EXPOSE list had been placed before J, the caller's value of J would not have been visible at that time, so X.1 would not have been exposed.

If a stem is declared in names, all possible compound variables whose names begin with that stem are exposed. (A stem is a symbol containing just one period, which is the last character.) Example:

```
 Procedure Expose i j a. b.
/* This exposes "I", "J", and all variables whose */
/* name starts with "A." or "B."                  */
A.1='7'  /* This will set "A.1" in the caller's   */
         /* environment, even if it did not       */
         /* previously exist.                     */
```

Variables may be exposed through several generations of routines, if desired, by ensuring that they are included on all intermediate PROCEDURE instructions. Only one PROCEDURE instruction in each

level of routine call is allowed, all others (and those met outside of internal routines) are in error.

Notes:

1. An internal routine need not include a PROCEDURE instruction, in which case the variables it is manipulating are those "owned" by the caller.

2. It is suggested that the PROCEDURE instruction should be the first instruction executed after the CALL or function invocation - that is, it should be the first instruction following the label. This is not enforced.

PULL (template);

Where, template is a list of symbols separated by blanks and/or "patterns."

PULL is used to read a string from the program stack (system-provided data queue), see note. It is just a short form of the instruction:

PARSE UPPER PULL (template);

The current head-of-queue will be read as one string. If no template is specified, no further action is taken (and the data is thus effectively discarded). Otherwise, the data is translated to uppercase and then parsed into variables according to some rules. Use the PARSE PULL instruction if uppercase translation is not desired.

Note: If the program stack is empty, the terminal input buffer is used. If that too is empty, a console read will occur. Conversely, if you "type-ahead" before an EXEC asks for your input, your input data is added to the end of the terminal input buffer and will be read at the appropriate time. The length of data in the program stack is restricted to 255 characters. The length of data in the terminal input buffer is restricted to 130 characters.

Example:

```
Say 'Do you want to erase the file? Answer Yes or No:'
Pull answer .
if answer='YES' then Erase filename filetype filemode
```

Here the dummy placeholder "." is used on the template so as to isolate the first word entered by the user.

PUSH (expression);

The string resulting from expression will be stacked LIFO - Last In, First Out - onto the most recently created buffer of the program stack (system-provided data queue), see note. If expression is not specified, a null string is stacked.

Note: The length of the data in the program stack is restricted to 255 characters. The program stack contains one buffer initially, but additional buffers may have been created using the CMS command MAKEBUF. Example:

```
a='Fred'
push       /* Puts a null line onto the stack */
push a 2   /* Puts "Fred 2"    onto the stack */
```

The number of lines currently in the queue may be found with the QUEUED built-in function.

QUEUE (expression);

The string resulting from expression will be appended to the most recently created buffer of the program stack (system-provided data queue). That is, it will be stacked FIFO - First In, First Out. Example:

```
a='Toft'
queue a 2  /* Enqueues "Toft 2" */
queue      /* Enqueues a null line behind the last */
```

QUEUED

QUEUED() returns the number of lines remaining in the program stack (system-provided data queue) at the time when the function is invoked.

If no lines are remaining, a PULL or PARSE PULL will read from the terminal input buffer. If there is no terminal input waiting this causes a console read (VM READ). Here is an example:

```
QUEUED()    ->    5    /* Perhaps */
```

RANDOM

RANDOM((min)(,(max)(,seed)))

returns a pseudo-random non-negative whole number in the range min to max inclusive. If only one argument is specified, the range will be from 0 to that number. Otherwise, the default values for min and max are 0 and 999 respectively. A specific seed (which must be a whole number) for the random number may be specified as the third argument if repeatable results are desired. Here are some examples:

```
RANDOM()          ->    305

RANDOM(5,8)       ->    7
```

RETURN (expression);

RETURN is used to return control (and possibly a result) from a REXX program or internal routine to the point of its invocation. If no internal routine (subroutine or function) is active, RETURN is essentially identical to EXIT. If a subroutine is being executed (see the CALL instruction), expression (if any) is evaluated, control passes back to the caller, and the REXX special variable RESULT is set to the value of expression. If expression is not specified, the special variable RESULT is dropped (becomes uninitialized). The various settings saved at the time of the CALL (tracing, addresses, etc.) are also restored.

If a function is being executed, the action taken is identical, except that expression must be specified on the RETURN instruction. The result of expression is then used in the original expression at the point where the function was invoked.

If a PROCEDURE instruction was executed within the routine (subroutine or internal function), all variables of the current generation are dropped (and those of the previous generation are exposed) after expression is evaluated and before the result is used or assigned to RESULT.

REVERSE(string)

returns string, swapped end for end. Here are some examples:

```
REVERSE('ABc.')    ->    '.cBA'
REVERSE('XYZ ')    ->    ' ZYX'
```

RIGHT(string,length(,pad))

returns a string of length length containing the right-most length characters of string. The string returned is padded with pad characters (or truncated) on the left as needed. The default pad character is a blank. length must be non-negative. Here are some examples:

```
RIGHT('abc  d',8)     ->    '  abc  d'
RIGHT('abc def',5)    ->    'c def'
RIGHT('12',5,'0')     ->    '00012'
```

SAY (expression);

The result of evaluating expression is displayed to the user. The result of expression may be of any length. Example:

```
data=100
Say data 'divided by 4 =>' data/4
/* Would display: "100 divided by 4 => 25"  */
```

SELECT

```
SELECT;
      WHEN expression(;) THEN(;) instruction
    +                                        +
      WHEN expression(;) THEN(;) instruction
       "       "          "         "
       "       "          "         "
       "       "          "         "
    +                                        +

    +              +             + +
      OTHERWISE(;)   instruction
                       "
                       "
                       "

    +              +             + +
END;
```

SELECT is used to conditionally execute one of several alternative instructions. Each expression following a WHEN is evaluated in turn and must result in 0 or 1. If the result is 1, the instruction following theÂ THEN (which may be a complex instruction such as IF, DO, or SELECT) is executed and control will then pass to the END. If the result is 0, control will pass to the next WHEN clause.

If none of the WHEN expressions evaluate to 1, control will pass to the instruction(s), if any, following OTHERWISE. In this situation, the absence of an OTHERWISE will cause an error. Example:

```
State Fn Ft Fm
Select
      when rc=0 then do
                       erase Fn Ft Fm
                       say 'File existed, Now erased'
                     end
      when rc=28 ! rc=36 then say 'File does not exist'
      otherwise
              say 'Unexpected return code' rc 'from STATE'
              exit rc
End /* Select */
```

Note: A null clause is not an instruction, so putting an extra semicolon after a WHEN clause is not equivalent to putting a dummy instruction. The NOP instruction is provided for this purpose.

SIGNAL

```
SIGNAL < labelname          > ;
       <                    >
       < (VALUE) expression >
       <                    >
       <         < ERROR   > >
       < (  ON  ) < HALT    > >
       < (  OFF ) < NOVALUE > >
       <          < SYNTAX  > >
```

Where, labelname is a symbol that is taken as a constant.

The SIGNAL instruction causes an abnormal change in the flow of control, or (if ON or OFF is specified) controls the trapping of exceptions.

In the case of neither ON nor OFF being specified: labelname is used directly, or is the result of expression if VALUE is specified All active pending DO, IF, SELECT, and INTERPRET instructions in the current routine are then terminated (that is, they cannot be reactivated). Control then passes to the first label in the program that matches the required string, as though the search had started from the top of the program. The match is done independently of alphabetic case, but otherwise the label must match exactly. Example:

```
Signal fred;  /* Jump to label "FRED" below */
 ....
 ....
Fred: say 'Hi!'
```

Since the search effectively starts at the top of the program, control will always pass to the first label in the program if duplicates are present. That is, duplicate labels are ignored.

In the case of ON or OFF being specified: The condition is either enabled (ON) to trap an event or disabled (OFF). When a condition is enabled and the corresponding event occurs, the corresponding action (described below) will be taken.

The conditions and their corresponding events, which may be trapped, are:

ERROR: any host command returns a non-zero return code.

HALT: an external attempt is made to interrupt execution of the program, for example, by using the CMS immediate command, HI (Halt Interpretation).

NOVALUE: an uninitialized variable is used in an evaluated expression, or following the VAR keyword of the PARSE instruction, or in an UPPER instruction. NOVALUE will trap a return of LIT on a function call SYMBOL('name').

SYNTAX: an interpretation error is detected.

The initial setting of all conditions is OFF. When a condition is disabled (either initially or if OFF has been specified) the trap is not in effect. So, when the corresponding event occurs, no special action is taken.

When a condition is currently enabled (ON has been specified) the trap is in effect. So, when the corresponding event occurs, instead of the usual action at that point, the special action is taken - execution of the current instruction is terminated and a SIGNAL instruction is executed automatically. This causes control to pass to the first label in the program that matches the condition. Example:

```
Signal on error
 ...
erase               /* this command gives a non-zero  */
                    /* return code                    */
 ...
ERROR:               /* Program will continue from here */
say "Return code was" rc
```

Once an event is trapped, its corresponding condition is disabled (before the SIGNAL takes place), and a new SIGNAL ON instruction is required to re-enable it. Therefore, for example, if the required label is not found, a normal Syntax Error exit will be taken, which traces the name of that label and the clause in which the event occurred. For ERROR and SYNTAX, the REXX special variable RC is set to the error return code or syntax error number respectively before control is transferred to the condition label.

SOURCELINE((n))

If n is omitted, returns the line number of the final line in the source file. If n is given, the nth line in the source file is returned. n must be a positive whole number, and must not exceed the number of the final line in the source file. Here are some examples:

```
SOURCELINE()    ->  10
SOURCELINE(1)   ->  '/* This is a 10-line program */'
```

SPACE(string(,(n)(,pad)))

formats the blank-delimited words in string with n pad characters between each word. n must be non-negative. If it is 0, all blanks are removed. Leading and trailing blanks are always removed. The default for n is 1, and the default pad character is a blank. Here are some examples:

```
SPACE('abc  def  ')          ->   'abc def'
SPACE('  abc def',3)         ->   'abc   def'
SPACE('abc  def  ',1)        ->   'abc def'
SPACE('abc  def  ',0)        ->   'abcdef'
SPACE('abc  def  ',2,'+')    ->   'abc++def'
```

STORAGE((address(,(length)(,data))))

returns the current virtual machine size expressed as a hexadecimal string if no arguments are specified. Otherwise, returns length bytes from the user's memory starting at address. length is in decimal; the default is 1 byte. address is a hexadecimal number.

If data is specified, after the "old" value has been retrieved, storage starting at address is overwritten with data (the length argument has no effect on this). If length would imply returning storage beyond the virtual machine size, only those bytes up to the virtual machine size are returned; and if an attempt is made to alter any bytes outside the virtual machine size, they are left unaltered. Example:

```
STORAGE(AA,9)   ->   'IBM VM/SP' /* Maybe!              */
```

STRIP(string(,(option)(,char)))

removes characters from string based on the option specified. Valid xph. options (of which only the capitalized letter is significant, all others are ignored) are:

Leading removes leading characters from string;

Trailing removes trailing characters from string;

Both removes both leading and trailing characters from string. The default is B.

The third argument, char, specifies the character to be removed, with the default being a blank. If given, char must be exactly one character long. Here are some examples:

```
STRIP('  ab c  ')          ->    'ab c'
STRIP('  ab c  ','L')      ->    'ab c  '
STRIP('  ab c  ','t')      ->    '  ab c'
STRIP('12.7000',,0)        ->    '12.7'
STRIP('0012.700',,0)       ->    '12.7'
```

SUBSTR(string,n(,(length)(,pad)))

returns the substring of string that begins at the nth character, and is of length length, padded with pad if necessary. n must be a positive whole number. If length is omitted it defaults to be the rest of the string. The default pad character is a blank. Here are some examples:

```
SUBSTR('abc',2)            ->    'bc'
SUBSTR('abc',2,4)          ->    'bc  '
SUBSTR('abc',2,6,'.')      ->    'bc....'
```

Note: In some situations the positional (numeric) patterns of parsing templates are more convenient for selecting substrings, especially if more than one substring is to be extracted from a string.

SUBWORD(string,n(,length))

returns the substring of string that starts at the nth word, and is of length length blank-delimited words. n must be a positive whole number. If length is omitted, it defaults to be the remaining words in string. The returned string will never have leading or trailing blanks, but will include all blanks between the selected words. Here are some examples:

```
SUBWORD('Now is the  time',2,2)    ->    'is the'
SUBWORD('Now is the  time',3)      ->    'the  time'
SUBWORD('Now is the  time',5)      ->    ''
```

UPPER variable (variable) (variable)...;

Where: Variable is a symbol, separated from any other variables by one or more blanks.

UPPER may be used to translate the contents of one or more variables to uppercase. The variables are translated in sequence from left to right. Example:

```
a='Hello';  b='there'
Upper a b
say a b      /* would display "HELLO THERE" */
```

Only simple symbols and compound symbols may be specified. An error is signalled if a constant symbol or a stem is encountered. Using an uninitialized variable is not an error, and has no effect, except that it will be trapped if the NOVALUE condition (SIGNAL ON NOVALUE) is enabled.

WORD(string,n)

returns the nth blank-delimited word in string. n must be a positive whole number. If there are less than n words in string, the null string is returned. This function is exactly equivalent to SUBWORD(string,n,1). Here are some examples:

WORD('Now is the time',3) -> 'the'
WORD('Now is the time',5) -> ''

WORDINDEX(string,n)

returns the position of the nth blank-delimited word in string. n must be a positive whole number. If there are not n words in the string, 0 is returned. Here are some examples:

WORDINDEX('Now is the time',3) -> 8
WORDINDEX('Now is the time',6) -> 0

WORDLENGTH(string,n)

returns the length of the nth blank-delimited word in string. n must be positive. If there are not n words in the string, 0 is returned. Here are some examples:

WORDLENGTH('Now is the time',2) -> 2
WORDLENGTH('Now comes the time',2) -> 5
WORDLENGTH('Now is the time',6) -> 0

WORDS(string)

returns the number of blank-delimited words in string. Here are some examples:

WORDS('Now is the time') -> 4
WORDS(' ') -> 0

X2C(hex-string)

Hexadecimal to Character. Converts hex-string (a string of hexadecimal characters) to character. hex-string will be padded with a leading 0 if necessary to make an even number of hexadecimal digits.

Blanks may optionally be added (at byte boundaries only, not leading or trailing) to aid readability; they are ignored. Here are some examples:

X2C('F7F2 A2') -> '72s'
X2C('F7f2a2') -> '72s'
X2C('F') -> '0F'x

X2D(hex-string(,n))

Hexadecimal to Decimal. Converts hex-string (a string of hexadecimal characters) to decimal. If the result cannot be expressed as a whole number, an error results. hex-string may be the null string. If n is not specified, hex-string is taken to be an unsigned number. Here are some examples:

X2D('0E') -> 14
X2D('81') -> 129
X2D('F81') -> 3969
X2D('FF81') -> 65409
X2D('c6 f0'X) -> 240 [5]

CHAPTER 4
CMS WINDOW AND EXECIO FACILITY

4.1 Introduction

The addition of CMS Session Services improves the usability of VM/SP on 3270-type terminals. New functions let you work with data through windows.

4.1.1 Window Functions and Virtual Screens

You can now manage several pieces of information on the physical screen at the same time. Through windows, you can manipulate information as you might rearrange pieces of paper on your desk.

A window is an area on your physical screen that lets you display and manipulate data. Data is maintained in virtual screens. A virtual screen is a "presentation space" or a functional simulation of a physical screen. When you enter input or view output through a window, you are really looking into the virtual screen data.

Because a window reflects a virtual screen, you can do several operations against a virtual screen and view the results in a window. The characteristics of virtual screens that you can manipulate include:

* Reserved areas for information such as titles and PF key descriptions.
* Color and Highlighting.
* Options to log data into a file.

When you work with windows, you do not have to consider the internal interactions between windows and virtual screens. However, as you become more familiar with how they work, you might find it useful to change or manipulate the system's default settings. You can make changes by using the new CMS commands for windows and virtual screens.

4.1.2 What is in a Window

You can position a window almost anywhere on the physical screen. You can have many windows on the screen at once. You can display windows on top of each other and overlap them.

When you work with data in a window, you are actually working with the data in a virtual screen. You can view the data and scroll forward, backward, right, or left through it.

Windows are maintained in an ordered list. You can shuffle the order by "popping" and "dropping" windows.

4.2 Window and Virtual Screen Commands

ALARM VSCREEN

Use the ALARM VSCREEN command to sound the terminal alarm the next time a virtual screen is displayed in a window on the screen. The format of the ALARM VSCREEN command is:

ALARM VSCreen vname

where, vname is the name of the virtual screen for which the alarm sounds. The terminal alarm sounds.

CLEAR VSCREEN

Use the CLEAR VSCREEN command to erase data in a virtual screen. The format of the CLEAR VSCREEN command is:

CLEAR VSCreen vname

USAGE NOTES: The CLEAR VSCREEN command

* Clears the scrollable data area by overwriting the data buffer with nulls.
* Purges data in the queue.

* Reconnects all windows connected to the virtual screen to the top of the virtual screen.

* Leaves the reserved areas and cursor position unchanged.

CURSOR VSCREEN

Use the CURSOR VSCREEN command to position the cursor on a specified line and column in a virtual screen. The format of the CURSOR VSCREEN command is:

```
CURSOR VSCreen vname line col ((options()))
                                options: Reserved
                                         Data
```

where, line is the line number in the virtual screen where the cursor is positioned; col is the column in the virtual screen where the cursor is positioned.

OPTIONS:

Reserved places the cursor in the reserved area of the virtual screen. The line number must be less than or equal to the number of lines in the reserved area. A negative line number positions the cursor in the bottom reserved area, and a positive line number positions the cursor in the top reserved area. The col must be less than or equal to the number of columns in the virtual screen. You cannot specify a line or col of zero when placing the cursor in the RESERVED area.

Data places the cursor in the scrollable data area at the specified line and column. DATA is the default. The line number must be less than or equal to the number of lines in the data area, and must be zero or greater. A value of zero positions the cursor at the line following the current bottom of the virtual screen. The col must be less than or equal to the number of columns in the virtual screen, and must be zero or greater. A value of zero positions the cursor in column two. This allows for a start field in column one.

DEFINE VSCREEN

Use the DEFINE VSCREEN command to create a virtual screen. A virtual vscreen is a functional simulation of a physical display screen. It is a "presentation space" where data is written. The format of the DEFINE VSCREEN command is:

DEFINE VSCreen vname lines cols rtop rbot ((optionA optionB optionC optionD()))

OptionA: TYPe
NOType

OptionB: PRotect High
NOPRotect NOHigh

OptionC: (color) (exthi) (psset)

OptionD: USer
SYstem

where, vname is the name assigned to the virtual screen. You may specify a name up to eight characters in length; lines is the number of scrollable data lines that the virtual screen contains. The number of lines must be one or greater.

cols is the number of columns that the virtual screen contains.

rtop is the number of reserved lines maintained in the top reserved area of the virtual screen.

rbot is the number of reserved lines maintained in the bottom reserved area of the virtual screen.

OPTION A: Option A controls the actions when information is written to the specified virtual screen. The following may be specified:

TYPe specifies that data is moved to the virtual screen when the virtual screen queue is processed. TYPE is the default.

NOType specifies that the virtual screen is not updated.

OPTION B: Option B indicates the default attributes of the data in the virtual screen. The following may be specified:

PRotect: the data is protected.

NOPRotect: the data is not protected. NOPRotect is the default.

High: data is displayed in high intensity.

NOHigh: data is displayed in a normal intensity. NOHigh is the default.

OPTION C: Option C indicates the default extended attributes for the virtual screen. The following may be specified:

color: the color may be Default, Blue, Red, Pink, Green, Turquoise, Yellow, or White.

exthi: the extended highlighting may be None (default), REVvideo, BLInk, or Underline.

psset: the Programmed Symbol Set (PSset) may be specified as PS0 (the default), PS1, PSA, PSB, PSC, PSD, PSE, or PSF.

OPTION D: Option D indicates whether or not the virtual screen is retained when a task abnormally ends (abend) or when the HX (halt execution) command is issued. The following may be specified:

USer: indicates that the virtual screen is deleted when a task abnormally ends (abend) or when the HX (halt execution) command is issued.

SYstem: indicates that the virtual screen is retained when a task abnormally ends (abend) or when the HX (halt execution) command is issued.

DELETE VSCREEN

Use the DELETE VSCREEN command to remove a virtual screen definition. The format of the DELETE VSCREEN command is:

DELete VSCreen vname

USAGE NOTE: When you delete a virtual screen, all windows connected to it are disconnected. Any ROUTE command message classes that have been directed to the virtual screen are rerouted to the CMS virtual screen.

GET VSCREEN

Use the GET VSCREEN command to write lines from a CMS file to the specified virtual screen. The format of the GET VSCREEN command is:

GET VSCreen	vname	fn	ft	fm *	fromrec 1	numrec *

where, vname is the name of the virtual screen to be updated with the data in the specified CMS file.

fn and ft is the filename and filetype of the file.

fm is the filemode of the file. If you do not specify fm or if * is specified, then all accessed disks are searched.

fromrec is the starting record of the file that is moved into the virtual screen. GET VSCREEN starts with the first record in the file by default.

numrec is the number of records that are read from the file and moved into the virtual screen. All the records are read by default. An * means that records are processed until the end of the file is reached.

USAGE NOTE: GET VSCREEN queues each record of a CMS file to the virtual screen.

The information is appended to the end of the virtual screen when the virtual screen is updated.

PUT VSCREEN

Use the PUT VSCREEN command to write the data from the scrollable data area of a virtual screen to a CMS file. The format of the PUT VSCREEN command is:

```
PUT VSCreen  vname fn ft    fm        fromlin        numlin
                            *         1              *
                            A1
```

fm is the filemode of the file. The default is *, which is the first read/write disk in the search order containing the specified file. See the Usage Notes for more information.

fromlin is the starting line in the virtual screen to be copied into the file. PUT VSCREEN starts with the first line in the virtual screen by default. The number specified for fromlin must be within the range of the current top and bottom of the virtual screen.

numlin is the number of lines to be written into the specified file. If numlin is not specified, the default is *, meaning all the lines in the virtual screen starting with the specified line (fromlin) are copied into the file.

USAGE NOTE: If the specified file does not exist, the file is created on the A-disk and the lines are inserted. If the file already exists, the data is appended to the end of the file.

WAITREAD VSCREEN

Use the WAITREAD VSCREEN command from an EXEC to update the virtual screen with data in the virtual screen queue, refresh the physical screen, and wait for the next attention interrupt. The format of the WAITREAD VSCREEN command is:

```
WAITREAD VSCreen vname
```

where, vname is the name of the virtual screen that is waiting for input.

USAGE NOTES:

1. All windows showing virtual screens other than the virtual screen specified in the WAITREAD command are protected. The windows showing the specified virtual screen are either protected or unprotected, depending on how the data being displayed is defined in the virtual screen.

2. WAITREAD VSCREEN executes in the following manner:

* Each virtual screen is updated with data from the virtual screen queue

* The screen image is rebuilt based on the ordered list of windows and displayed on the physical screen

* Wait for the next interrupt (The virtual screen specified in the WAITREAD VSCREEN command is now active.)

* When the interrupt is received, the screen is read

* Information regarding the key pressed, the cursor, and modified fields are passed back to the exec in variables. The virtual screen can then be updated with the variables by using the WRITE VSCREEN command. The exec variables contain:

WAITREAD.0 : the number of variables returned (excluding WAITREAD.0)

WAITREAD.1 : the key that caused the attention interrupt (such as PAKEY n, PFKEY n,CLEAR, or ENTER)

WAITREAD.2 : the word CURSOR followed by the line number and column number of the cursor in the virtual screen, followed by the word DATA or RESERVED indicating the area in which the cursor was located. If the cursor was in the DATA area, the following line and column number pairs may be returned:

	LINE	COLUMN
a.	n	n
b	0	n
c	0	2

a. means that the cursor was positioned between the top and the bottom of the vscreen. The line number may range from 1 to the number of scrollable data lines in the virtual screen. The column number may range from 1 to the number of columns in the vscreen.

b. means that the cursor was positioned on the line immediately following the bottom of the vscreen. The line number will be 0 and the column number may range from 1 to the number of columns in the vscreen.

c. means that the cursor was positioned on a line following the bottom of the vscreen but that the line was not necessarily the one immediately following the vscreen bottom. The line number will be 0 and the column number will be 2.

If the cursor was not in a DATA or RESERVED area, the line and column numbers will both be -1.

WAITREAD.3

.

.

.

WAITREAD.n

where each variable contains the word DATA or RESERVED indicating the type of text updated, followed by the line number and column number of the field that was modified, followed by the modified text.

Note: The keyword values returned in WAITREAD.0, WAITREAD.1, and WAITREAD.2, as well as the EXEC variables WAITREAD.n, are always in American English (AMENG).

3. WAITREAD VSCREEN is an important command for EXECs that read from and write to windows. A typical sequence for such an EXEC would be:

* Define a window and virtual screen (WINDOW DEFINE and DEFINE VSCREEN commands)

* Connect the window to the virtual screen (SHOW WINDOW command)

* Write data to the virtual screen (WRITE VSCREEN command)

* Issue the WAITREAD VSCREEN command

When an interrupt is received, the EXEC can process the WAITREAD.n variables and update the virtual screen using the WRITE VSCREEN command.

4. A field definition character is placed at the start of each row if:

* A window is not connected to a virtual screen at column 1, or

* The number of columns in the window, virtual screen, and physical screen are not the same. As a result, data is shifted one character to the right in each row so that data in column one is not lost. Lines may be truncated on the right, and the location information indicates that the window is showing one less character from the virtual screen.

5. When windows overlap on the physical screen, a field definition character is placed at each window boundary.

6. If only part of a field from a virtual screen is displayed on the physical screen and the field is modified, the entire field is returned as a modified field in a variable WAITREAD.n.

7. The same field can be modified in different windows. These modifications are processed from the top of the screen to the bottom, moving from left to right.

8. The lines in a window that are not top reserved lines, bottom reserved, or data lines from the virtual screen are called pad lines. When a window is connected to the active virtual screen, the pad lines are unprotected. If there are multiple windows connected to the same active virtual screen, only the pad lines of the top-most window are unprotected.

9. Any part of a window that does not map to the virtual screen is protected. For example, suppose you have defined a window that is 24 rows by 80 columns and a virtual screen that is 20 rows by 60 columns. When you connect the window to the virtual screen at row 1 column 1, the 4 rows and 20 columns in the window that do not map to the virtual screen are protected.

WAITT VSCREEN

Use the WAITT VSCREEN command to update a virtual screen with data. The format of the WAITT VSCREEN command is:

```
WAITT  VSCreen vname
                *
```

where, vname is the name of the virtual screen to be updated. An * indicates that all the virtual screens are updated. An * is the default.

WRITE VSCREEN

Use the WRITE VSCREEN command to enter information in a virtual screen. Information is queued to a virtual screen and is displayed the next time the screen is refreshed. The WRITE VSCREEN command provides many possibilities for defining fields, writing data, and updating the plane buffers associated with the virtual screen (color, extended highlighting, and Programmed Symbol Sets). The format of the WRITE VSCREEN command is:

```
WRITE VSCreen   vname line col length (((REServed)(optionA) (optionB)
                                        optionC) (optionD) ()))
                                        optionA: BLANKs
                                                 NULls
                                        optionB: PROtect        High
                                                 NOPROtect      NOHigh
                                                 Invisible
                                        optionC: (color) (exthi) (psset)
                                        optionD: <FIELD >
                                                 <DATA  >
                                                 <COLOR >       text
                                                 <EXTHI >
                                                 <PSS   >
```

Note: If option D is used, a right parenthesis should not be used to mark the end of the options.

where, vname is the name of the virtual screen.

line is the line number of the virtual screen where the write is to begin.

col is the column number of the virtual screen where the write is to begin.

length is the length to be written.

REServed indicates that the line number refers to a reserved line. If you specify RESERVED, line must be a number less than or equal to the size of the reserved area. A negative line number indicates a write in the bottom reserved area, and a positive line number indicates a write in the top reserved area.

OPTION A: Option A is used for padding text in the data buffer. The following may be specified:

BLAnks: pad data with blanks
NULls: pad data with nulls (default)

OPTION B: Option B is the attribute for defining a field. The following may be specified:

PROtect: protected field
NOPRotect:not protected field
High: high intensity field
NOHigh: normal intensity field
Invisible: invisible field. The data written is not displayed on the screen.

OPTION C: Option C is the extended attribute for updating the virtual screen buffers and/or for padding the text being written. If option C is not specified, then the default characteristics of the virtual screen are used. if necessary (see the DEFINE VSCREEN command).

OPTION D: Option D is the buffer for the text. The following may be specified:

FIELD: creates a field. All the buffers will be initialized with either the virtual screen defaults, or the settings specified in options A and/or B and/or C. Text is placed in the field left justified, and is padded or truncated to the field length.

DATA: indicates that the text is to be written to the data buffer of the virtual screen. The text is written for the length specified or until the next field is encountered.

REFRESH

Use the REFRESH command to update virtual screens and their associated windows and refresh the screens. The format is :

REFRESH

CLEAR WINDOW

Use the CLEAR WINDOW command to scroll past all data in the virtual screen to which the window is connected so that no scrollable data is displayed in the window.

USAGE NOTES:

1. CLEAR WINDOW is valid only when the window is connected to a virtual screen (see WINDOW HIDE and SHOW WINDOW).

2. If the window you want to clear is variable size, the window is not displayed when the physical screen is refreshed.

3. If you are writing output (see the VSCREEN WRITE command), CLEAR WINDOW provides a means for starting output at the top of the window. It does not erase data in the virtual screen but instead works like scrolling. CLEAR WINDOW positions the window at the line following the last data line in the virtual screen. When output is written, lines of data are appended to the virtual screen and are displayed starting at the first nonreserved line of the window.

DEFINE WINDOW

Use the DEFINE WINDOW command to create a window with the specified name, size, and position on the physical screen. The format of the DEFINE WINDOW command is:

DEFine WINdow wname lines cols psline pscol ((options()))

options:	VARiable	BORder	POP
	FIXed	NOBorder	NOPop
	TOP	USer	
	NOTop	SYstem	

where, wname is the name assigned to the window. You may specify a name up to eight characters in length.

lines is the number of lines the window displays.

cols is the number of columns the window displays.

psline is the line on the physical screen where the upper or lower edge of the window is positioned. A positive number positions the upper edge of the window on the specified line relative to the top of the screen. A negative number positions the lower edge of the window on the specified line relative to the bottom of the screen.

pscol: is the column on the physical screen where the left edge of the window is positioned.

OPTIONS:

VARiable: indicates that the number of lines in the window may vary depending on the amount of scrollable data displayed.

FIXed: indicates that the number of lines in the window is always constant. FIXED is the default.

BORder: indicates that the borders are displayed when possible. BORDER is the default.

NOBorder: indicates that borders are not displayed.

POP: specifies that the window is displayed on top of all other windows when the virtual screen that the window is showing is updated.

NOPop: specifies that there is no effect on the window's position in the ordered list of windows when the virtual screen that the window is showing is updated. NOPOP is the default.

TOP: specifies that the window may qualify as the topmost window. Most windowing commands process the topmost window by default or when = is specified as the window name.

NOTop: specifies that the window cannot qualify as the topmost window. Windows defined as NOTOP are not processed by default or when a command is specified with = for the window name.

USer: indicates that the window is deleted when a task abnormally ends (abend) or when the HX (halt execution) command is issued.

SYstem: indicates that the window is retained when a task abnormally ends (abend) or when the HX (halt execution) command is issued.

USAGE NOTES:

1. A maximum of 255 windows may be defined at any time.

2. A window may not be defined with the same name as a window that already exists. A window name of "*" and "=" is invalid.

3. Defining a window does not automatically display it. To display a window, it must be connected to a virtual screen.

4. Use the SET WINDOW command to change the options for a window.

5. A window displays as many top and bottom reserved lines as possible, regardless of the position on the virtual screen where the window is connected. The reserved lines are defined and maintained in the virtual screen. See the SET RESERVED command to change the way in which reserved lines are displayed in a window.

6. A window's size and location must be specified in such a way that, excluding borders, the entire window fits on the physical screen.

7. Psline may be specified as either a positive or a negative number. When positive, the window's upper left corner is placed on the physical screen at the line number specified. When negative, the window's lower left corner is placed relative to the bottom of the physical screen. Thus, a psline value of +1 positions a window by its upper left corner to the top line of the physical screen. A psline value of -1 positions it by its lower left corner to the bottom line of the physical screen.

8. Window borders are built outside the area defined for the window. Therefore, it is possible that some or all of the borders may not fit on the physical screen due to the size and position of the window. Borders are highlighted and displayed using the following characters:

top border character is a dash, '-'

bottom border character is a dash, '-'

left border character is a vertical bar, ''

right border character is a vertical bar, ''

Border corners are identified with a plus (+) sign. Use the SET BORDER command to alter border attributes and characters.

9. Single-character Border commands can be entered in the border corners.

10. If the window, virtual screen, and physical screen do not have the same number of columns, it is recommended that you define the window with one column greater than the number of columns in the virtual screen that it is displaying. This provides for the additional field definition character (Start Field) that is necessary for the proper display of the window on the screen, and ensures that a maximum number of columns of virtual screen data are displayed.

11. When you specify a window as VARIABLE, the current number of lines in the window may vary from 0, in which case the window is not displayed, to the number of lines specified for the window. The window size depends upon the amount of scrollable data being displayed.

12. If multiple windows defined with the POP option are showing the same virtual screen, all the windows are displayed on top of the other windows when the virtual screen is updated. Their position in relation to each other is maintained.

DELETE WINDOW

Use the DELETE WINDOW command to remove a window definition.

DROP WINDOW

Use the DROP WINDOW command to move a window down in the order of displayed windows. The format of the DROP WINDOW command is:

DROp WINdow name n

where, wname is the name of the window to be dropped. = indicates that the topmost window is dropped.

n is the number of positions the window is moved down. An "*" positions the window behind all other windows. If this operand is not specified, "*" is assumed.

HIDE WINDOW

Use the HIDE WINDOW command to prevent the specified window from being displayed and to connect the window to a virtual screen. The SHOW WINDOW command will redisplay the window. The format of the WINDOW HIDE command is:

HIDe WINdow wname (ON vname (line col))

where, line is the virtual screen line number where the upper left corner of the window is placed.

col is the column number of the virtual screen where the upper left corner of the window is placed.

MAXIMIZE WINDOW

Use the MAXIMIZE WINDOW command to expand a window to the physical screen size. The RESTORE WINDOW command returns the window to its size and location prior to the maximize.

MINIMIZE WINDOW

Use the MINIMIZE WINDOW command to reduce the size of the window to one line.

NEXT WINDOW

Use the NEXT WINDOW command to move the window toward the bottom of the virtual screen the number of lines specified.

POP WINDOW

Use the POP WINDOW command to move a window up in the order of displayed windows. The format of the POP WINDOW command is:

POP WINdow wname n

where, n is the number of positions the window is to be moved up. An "*" indicates that the window will be positioned on top of the other displayed windows. If this operand is not specified, "*" is assumed.

POSITION WINDOW

Use the POSITION WINDOW command to change the location of a window on the physical screen. The format of the POSITION WINDOW command is:

POSition WINdow <wname> psline pscol

where, psline is the line on the physical screen where the upper or lower edge of the window is positioned. A positive number positions the

upper edge of the window on the specified line relative to the top of the screen. A negative number positions the lower edge of the window on the specified line relative to the bottom of the screen.

pscol is the column on the physical screen where the left edge of the window is positioned.

USAGE NOTES:

1. The window's size and location must be such that, excluding borders, the entire window fits on the physical screen.

2. 'Psline' may be specified as either a positive or a negative number. When positive, the window's upper left corner is placed on the physical screen at the line and column number specified. When negative, the window's lower left corner is placed relative to the bottom of the physical screen. Thus, a psline value of +1 positions a window by its upper left corner to the top line of the physical screen. A psline value of -1 positions it by its lower eft corner to the bottom line of the physical screen.

RESTORE WINDOW

Use the RESTORE WINDOW command to return a maximized or minimized window to its size and location prior to the maximize or minimize.

SHOW WINDOW

Use the SHOW WINDOW command to place a window at the top of the window display order and to connect the window to a virtual screen. The format of the SHOW WINDOW command is:

SHOw WINdow wname (ON vname (line col))

where, wname is the name of the window.

vname is the name of the virtual screen to which the window is connected.

line is the line number of the virtual screen where the upper left corner of the window is placed.

col is the column number of the virtual screen where the upper left corner of the window is placed.

USAGE NOTES:

1. Multiple windows may be connected to a single virtual screen.

2. If the window is already connected to a virtual screen when you issue the SHOW WINDOW command, you do not have to specify the virtual screen information.

3. When you specify a virtual screen name, line, and column, the line and column values must be less than or equal to the corresponding virtual screen dimensions. The minimum line and column value is 1. If line and column are not specified, the default is 1 for both. If the line specified is past the current virtual screen bottom, the window is connected to the virtual screen bottom.

4. A variable size window is only displayed when there is at least one scrollable line to show. If a variable size window is showing a virtual screen that does not contain any scrollable lines, the SHOW WINDOW command places the window at the top of the window display order and connects it to the specified virtual screen, but it is not displayed.

5. When you are using full-screen CMS and you enter the SHOW WINDOW command from the command line, the command is executed and then the screen is refreshed. As part of the refresh processing, any pop-type window that has output waiting is moved to the top of the order of displayed windows. Therefore, the window that you specified may not be displayed at the top of the display order because another has been popped afterwards.

SIZE WINDOW

Use the SIZE WINDOW command to change the number of lines and columns for a specified window. The format of the SIZE WINDOW command is:

SIZe WINdow <wname> lines (cols)
< = >

where, lines is the number of lines in the window.

cols is the number of columns in the window. The default is the current number of columns. [6]

4.3 EXECIO Command

Use the EXECIO command to:

* Read lines from disk or virtual reader to the program stack or a variable.
* Write lines from the program stack or a variable to a CMS disk file or to a virtual spool device (punch or printer).
* Cause execution of CP commands and recover resulting output.

In some cases output data to be written may be supplied directly on the EXECIO command line.

In the following descriptions, "relative line number" means the number of lines processed to satisfy an EXECIO operation; "absolute line number" means the number of the line relative to the top of the file. The format of the EXECIO command is:

```
EXECIO                                  Options:
       <lines> <DISKR fn ft (fm (linenum)) (((FINIs)   (a) (b))      ())>
       <  *  > <CARD  ((               (a) (b))                      ())>
       <CP             ((              (a) (b) (d)) (e))             ())>
       <DISKW fn ft fm (linenum                                       >
       <       (recfm (lrecl))) (((FINIs)   (b) (c) (d))             ())>
       <PUNCH ((                       (b) (c) (d))                  ())>
       <PRINT (((CC <code>)            (b) (c) (d))                  ())>
       <       ( (   <DATA>)                                          >
       <EMSG  ((                       (b) (c) (d))                  ())>

       Option formats:
       (a)
              +              +    +              +   +    +
       FInd /chars/     Zone <n1 n2>  LIFO    (SKip)
       LOcate /chars/           <1  * >FIFO
       Avoid /chars/             +             +   +    +
          +              +

       (b)
          +               +                          +          +
       Margins <n1 n2>     (STRIP)   (NOTYPE)     STEm xxxxn
              <1  * >                            VAR xxxx
          +               +                          +          +
       (c)
          +        +
       CAse   <U>
              <M>
          +        +
       (d)
       (STring xxx...)
       (e)
       (BUFfer length)
```

Note: Parsing of the EXECIO command differs from that of other CMS commands in that it involves handling of strings that may contain embedded blanks, parentheses, other special characters, and words of more that eight characters. Therefore, if a right parenthesis is used to mark the end of an EXECIO option, it must be preceded by at least one blank character. A right parenthesis cannot be used to mark the end of the STRING option.

where:

lines

is the number of source lines processed. This can be any non-negative integer. With the VAR option, the number of lines must be 1. An asterisk (*) indicates that the operation is to terminate when:

1. a null (0-length) line is read during an output operation;
2. an end-of-file condition is detected during an input operation.

Specification of *, together with the STRING option, is valid only with the CP operand. Using the * and STRING combination with any other operand causes an error message to be issued. Also the combination of the * and the VAR options is not allowed. If "lines" is specified as zero (0), no I/O operation is performed other than FINIS, when it is specified as an option.

DISKR

is used to read a specified number of lines from the CMS file "fn ft (fm)" to the program stack FIFO (first-in first-out) or to an EXEC 2 or System Product interpreter variable if the STEM or VAR options are specified.

CARD

is used to read a specified number of lines from the virtual reader to the program stack (FIFO) or to an EXEC 2 or System Product interpreter variable if the STEM or VAR options are specified.

CP

causes output resulting from a CP command to be placed on the program stack (FIFO) or to an EXEC 2 or System Product interpreter variable if the STEM or VAR options are specified. To obtain the reply from a CP command, specify the "lines" operand as an asterisk (*). If you want to issue a command to CP, suppressing messages and obtaining only the return code, specify the "lines" operand as a zero (0). You may specify which CP command is to be issued via:

1. the STRING option on the EXECIO command line;
2. the next line from the program stack.

Keep in mind that all characters of CP commands must be in upper case.

DISKW

is used to write a specified number of lines from the program stack or from an EXEC 2 or System Product interpreter variable if the STEM or VAR options are specified to a new or existing CMS file "fn ft fm." Inserting a line into a variable length CMS file can cause truncation of the portion of the file following the inserted line.

recfm
lrecl

define the record format and record length for any new file created as a result of a DISKW operation. The default value for recfm is V (variable), in which case "lrecl" has no meaning. If you specify F (fixed) for recfm, the default lrecl value is 80. The maximum lrecl value that may be specified is 255 unless the VAR or STEM option is used to bypass the use of the program stack, in which case the maximum is that which is defined for the type of file in use (800 byte or EDF file). If recfm or lrecl is specified for the DISKW operation, then a linenum value must be specified explicitly.

PUNCH

is used to transfer a specified number of lines from the program stack or from an EXEC 2 or System Product interpreter variable the virtual punch if the STEM or VAR options are specified to .

PRINT

is used to transfer a specified number of lines from the program stack or from an EXEC 2 or System Product interpreter variable if the STEM or VAR options are specified, to the virtual printer using the PRINTL macro.

CC

is used with the PRINT operand to specify carriage control for each line transferred to the virtual printer. Using the CC operand, you can supply carriage control code explicitly, or, by specifying DATA, indicate that the carriage control character is the first byte of each line.

code

is the character (ASA or machine code) that defines carriage control. A blank code (the default value) cannot be specified on the command line.

DATA

specifies that the first byte of each line sent to the virtual printer is a carriage control character.

EMSG

causes a message to be displayed.

fn
ft

is the filename and filetype of the file.

fm

is the filemode of the file. When filemode is specified, that disk and its extensions are searched. If filemode is not specified, or is specified as an asterisk (*), all accessed disks are searched for the specified file.

linenum

is the absolute line number within the specified file where a DISKR or DISKW operation is to begin. If a value is zero or not specified for newly opened files, reading begins at the first line and writing begins at the last line. For other files, reading or writing begins at the line following the one at which the previous operation ended. Because EXEC processors manipulate EXECs that are currently executing, a read or write to a currently running EXEC should explicitly specify the linenum operand. Failure to do so may cause the first line to be read or written each time. If recfm or lrecl is specified for the DISKW operation, then a linenum value must be specified explicitly.

FINIs

causes the specified file to be closed following completion of a DISKR or DISKW operation.

OPTION A

FInd

is used to write to the program stack LIFO (last-in first-out) or FIFO (first-in first-out) to an EXEC 2 or System Product interpreter array.

1. the contents of that line;
2. the line number of the first occurrence of a line (or zone portion of that line) that begins with the characters specified between delimiters. For DISKR operations both the relative and absolute line numbers are written. Otherwise, only the relative line number is written.

If you wish to search only a portion of each line, use the ZONE option, explained below. If you wish to write only a portion of any line matching the search argument to the program stack or a variable, use the MARGINS option, also explained below.

LOcate

is like the FIND option explained above, except that the object characters may occur any place within a line (or zone portion of that line).

Avoid

is like the LOCATE option explained above, except that the search is for a line (or zone portion of that line) that does not contain the specified characters.

Zone

is used to restrict the portion of the input lines searched as a result of the FIND, LOCATE, or AVOID options. The search is between columns n1 and n2 (inclusive), if specified. The default values are column 1 through the end of the line (*). The limits of values that may be specified for n1 or n2 are 1 through 2(31)-1.

LIFO

FIFO

defines the order in which lines are written to the program stack. Generally, the default order is FIFO (first-in first-out). The exceptions are operations that put line numbers on the program stack as a result of a search operation (FIND, LOCATE, or AVOID). These operations default to LIFO (last-in first-out). The use of VAR or STEM is not valid with this option.

SKip

allows a read function (DISKR, CARD, CP) to occur without writing any information to the program stack.

OPTION B

Margins

specifies that only a portion (columns n1 through n2 inclusive) of affected lines is to be processed (from the stack or a variable). The default values are column 1 through the end of each line (*). The limits of values that may be specified for n1 or n2 are 1 through 2(31)-1.

STRIP

specifies that trailing blank characters are to be removed from any output lines or lines returned.

NOTYPE

suppresses the display of message DMSEIO632E at the virtual console.

STEm xxxxn

indicates that the variables xxxxn are to be used to supply input data for output-type operations (PUNCH, PRINT, EMSG, and DISKW) or that variables xxxxn are the destination for output for the input-type operations (CARD, DISKR, and CP). The variables used are a concatenation of the specified stem and a number that corresponds to the number of lines of output. The special variable xxxx0 is set to the number of lines of information returned for the input-type operations. STEM can be used with the STRING operation only with the CP operation. The maximum length variable name with the STEM option is 240 bytes.

VAR xxxx

indicates that the variable xxxx is to be used to supply input data for output-type operations (PUNCH, PRINT, EMSG, and DISKW) or that variable xxxx is the destination for output for the input-type operations (CARD, DISKR, and CP). If VAR is specified, then the number of lines must be 1. VAR cannot be used with the FIFO or LIFO options, nor with

the FIND, AVOID, and LOCATE options because they cause more than one line of output to be written. VAR can be used with the STRING option only with the CP operation. The maximum length variable name is 250 characters.

OPTION C

CAse

causes data read from the program stack, from a STEM, or from a variable to be:

1. translated to uppercase if U is specified;
2. not translated (mixed) if M is specified. M (mixed) is the default value.

OPTION D

STring

is used to supply output data explicitly on the EXECIO command line. Any characters following the STRING keyword are treated as string data, not additional EXECIO operands. Therefore, STRING, if specified, must be the final option on the command line.

OPTION E

BUFfer length

specifies the length, in characters (bytes), of the CP command response expected from a CP operation. The limits of values that may be specified for length are 1 through 2(31)-1. If this option is not specified, up to 8192 characters of the response are returned. [7]

EXECIO command was used on a large scale in this study. To record the the publication data, 'WRITE' option of EXECIO was used. The incoming data is stored in files sequentially. In order to quickly locate the key words in publication files, EXECIO was used with the 'LOCATE' option.

CHAPTER FIVE
THE DATABASE OF PUBLICATIONS

5.1 Introduction

The IUCV and window features of CMS were used in this database program written in REXX.

The database was limited to the publications of the Control and Computer Engineering Department in order to facilitate performance analysis.

The user of this program is asked whether to record his publication or to search for items in the database. If the user selects the recording option, a window is opened and he is presented with the general recording form. Then the user is requested to enter some information such as the title, the language, the type of the publication, name(s) and the title(s) of the author(s). In order to reduce storage overhead, some of the information such as the title of author, the type and the language of publication was coded. If the user enters a wrong code, he is requested to enter a correct one.

The publications have been classified as Scientific Article, Published Notice, Book, Course Note, Published Report , M.S. or Ph.D. Thesis.

After the user completes the general recording form, another window is opened. In this window, a form associated with the publication type is displayed and the user is requested to fill out these fields as well.

Finally, if the user press 'ENTER' key, the program sends the data to database VM (virtual machine) via IUCV. The database VM collects information sent and processes it. If the first item of the data received on the database VM is 'RECORDING', then the following data is recorded on the database appropriately.

If the request is 'SEARCH' then the following data is regarded as the search criteria. If any items are found on the database satisfying the criteria, the related data is sent to the user VM via IUCV. The user program which receives data then shows the information of publications in a new window.

5.2 The Database Program

The program first loads IUCV and prepares to receive messages from all users with the command 'IUCV CONNECT *MSG'.

Should the execution of this command fail for any reason, a warning message is sent to the author's VM (virtual machine), 'FARUK'. After successful initialization of IUCV services, the program continues execution by assigning the database files to variables, one for each publication type:

```
File.1 = 'ARTICLE DATA A'
    .
    .
File.7 = 'THESIS DATA A'
```

This classification scheme provides for shorter search delays.

The incoming 'RECORD' requests are classified according to publication types and are stored in the file associated with their type.

The rest of the program can be summarized as follows:

```
Do forever
        Pull msg
        Select
                When msg.1 = 'RECORD' then call RECORD
                When msg.1 = 'SEARCH' then call SEARCH
                Otherwise nop
        End
End
```

The first task in the main loop is to check whether there is any queued request from another user. If there is any, then this request is dealt with. If not, then the program waits for requests with the IUCV command.

If the first word in the incoming message is 'RECORD', then the procedure named RECORD is called to handle the recording process. If it is 'SEARCH', then the second word is taken as the search criterion. The requested search is then handled by a procedure, as before. If the first word is not either 'RECORD' or 'SEARCH' then the request cannot be recognized and no operation takes place.

The <Select-End> structure described above is repeated whenever a new message arrives, thanks to the (Do forever - End) loop.

5.3 The Procedure RECORD

This procedure first determines whether the file 'FORM NO A' is present. The first time a publication is recorded, this file will be absent and the form number is reset to zero. On subsequent recordings, this file will be present and will furnish the number of the last form recorded. The current form number is determined by incrementing this, and the new number is written back to the file 'FORM NO A' for future reference.

At this stage, the publication data is received from IUCV. The File Management Facilities of CMS have been effectively used when storing these data. For example, the EXECIO command of CMS writes data using the variable line size format. Since variable record format files take up less space than fixed format files, this command helps to reduce storage consumption.

In the variable format, the linesize is limited to 255 characters. In order to fully utilize this limit, some data fields were concatenated before recording using the Carriage Return character "15'X as a separator. Concatenation has further reduced storage consumption.

5.4 The Procedure SEARCH

This procedure is called to handle search requests from the users. Consider the case in which the user wants publications searched according to authors' surnames. In this case, the first word in the user's message will be 'SEARCH' and the second word will be 'SURNAME'. Finally, the third word will be the surname of the queried author.

The actual search is carried out by the associated search procedure, in this case SRCH_SUR. The LOCATE option of the command EXECIO is used to scan through the files associated with each publication type. When a match occurs, the publication data is transferred to the requesting user's VM via IUCV. If no match occurs, the message 'NO DATA' is sent to the user, and the user's program deals with it as appropriate.

After the completion of all these tasks, program control returns to the <Do forever - End> loop in the main body of the program.

5.5 User Interface Program

The user interface program communicates with the database VM in order to carry out the user's requests. In case of error, the statement 'Signal on ERROR' causes a branch to the label ERROR. The type of error is indicated by the reserved system variable Rc, so appropriate action can be taken.

For example, if the user VM is unable to establish an IUCV connection with the database VM, this is indicated by the return code Rc, and a message is sent to the operator's VM requesting him to start the database VM.

The database VM is identified in the user program with the statement

dbase_vm = '.....'

where the dots inside the quotes represent the userid of the database VM.

The connection with the database VM is established with the command 'IUCV CONNECT' dbase_vm.

The Window Facility of CMS Release 5 is used to build a 'user friendly' environment of communication. Before the user is presented with a window to interact with, the fixed (i.e. not editable) fields are defined and initialized by the program.

At first the logo 'İTÜ KONTROL & BİLGİSAYAR BÖLÜMÜ YAYIN BİLGİ BANKASI' is displayed. Afterwards the user is presented with a main menu allowing him to choose either recording or searching options.

If the user asks to record a new publication, a form will be presented inside a window. This form will prompt the user for general data on the publication, e.g. its title, type, language and author's name.

Some data such as publication language and type have been numerically encoded. If the user enters an invalid code, he is rejected with an alert message.

If all the data entered is valid, then the program proceeds to the next stage. The variable publication type is tested in a <Select - End> construct to transfer execution to the publication type dependent part of the programs:

```
Select
        When type = 1 then signal ARTICLE
                :
        When type = 5 then signal REPORT
        Otherwise signal THESIS
End
```

After the type-associated branching, the user is presented with a second form prompting him for publication type dependent data. As before, this form is displayed using the window facility of CMS. Therefore the user can switch back and forth between the two forms at the press of a key.

After both forms have been filled out with valid data and the 'ENTER' key is pressed, the user is presented with the main menu again.

The data entered into the window forms is parsed by the user interface and sent to the database VM in the appropriate format. The first word in this format will be the word 'RECORD', to set this request apart from the search request.

If instead of recording, the user asks to search a previously recorded publication, he will be presented with a second menu prompting for a numerically encoded search field. For example, the code '1' represents a search according to authors' surnames. The codes are explained in the menu.

At this example, the user requests a surname based search, he is prompted for the surname, which is assigned to the variable surnamq.

The search request data is sent to the database VM as follows:

```
'IUCV SEND' dbase_vm 'SEARCH'
'IUCV SEND' dbase_vm 'SURNAME'
'IUCV SEND' dbase_vm surnamq
```

After that, the reply from the database VM is waited using the command 'IUCV WAIT'. It was observed that when the computer is heavily loaded, the reply is fragmented and the 'IUCV WAIT' command misses the later parts of the message. To solve this problem, the delay command 'CP SLEEP 1 SEC' was used.

If the reply is "NO_DATA' then the user is informed that the search was unsuccessful. Otherwise the reply is made up of matching publications and is presented to the user by a procedure.

This procedure strips the '15'x characters from the reply and assigns the data fields to the associated variables.

A stand alone '15'x on a line indicates the end of data on a particular publication so that any data that follows it is assumed to be related to another publication.

The results of the search are displayed to the user inside a new window. The user can ask for detailed information on a publication by typing an 'x' beside the author's surname information.

RESULTS AND CONCLUSIONS

The program involved in this study demonstrates how one could use the CMS IUCV support in VM using Message System Service. It establishes communications via IUCV and then waits for incoming messages. Each time a message comes in, its request is distinguished by examining the first word of it.

The database is established only on the Control and Computer department publications for the purpose of testing.

IUCV module used in this study has some limitations. For example one can send of a data line of maximum length 247, because the userid part of the message occupies the first eight bytes. Also the maximum number of lines that can be sent at a time is 255. So the total amount of information that can be sent at a time is nearly 64 Kbytes. That limits the number of items that will be sent if the number exceeds 256.

Consequently the IUCV method is not suitable for applications that require transfer of very large volumes of data such as a university's central library's database system.

This method can be used for local applicatons such as a departmental publication database.

LIST OF REFERENCES

[1] KRISHNAMURTHY, E.V., Parallel Processing, International Computer Sciences Series, Addison Wesley Publishing Company, 1989

[2] IBM, System Facilities for Programming, Release 5, IBM U.S.A., 1986

[3] University of Maine System, IUCV Helpdisk, U.S.A., 1987

[4] IBM, System Product Interpreter User's Guide, Release 5, IBM U.S.A., 1986

[5] IBM, System Product Interpreter Reference, Release 5, IBM U.S.A., 1986

[6] IBM, Using Release 5 Enhancements, IBM U.S.A., 1986

[7] IBM, CMS Command Reference, Release 5, IBM U.S.A., 1986

APPENDIX
SAMPLE SCREEN SHOTS OF THE USER PROGRAM RUNNING

(1) - YAYIN GİRİŞİ

(2) - YAYIN TARAMA

PF12 : ÇIKIŞ

YAYIN GİRİŞ FORMU

Yayın Adı: _____
_____
_____

Ünvanlar
(01)-Prof.Dr. (02)-Prof. (03)-Doç.Dr. (04)-Doç. (05)-Y.Doç.
(06)-Y.Doç.Dr. (07)-Dr. (08)-Araş.Gör. (09)-Uzman (10)-Öğr.Gör.

Yazarlar

	Ünvan	Adı	SOYADI	Kurumu (İTÜ ise Sicil No)
1.	__	_____	_____	_____
2.	__	_____	_____	_____
3.	__	_____	_____	_____
4.	__	_____	_____	_____
5.	__	_____	_____	_____

Yayın Cinsi: _ (1)-Makale (2)-Bildiri (3)-Ders Notu (4)-Kitap
(5)-Rapor (6)-Y.Lis.Tezi (7)-Doktora Tezi
Yayın Dili : __ (01)-Almanca (03)-Farsça (05)-İngilizce (07)-İtalyanca
(02)-Arapça (04)-Fransızca (06)-İspanyolca (08)-Rusça
Yayının Konusu ile ilgili Doç.Bilim Dalı Kodları (00)-TÜRKÇE
_____ _____ _____ _____ _____

ENTER : Bir sonraki sayfaya geç PF12 : Ana menüye dön

Yayın cinsi "BİLİMSEL MAKALE" ise ,

Yayınlandığı Periyodiğin Adı : _____
_____

Periyodiğin Yıl, Cilt, Sayı, Sayfa No.ları
19__ ___ ____ ____ ____

Periyodik "Science Citation Index" içinde ise
Periyodiğin Index'teki Kodu : _____

Çeviri Makale ise Orijinal Adı : _____
_____
_____
_____

Orijinal Makalenin Yayınlandığı Periyodun Adı
_____
_____

Yayın Yılı : 19__

ENTER : Yazılanları kaydet PF10 : Önceki sayfaya dön PF12 : Ana menüye dön

Yayın cinsi "YAYINLANMIŞ BİLDİRİ" ise ,

Bildirinin verildiği Kongre veya Sempozyum:
_____
_____

Düzenleyen Kurum : _____
_____

Düzenlendiği Şehir / Ülke : _____ / _____

Bildirinin Yayınlandığı Yayının Adı: _____
_____

Yılı / Sayfa No.ları : 19__ / ____ ____

ENTER : Yazılanları kaydet PF10 : Önceki sayfaya dön PF12 : Ana menüye dön

Yayın cinsi "DERS NOTU" ise ,

Yayınlayan Kurum : _____
_____

Yayın Yılı : 19__

Kaçıncı Baskı : __

Sayfa Sayısı : ____

ENTER : Yazılanları kaydet PF10 : Önceki sayfaya dön PF12 : Ana menüye dön

Yayın cinsi "KİTAP" ise ,

Yayınevi veya Yayınlayan Kurum : _____
_____

Yayın Yılı : 19__

Kaçıncı Baskı : __

Sayfa Sayısı : ____

Çeviri ise Orijinal Adı : _____
_____
_____
_____

Yayın Yılı : 19__

ENTER : Yazılanları kaydet PF10 : Önceki sayfaya dön PF12 : Ana menüye dön

Yayın cinsi "YAYINLANMIŞ RAPOR" ise ,

Yayınlayan Kurum : _____
_____

Yayınlanma Şekli : _ Baskı ==> 1 Teksir ==> 2

Yayın Yılı : 19__

Sayfa Sayısı : ____

ENTER : Yazılanları kaydet PF10 : Önceki sayfaya dön PF12 : Ana menüye dön

Yayın cinsi "Y.LİSANS TEZİ" veya "DOKTORA TEZİ" ise ,

Tezin Kabul Edildiği Yıl : 19__

Öğrenci : _ <1> ==> İTÜ'den
<2> ==> Başka Üniversiteden

İlgili Enstitü : _ <1> ==> Fen B.Ens.
<2> ==> Sos.B.Ens.
<3> ==> N.En. Ens.

ENTER : Yazılanları kaydet PF10 : Önceki sayfaya dön PF12 : Ana menüye dön

YAYIN TARAMA KRİTERLERİ

1 ==> Yazar Soyadına göre
2 ==> Yayın Adına göre
3 ==> Yayın Yılına göre
4 ==> Yayın Tipi ve Yılına göre
5 ==> Yayın Diline göre
6 ==> Yayın Tipi ve Diline göre
7 ==> Doçentlik Koduna göre

Seçiminiz : _

ENTER : Veri Tabanını sorgula PF12 : Ana menüye dön

YAYIN TARAMA KRİTERLERİ

1 ==> Yazar Soyadına göre
2 ==> Yayın Adına göre
3 ==> Yayın Yılına göre
4 ==> Yayın Tipi ve Yılına göre
5 ==> Yayın Diline göre
6 ==> Yayın Tipi ve Diline göre
7 ==> Doçentlik Koduna göre

Seçiminiz : 1

Yazarın Soyadı : AK_____

ENTER : Veri Tabanını sorgula PF12 : Ana menüye dön

YAZARIN SOYADI, ADI	YAYININ TİPİ	YAYIN DİLİ	FORM NO
AKMAN, CELAL	BİLİMSEL MAKALE	ALMANCA	22
AKYAMAN, MAHMUT	YAYINLANMIŞ BİLDİRİ	FRANSIZCA	37
AKGÜN, TEVFİK	DERS NOTU	TÜRKÇE	28
AKTUNA, FAHRİ	KİTAP	İNGİLİZCE	2
AKTURA, HASAN	YAYINLANMIŞ RAPOR	TÜRKÇE	16
AKGÜN, OLCAY	Y.LİSANS TEZİ	TÜRKÇE	38
AKGÜN, TEVFİK	DOKTORA TEZİ	İNGİLİZCE	10

YAYIN HAKKINDA AYRINTILI BİLGİ ALMAK İÇİN İLGİLİ SÜTUNA X YAZIP ENTER'A BASINIZ

PF7 : Önceki sayfaya dön PF8 : Bir sonraki sayfaya geç PF12 : Ana menüye dön

/* THE DATABASE PROGRAM */

```
trace O
                    /* INITIALIZING IUCV */
'IUCV LOAD'
'SET CMSTYPE HT'
Signal on SYNTAX
'IUCV CONNECT *MSG' ; RS=RC
'SET CMSTYPE RT'
If RS^=0 then do ; 'tell faruk IUCV"ye bağlanılamadı!' ; exit ; end
E='15'X
file.1='ARTICLE DATA A'
file.2='NOTICE  DATA A'
file.3='NOTE    DATA A'
file.4='BOOK    DATA A'
file.5='REPORT  DATA A'
file.6='THESIS  DATA A'
file.7='THESIS  DATA A'

                    /* WAITING FOR USER REQUESTS */
/* Main LOOP */
LOOP:
Do Forever
   Drop DATA. MSG. which
  'DROPBUF'
   If que='ON' then
      Do
         que='OFF'
         size=sizeq
         user=userq
         Do i=1 to size
            dt.i=dtq.i
         End
         call Q_PARSE
      End ;    else
      Do
        'IUCV WAIT'      ;   'CP SLEEP 1 SEC'
        'IUCV GRAB'
         Size=queued()   ;    str=''
         Do i=1 to size  ;    Pull dt.i   ;   End
         User = strip(substr(dt.1,1,8))
         call Q_PARSE
      End
         Do i=1 to size
            MSG.I=translate(substr(dt.i,9),'IĞÜŞÖÇ','ığüşöç')
         End
   'IUCV CONNECT' user
   If RC^=0 then
      Do
          'CP MSG' user 'İSTEĞİNİZ YERİNE GETİRİLEMEDİ, LÜTFEN BİR
DAHA DENEYİN'
         Signal LOOP
      End
```

```
            Select
              When (MSG.1='RECORD') then call RECORD
              When (MSG.1='SEARCH') then
                                Do
                                  flag = 0 ; ind = 0 ; sdata. = ''
                                  Select
                                    When (MSG.2='SURNAME') then call SRCH_SUR
                                    When (MSG.2='PRTITLE') then call SRCH_PRT
                                         When (MSG.2='YEAR') then call SRCH_YEAR
'ALL'
                                         When (MSG.2='YRTY') then call SRCH_YEAR
'SOME'
                                         When (MSG.2='LANG') then call SRCH_LANG
'ALL'
                                         When (MSG.2='LNTY') then call SRCH_LANG
'SOME'
                                    When (MSG.2='SCOD') then call SRCH_SCOD
                                    Otherwise nop
                                  End
                                End
              Otherwise nop
            End
          End

          Q_PARSE:
                Drop sizeq msgq.
                Do i=1 to size
                  If strip(substr(dt.i,1,8))^=User then
                    Do
                      que='ON'
                      userq=strip(substr(dt.i,1,8))
                      Do j=i to size
                        k=j-i+1
                        dtq.k=dt.j
                      End
                      sizeq=k
                      size=i-1
                      Leave i
                    End
                End
          Return

                          /* RECORDING COMING DATA */
          RECORD:
          Drop prtitle maxi maxs type lang
          Address command STATE FORM NO A
          If RC=0 then 'EXECIO 1 DISKR FORM NO A (VAR FORM_NO'
                 else form_no = 0
          form_no = form_no + 1
          'EXECIO 1 DISKW FORM NO A 1 (FINIS STR' form_no
          if MSG.3='MSG.3' then
               Do
                  'CP MSG' USER 'YAYININIZ KAYDEDİLEMEDİ, LÜTFEN BİR DAHA
DENEYIN'
                        'CP MSG OPERATOR YAYIN PROGRAMINDA BİR HATA
VAR,FARUK"U ARAYIN'
```

```
Do i=4 to x by 4
   j=i+1 ; k=i+2 ; l=i+3
        /*  Title,   Name,   Surname, and Regno variables received */
   aut=aut!!MSG.i!!e!!MSG.j!!e!!MSG.k!!e!!MSG.l!!e
End
m= x+4 ; mm = m+1 ; type = MSG.m ; lang = MSG.mm
nn=mm+1 ; maxs=MSG.nn ; n=nn+maxs
recs=e!!form_no!!e!!prtitle!!e!!lang
Do i=1 to maxs
   S=nn+i
   Scode.i=MSG.s
   Recs=recs!!e!!scode.i
End
'EXECIO 1 DISKW' file.type '(STR' recs
'EXECIO 1 DISKW' file.type '(STR' aut
Select
  When type=1 then
      Do
         Drop per year pervol perno perpg1 perpg2 percode orart orper oryear
         Do i=n+1 to size
            Select
               When i=n+1  then  per=MSG.I
               When i=n+2  then  year=MSG.I
               When i=n+3  then  pervol=MSG.I
               When i=n+4  then  perno=MSG.I
               When i=n+5  then  perpg1=MSG.I
               When i=n+6  then  perpg2=MSG.I
               When i=n+7  then  percode=MSG.I
               When i=n+8  then  orart=MSG.I
               When i=n+9  then  orper=MSG.I
               When i=n+10 then  oryear=MSG.I
               Otherwise nop
            End
         End
         Art_inf=per!!e!!year!!e!!pervol!!e!!perno!!e!!perpg1!!e!!perpg2
         If percode^='PERCODE' then art_inf=art_inf!!e!!percode
        'EXECIO 1 DISKW' file.1 '(STR' art_inf
         If orart^='ORART' then
            Do
               Or=orart!!e!!orper!!e!!oryear
              'EXECIO 1 DISKW' file.1 '(STR' or
            End
      End
  Otherwise
      Do
         Inf=''
         Do i=n+1 to size
            Inf=inf!!MSG.I!!e
         End
        'EXECIO 1 DISKW' file.type '(STR' inf
      End
End
'FINIS' file.type
Return
```

```
   /* SEARCH PROCEDURE USING SURNAME AS A PARAMETER */

SRCH_SUR:
Surnamq=MSG.3
Do type=1 to 6
   k=0  ; drop num. line.
   Do forever
     'EXECIO * DISKR' file.type '(LOCATE /'surnamq'/'
      If rc^=0 then leave
      K=k+1
      Pull rel.k num.k  ; Parse pull line.k
   End
   Do i=1 to k
      drop line_f line_s line_t
      l=num.i
     'EXECIO 1 DISKR' file.type l-1 '(VAR LINE_F'
      If ^( substr(line_f,1,1)='15'X  &  FNDX(line.i';;'surnamq)='OK' )
                                      then iterate i
     'EXECIO 1 DISKR' file.type l+1 '(VAR LINE_S'
      call collect type
      call collect left(line_f,247)
      call collect left(line.i,247)
      call collect left(line_s,247)
      If type=1 then Do
                     'EXECIO 1 DISKR' file.type l+2 '(VAR LINE_T'
                      If Rc=0 & substr(line_t,1,1)^=e then
                      call collect left(line_t,247)
                     End
      call collect '15'X
      Flag = 1
   End
'FINIS' file.type
End
If flag=0 then call collect 'NO_DATA'
call SEND
Return

   /* SEARCH PROCEDURE USING PUBLICATIN TITLE AS A PARAMETER */
SRCH_PRT:
Prtq=MSG.3
call common prtq';;'3
Return
   /* SEARCH PROCEDURE USING LANGUAGE DATA AS A PARAMETER */
SRCH_LANG:
arg which
Langq=MSG.3
call common langq';;'4
Return



COMMON:
Arg com_arg
parse var com_arg key';;'loc
Do type=1 to 6
   If which='SOME' then if type^=MSG.4 then iterate type
   k=0  ; drop num. line.
```

```
   Do forever
     'EXECIO * DISKR' file.type '(LOCATE /'key'/'
      If rc^=0 then leave
      K=k+1
      Pull rel.k num.k  ; Parse pull line.k
   End
   Do i=1 to k
      drop line_f line_s line_t
      If ^(substr(line.i,1,1)='15'X  &  FNDP(loc';;'line.i';;'key)='OK')
                                      then iterate i
      l=num.i
     'EXECIO 1 DISKR' file.type l+1 '(VAR LINE_F'
     'EXECIO 1 DISKR' file.type l+2 '(VAR LINE_S'
      call collect type
      call collect left(line.i,247)
      call collect left(line_f,247)
      call collect left(line_s,247)
      If type=1 then Do
                     'EXECIO 1 DISKR' file.type l+3 '(VAR LINE_T'
                      If Rc=0 & substr(line_t,1,1)^=e then
                      call collect left(line_t,247)
                     End
      call collect '15'X
      Flag = 1
   End
'FINIS' file.type
End
If flag=0 then call collect 'NO_DATA'
call SEND
Return

    /* SEARCH PROCEDURE USING YEAR DATA AS A PARAMETER */
SRCH_YEAR:
arg which
yearq=MSG.3
plc.1=2  ;  plc.2=6  ;  plc.3=2  ;  plc.4=2  ;  plc.5=3  ;  plc.6=1
Do type=1 to 6
   If which='SOME' then if type^=MSG.4 then iterate type
   k=0  ; drop num. line.
   Do forever
     'EXECIO * DISKR' file.type '(LOCATE /'yearq'/'
      If rc^=0 then leave
      K=k+1
      Pull rel.k num.k  ; Parse pull line.k
   End
   Do i=1 to k
      drop line_f line_s line_t   ;    l=num.i
     'EXECIO 1 DISKR' file.type l-2 '(VAR LINE_F'
      If ^(substr(line_f,1,1)='15'X  &  FNDP(plc.type';;'line.i';;'yearq)='OK')
                                      then iterate i
     'EXECIO 1 DISKR' file.type l-1 '(VAR LINE_S'
      call collect type
      call collect left(line_f,247)
      call collect left(line_s,247)
      call collect left(line.i,247)
```

```
      If type=1 then Do
                     'EXECIO 1 DISKR' file.type l+1 '(VAR LINE_T'
                      If Rc=0 & substr(line_t,1,1)^=e then
                      call collect left(line_t,247)
                     End
      call collect '15'X
      Flag = 1
   End
'FINIS' file.type
End
If flag=0 then call collect 'NO_DATA'
call SEND
Return

SRCH_SCOD:
arg which
scodq=MSG.3
Do type=1 to 6
   If which='SOME' then if type^=MSG.4 then iterate type
   k=0  ; drop num. line.
   Do forever
     'EXECIO * DISKR' file.type '(LOCATE /'scodq'/'
      If rc^=0 then leave
      K=k+1
      Pull rel.k num.k  ; Parse pull line.k
   End
   Do i=1 to k
      drop line_f line_s line_t
      If ^(substr(line.i,1,1)='15'X  &  FNDS(line.i';;'scodq)='OK')
                                     then iterate i
      l=num.i
     'EXECIO 1 DISKR' file.type l+1 '(VAR LINE_F'
     'EXECIO 1 DISKR' file.type l+2 '(VAR LINE_S'
      call collect type
      call collect left(line.i,247)
      call collect left(line_f,247)
      call collect left(line_s,247)
      If type=1 then Do
                     'EXECIO 1 DISKR' file.type l+3 '(VAR LINE_T'
                      If Rc=0 & substr(line_t,1,1)^=e then
                      call collect left(line_t,247)
                     End
      call collect '15'X
      Flag = 1
   End
'FINIS' file.type
End
If flag=0 then call collect 'NO_DATA'
call SEND
Return


COLLECT:
arg line_to_send
ind=ind+1
sdata.ind = line_to_send
Return
```

```
                    /* SENDING THE LOCATED DATA TO THE USER */
SEND:
Do iz=1 to ind
   'IUCV SEND' user sdata.iz
End
Return

FNDX: Procedure
      Arg arg_comp
      ret='NO_OK'
      Parse var arg_comp str';;'str_f
      S=0
      Do while str^=''
         S = s+1
         Parse var str a '15'x str
         If pos(str_f,a)^=0 then if s//4=3 then do; ret='OK'; leave; end
      End
      Return ret

FNDS: Procedure
      Arg arg_comp
      ret='NO_OK'
      Parse var arg_comp str';;'str_f
      S=0
      Do while str^=''
         S = s+1
         Parse var str a '15'x str
         If pos(str_f,a)^=0 then if s>4 & s<10 then do; ret='OK'; leave; end
      End
      Return ret

FNDP: Procedure
      Arg arg_comp
      ret='NO_OK'
      Parse var arg_comp pl';;'str';;'str_f
      S=0
      Do while str^=''
         S = s+1
         Parse var str a '15'x str
         If pos(str_f,a)^=0 then if s=pl then do; ret='OK'; leave; end
      End
      Return ret

 SYNTAX:
 Rs=Rc
 say sigl 'nolu satırda HATA var!'
 'IUCV SEVER *MSG'
 exit Rs
```

```
                    /*  USER PROGRAM  */
trace o
ADDRESS COMMAND
'SET CMSTYPE HT'
'NUCXDROP *'
Call DEL_ALL
Signal on ERROR
Dbase_vm = 'KBBYAYIN'
Parse source . . fn .
'IUCV LOAD'
'IUCV CONNECT' dbase_vm
'SET CMSTYPE RT'

                /* DEFINING HEADERS */
Q=x2c('FC')
Hdr.1='  İSTANBUL TEKNİK ÜNİVERSİTESİ  '
Hdr.2='  KONTROL VE BİLGİSAYAR BÖLÜMÜ  '
Hdr.3='ARAŞTIRMA VE YAYIN BİLGİ BANKASI'

Hds.1='(1) - YAYIN GİRİŞİ'
Hds.2='(2) - YAYIN TARAMA'

T.2 ='YAYIN GİRİŞ FORMU'
T.3 ='Yayın Adı: '
T.4 ='Ünvanlar'
T.5 ='(01)-Prof.Dr.  (02)-Prof. (03)-Doç.Dr.   (04)-Doç.  (05)-Y.Doç.  '
T.6 ='(06)-Y.Doç.Dr. (07)-Dr.   (08)-Araş.Gör. (09)-Uzman (10)-Öğr.Gör.'
T.7 ='                                                (00)-Diğer    '
T.8 ='Yazarlar'
T.9 ='Ünvan          Adı              SOYADI        Kurumu (İTÜ ise Sicil No)'
T.10='Yayın Cinsi:'
T.11='  (1)-Makale   (2)-Bildiri     (3)-Ders Notu   (4)-Kitap'
T.12='                (5)-Rapor       (6)-Y.Lis.Tezi  (7)-Doktora Tezi'
T.13='Yayın Dili :'
T.14='  (1)-Almanca  (3)-Farsça      (5)-Ingilizce   (7)-Italyanca'
T.15='  (2)-Arapça   (4)-Fransızca   (6)-Ispanyolca  (8)-Rusça'
T.16='                                               (0)-TÜRKÇE'
T.17='Yayının Konusu ile ilgili Doç.Bilim Dalı Kodları'

Ta.0='Yayın cinsi 'q'BİLİMSEL MAKALE'q' ise ,'
Ta.1='Yayınlandığı Periyodiğin Adı   :'
Ta.2='Periyodiğin Yıl, Cilt, Sayı, Sayfa No.ları'
Ta.3='Periyodik 'q'Science Citation Index'q' içinde ise'
Ta.4='Periyodiğin Index"teki Kodu     :'
Ta.5='Çeviri Makale ise Orijinal Adı :'
Ta.6='Orijinal Makalenin Yayınlandığı Periyodun Adı'
Ta.7='Yayın Yılı                      :'

Tn.0='Yayın cinsi 'q'YAYINLANMIŞ BİLDİRİ'q' ise ,'
Tn.1='Bildirinin verildiği Kongre veya Sempozyum:'
Tn.2='Düzenleyen Kurum                :'
Tn.3='Düzenlendiği Şehir / Ülke       :'
Tn.4='Bildirinin Yayınlandığı Yayının Adı:'
Tn.5='Yılı / Sayfa No.ları            :'

To.0='Yayın cinsi 'q'DERS NOTU'q' ise ,'
To.1='Yayınlayan Kurum :'
```

```
To.2='Yayın Yılı        :'
To.3='Kaçıncı Baskı     :'
To.4='Sayfa Sayısı      :'

Tb.0='Yayın cinsi 'q'KİTAP'q' ise ,'
Tb.1='Yayınevi veya Yayınlayan Kurum :'
Tb.2='Yayın Yılı                     :'
Tb.3='Kaçıncı Baskı                  :'
Tb.4='Sayfa Sayısı                   :'
Tb.5='Çeviri ise Orijinal Adı        :'

Tr.0='Yayın cinsi 'q'YAYINLANMIŞ RAPOR'q' ise ,'
Tr.1='Yayınlayan Kurum :'
Tr.2='Yayınlanma Şekli :'
Tr.3='Baskı ==> 1     Teksir ==> 2'
Tr.4='Yayın Yılı        :'
Tr.5='Sayfa Sayısı      :'

Tt.0='Yayın cinsi 'q'Y.LİSANS TEZI'q' veya 'q'DOKTORA TEZI'q' ise ,'
Tt.1='Tezin Kabul Edildiği Yıl :'
Tt.2='Öğrenci                  :'
Tt.3='<1> ==> İTÜ"den'
Tt.4='<2> ==> Başka Üniversiteden'
Tt.5='Ilgili Enstitü           :'

Ts.0='YAYIN TARAMA KRİTERLERİ'
Ts.1='1 ==> Yazar Soyadına göre'
Ts.2='2 ==> Yayın Adına göre'
Ts.3='3 ==> Yayın Yılına göre'
Ts.4='4 ==> Yayın Tipi ve Yılına göre'
Ts.5='5 ==> Yayın Diline göre'
Ts.6='6 ==> Yayın Tipi ve Diline göre'
Ts.7='7 ==> Doçentlik Koduna göre'

query=': Veri Tabanını sorgula'
menuq=': Tarama menüsü'
Save=': Yazılanları kaydet'
Menup=': Ana menüye dön'
Menum=': Ana menü'
next=  ': Sonraki sayfa'
back=  ': Önceki sayfa'
Pf='PF12'  ;  Cr='ENTER'  ;  Pfr='PF10' ;  Pfb='PF7' ; Pff='PF8'

                /* DEFINING SCREENS */
'DEFINE VSCREEN MAIN 24 80 0 0'
'DEFINE WINDOW  MAIN 24 80 1 1'
'DEFINE VSCREEN SCND 24 80 0 0'
'DEFINE WINDOW  SCND 24 80 1 1'
'DEFINE VSCREEN THRD 24 80 0 0'
'DEFINE WINDOW  THRD 24 80 1 1'
'DEFINE VSCREEN FRTH 99 80 2 2'
'DEFINE WINDOW  FRTH 24 80 1 1'
'DEFINE VSCREEN FIVE 24 80 0 0'
'DEFINE WINDOW  FIVE 24 80 1 1'
'DEFINE VSCREEN LAST 24 80 0 1'
'DEFINE WINDOW  LAST 24 80 1 1'
'SET LOCATION FRTH OFF'
```

```
'SET LOCATION LAST OFF'
Call WIN1 MAIN
'WRITE VSC MAIN 10 23' length(Hdr.1)+1 '(PRO HI FIELD' Hdr.1
'WRITE VSC MAIN 12 23' length(Hdr.2)+1 '(PRO HI FIELD' Hdr.2
'WRITE VSC MAIN 14 23' length(Hdr.3)+1 '(PRO HI FIELD' Hdr.3
'REFRESH' ; 'CP SLEEP 1 SEC'

MENU:
Call WIN1 MAIN
'WRITE VSC MAIN 10 30' length(Hdr.1)+1 '(PRO HI FIELD' Hds.1
'WRITE VSC MAIN 12 30' length(Hdr.2)+1 '(PRO HI FIELD' Hds.2
'WRITE VSC MAIN 15 31 2 (FIELD _'
'WRITE VSC MAIN 24 1 13 (PRO HI FIELD PF12 : ÇIKIŞ'
RE_INPUT:
Drop waitread.
'CURSOR VSC MAIN 15 32'
'WAITREAD VSC MAIN'
If waitread.1='PFKEY 12' then signal EX
Parse var waitread.3 . . . key
Select
  When key=1 then signal RECORD
  When key=2 then signal SEARCH
  Otherwise Do
              txt='LÜTFEN  <1> VEYA <2> SEÇENEKLERİNDEN BİRİNİ GİRİNİZ'
              'WRITE VSC MAIN 20 15 52 (HI PRO FIELD' txt
              'ALARM VSC MAIN'
              Signal RE_INPUT
            End
End
RECORD:
Drop prtitle prtitl. title. name. surname. regno. lang type scode.
Nodrop='NO'
Call WIN1 MAIN
'WRITE VSC MAIN 1  25' length(T.2)+1 '(PRO HI FIELD' T.2
'WRITE VSC MAIN 4   3' length(T.3)+1 '(PRO HI FIELD' T.3
'WRITE VSC MAIN 4  14 66 (FIELD' copies('_',65)
'WRITE VSC MAIN 5  14 66 (FIELD' copies('_',65)
'WRITE VSC MAIN 6  14 31 (FIELD' copies('_',30)
'WRITE VSC MAIN 7   3' length(T.4)+1 '(PRO HI FIELD' T.4
'WRITE VSC MAIN 8  12' length(T.5)+1 '(PRO FIELD' T.5
'WRITE VSC MAIN 9  12' length(T.6)+1 '(PRO FIELD' T.6
'WRITE VSC MAIN 10 12' length(T.7)+1 '(PRO FIELD' T.7
'WRITE VSC MAIN 10  3' length(T.8)+1 '(PRO HI FIELD' T.8
'WRITE VSC MAIN 11 11' length(T.9)+1 '(PRO FIELD' T.9
 Call genpf MAIN
Do i=1 to 5
   'WRITE VSC MAIN' i+11 ' 8  3 (PRO HI FIELD' i||'.'
   'WRITE VSC MAIN' i+11 '13  3 (FIELD __'
   'WRITE VSC MAIN' i+11 '18 16 (FIELD' copies('_',15)
   'WRITE VSC MAIN' i+11 '35 18 (FIELD' copies('_',17)
   'WRITE VSC MAIN' i+11 '54 26 (FIELD' copies('_',25)
   'WRITE VSC MAIN 23' i*10-6 '8 (FIELD _______'
End
   'WRITE VSC MAIN 18  3 13 (PRO HI FIELD' T.10
   'WRITE VSC MAIN 18 16 2 (FIELD _'
   'WRITE VSC MAIN 18 18'  length(T.11)+1 '(PRO FIELD' T.11
   'WRITE VSC MAIN 19 18'  length(T.12)+1 '(PRO FIELD' T.12
```

```
   'WRITE VSC MAIN 20  3 13 (PRO HI FIELD' T.13
   'WRITE VSC MAIN 20 16 2 (FIELD _'
   'WRITE VSC MAIN 20 18'  length(T.14)+1 '(PRO FIELD' T.14
   'WRITE VSC MAIN 21 18'  length(T.15)+1 '(PRO FIELD' T.15
   'WRITE VSC MAIN 22 18'  length(T.16)+1 '(PRO FIELD' T.16
   'WRITE VSC MAIN 22  3'  length(T.17)+1 '(PRO HI FIELD' T.17

                /* READING SCREEN */
first='YES'
'CURSOR VSC MAIN 4 15'
READ:
if prtitle^='PRTITLE' then
 Do
   'WRITE VSC MAIN  4 14 66 (FIELD' substr(prtitle,1,65,'_')
   'WRITE VSC MAIN  5 14 66 (FIELD' substr(prtitle,66,65,'_')
   'WRITE VSC MAIN  6 14 31 (FIELD' substr(prtitle,131,30,'_')
 End
  Do i=1 to 5
 if title.i^='TITLE.'!!i then
    'WRITE VSC MAIN' i+11 '13 3  (FIELD' substr(title.i,1,2,'_')
 if name.i^='NAME.'!!i then
    'WRITE VSC MAIN' i+11 '18 16 (FIELD' substr(name.i,1,15,'_')
 if surname.i^='SURNAME.'!!i then
    'WRITE VSC MAIN' i+11 '35 18 (FIELD' substr(surname.i,1,17,'_')
 if regno.i^='REGNO.'!!i then
    'WRITE VSC MAIN' i+11 '54 26 (FIELD' substr(regno.i,1,25,'_')
  End
 if type^='TYPE' then
  'WRITE VSC MAIN 18 16 2 (FIELD' substr(type,1,1,'_')
 if lang^='LANG' then
  'WRITE VSC MAIN 20 16 3 (FIELD' substr(lang,1,1,'_')
  Do i=1 to 5
   if scode.i^='SCODE.'!!i then
    'WRITE VSC MAIN 23' i*10-6 '8 (FIELD' substr(scode.i,1,7,'_')
  End
If first^='YES' then 'ALARM VSC MAIN'
Drop waitread.
'WAITREAD VSC MAIN'
first='NO'
If waitread.1='PFKEY 12' then signal MENU
'WRITE VSC MAIN 2 1 80 (PRO FIELD  '
Do i=3 to waitread.0
   Parse var waitread.i 'DATA' x y z
   z = strip(strip(z,,'_'))
   Select
     When (x= 4) then prtitl.1=z
     When (x= 5) then prtitl.2=z
     When (x= 6) then prtitl.3=z
     When (x=12) then select
                        When (y=13) then title.1 = z
                        When (y=18) then name.1 = tran(z)
                        When (y=35) then surname.1 = tran(z)
                        When (y=54) then regno.1 = z
                      end
```

```
    When (x=13) then select
                    When (y=13) then title.2 = z
                    When (y=18) then name.2 = tran(z)
                    When (y=35) then surname.2 = tran(z)
                    When (y=54) then regno.2 = z
                   end
    When (x=14) then select
                    When (y=13) then title.3 = z
                    When (y=18) then name.3 = tran(z)
                    When (y=35) then surname.3 = tran(z)
                    When (y=54) then regno.3 = z
                   end
    When (x=15) then select
                    When (y=13) then title.4 = z
                    When (y=18) then name.4 = tran(z)
                    When (y=35) then surname.4 = tran(z)
                    When (y=54) then regno.4 = z
                   end
    When (x=16) then select
                    When (y=13) then title.5 = z
                    When (y=18) then name.5 = tran(z)
                    When (y=35) then surname.5 = tran(z)
                    When (y=54) then regno.5 = tran(z)
                   end
    When (x=18) then type = z
    When (x=20) then lang = z
    When (x=23) then select
                    When (y= 4) then scode.1 = z
                    When (y=14) then scode.2 = z
                    When (y=24) then scode.3 = z
                    When (y=34) then scode.4 = z
                    When (y=44) then scode.5 = z
                   end
  End
End
prtitle=''
Do i=1 to 3
  if prtitl.i^='PRTITL.'!!i then prtitle=prtitle!!prtitl.i
End

                /* ERROR CONTROL */
If prtitle = '' then
  Do
    'WRITE VSC MAIN  2  1 55 (PRO HI FIELD *** Yayın Adını Giriniz ***'
    'CURSOR VSC MAIN 4 15'
    Signal READ
  End

 msgn='. Yazarın Adını Giriniz ***'
 msgs='. Yazarın Soyadını Giriniz ***'
 msgr='. Yazarın Kurum veya Sicil Nosunu Giriniz ***'
 msgt='. Yazarın Ünvanını Giriniz ***'
drop msg
```

```
Do i=1 to 5
   Select
     When title.i^='TITLE.'||i then
          Do
            if title.i<0 | title.i>10 | verify(title.i,'0123456789')^=0
              then Do
                     'WRITE VSC MAIN  2  1 55 (PRO HI FIELD',
                          ||i||'. Yazarın Ünvanı Hatalı! Tekrar Giriniz ***'
                   'CURSOR VSC MAIN' 11+i '14'  ;  Signal READ
                 End
                     Select
                       When name.i='NAME.'||i then msg=||msgn
                       When surname.i='SURNAME.'||i then msg=||msgs
                       When regno.i='REGNO.'||i then msg=||msgr
                       Otherwise nop
                     End
          End
     When name.i^='NAME.'||i then
                     Select
                       When title.i='TITLE.'||i then msg=||msgt
                       When surname.i='SURNAME.'||i then msg=||msgs
                       When regno.i='REGNO.'||i then msg=||msgr
                       Otherwise nop
                     End
     When surname.i^='SURNAME.'||i then
                     Select
                       When title.i='TITLE.'||i then msg=||msgt
                       When name.i='NAME.'||i then msg=||msgn
                       When regno.i='REGNO.'||i then msg=||msgr
                       Otherwise nop
                     End
     When regno.i^='REGNO.'||i then
                     Select
                       When title.i='TITLE.'||i then msg=||msgt
                       When name.i='NAME.'||i then msg=||msgn
                       When surname.i='SURNAME.'||i then msg=||msgs
                       Otherwise nop
                     End
     Otherwise if i=1 then msg='*** Hiç bir Yazar Bilgisi Girmediniz! ***'
   End
     If msg^='MSG' then
        Do
          'WRITE VSC MAIN  2  1 55 (PRO HI FIELD' msg
          Select
            when substr(msg,2)=msgt then
                    'CURSOR VSC MAIN' 11+substr(msg,1,1) '14'
            when substr(msg,2)=msgn then
                    'CURSOR VSC MAIN' 11+substr(msg,1,1) '19'
            when substr(msg,2)=msgs then
                    'CURSOR VSC MAIN' 11+substr(msg,1,1) '36'
            when substr(msg,2)=msgr then
                    'CURSOR VSC MAIN' 11+substr(msg,1,1) '55'
            otherwise 'CURSOR VSC MAIN 12 14'
          End
          Signal READ
        End
End
```

```
If type = 'TYPE' then
   Do
     'WRITE VSC MAIN  2  1 55 (PRO HI FIELD *** Yayın Cinsini Giriniz ***'
     'CURSOR VSC MAIN 18 17'
     Signal READ
   End
                else if type='' ! verify(type,'1234567')^=0 then
   Do
     'WRITE VSC MAIN  2  1 55 (PRO HI FIELD *** Yayın Cinsi Hatalı! Tekrar Giriniz
***'
     'CURSOR VSC MAIN 18 17'
     Signal READ
   End

If lang = 'LANG' then
   Do
     'WRITE VSC MAIN  2  1 55 (PRO HI FIELD *** Yayın Dilini Giriniz ***'
     'CURSOR VSC MAIN 20 17'
     Signal READ
   End
          else if lang='' ! verify(lang,'012345678')^=0 then
   Do
     'WRITE VSC MAIN  2  1 55 (PRO HI FIELD *** Yayın Dili Hatalı! Tekrar Giriniz
***'
     'CURSOR VSC MAIN 20 17'
     Signal READ
   End
                  /* Finding the number of science codes */
maxs=0
Do i=1 to 5
  if scode.i^='SCODE.'!!i then
   if datatype(scode.i)^='NUM' then
    Do
     'WRITE VSC MAIN  2  1 55 (PRO HI FIELD *** Doç.Bilim Kodu sayısal bilgi olmalı
***'
     'CURSOR VSC MAIN 23' i*10-5
     Signal READ
    End
                             else maxs=maxs+1
End
If maxs <1 then
   Do
     'WRITE VSC MAIN  2  1 55 (PRO HI FIELD *** Konuyla ilgili min. bir Doç.Bilim
Kodu girin ***'
     'CURSOR VSC MAIN 23 5' ;  Signal READ
   End
                  /* Finding the number of authors */
maxi=0
Do i=1 to 5
   if title.i^='TITLE.'!!i then maxi=maxi+1
End
          /* SELECTING THE PUBLICATION TYPE */
Select
  When type=1 then signal ARTICLE
  When type=2 then signal NOTICE
  When type=3 then signal NOTE
  When type=4 then signal BOOK
```

```
  When type=5 then signal REPORT
  otherwise signal THESIS
End

ARTICLE:
If nodrop^='YES' then Do
                    Drop per year pervol perno perpg1 perpg2 percode
                    Drop pe. orar. orpe. orart orper oryear
                   End
Call WIN1 SCND
'WRITE VSC SCND  1 34' length(ta.0)+1 '(PRO FIELD' ta.0
'WRITE VSC SCND  1 46' length(ta.0)-17 '(PRO HI FIELD' substr(ta.0,13,length(ta.0)-18)
'WRITE VSC SCND  4  1 33 (PRO HI FIELD' ta.1
'WRITE VSC SCND  4 34 47 (FIELD' copies('_',46)
'WRITE VSC SCND  5 34 47 (FIELD' copies('_',46)
'WRITE VSC SCND  7  1 43 (PRO HI FIELD' ta.2
'WRITE VSC SCND  8 34  5 (FIELD 19'||'__'
'WRITE VSC SCND  8 41  4 (FIELD ___'
'WRITE VSC SCND  8 47  5 (FIELD ____'
'WRITE VSC SCND  8 55  5 (FIELD ____'
'WRITE VSC SCND  8 61  5 (FIELD ____'
'WRITE VSC SCND 10  1' length(ta.3)+1 '(PRO HI FIELD' ta.3
'WRITE VSC SCND 11  1 33 (PRO HI FIELD' ta.4
'WRITE VSC SCND 11 34 12 (FIELD __________'
'WRITE VSC SCND 13  1 33 (PRO HI FIELD' ta.5
'WRITE VSC SCND 13 34 47 (FIELD' copies('_',46)
'WRITE VSC SCND 14 34 47 (FIELD' copies('_',46)
'WRITE VSC SCND 15 34 47 (FIELD' copies('_',46)
'WRITE VSC SCND 16 34 47 (FIELD' copies('_',46)
'WRITE VSC SCND 18  1' length(ta.6)+1 '(PRO HI FIELD' ta.6
'WRITE VSC SCND 19 34 47 (FIELD' copies('_',46)
'WRITE VSC SCND 20 34 47 (FIELD' copies('_',46)
'WRITE VSC SCND 22  1 33 (PRO HI FIELD' ta.7
'WRITE VSC SCND 22 34  5 (FIELD 19'||'__'  ; Call Pfs
                /* READING SCREEN */
'CURSOR VSC SCND 4 35'
READ_ART:
if per^='PER' then
 Do
   'WRITE VSC SCND  4 34 47 (FIELD' substr(per,1,46,'_')
   'WRITE VSC SCND  5 34 47 (FIELD' substr(per,47,46,'_')
 End
 if year^='YEAR' then
  'WRITE VSC SCND 8 34 5 (FIELD' substr(year,1,4,'_')
 if pervol^='PERVOL' then
  'WRITE VSC SCND 8 41 4 (FIELD' substr(pervol,1,3,'_')
 if perno^='PERNO' then
  'WRITE VSC SCND 8 47 5 (FIELD' substr(perno,1,4,'_')
 if perpg1^='PERPG1' then
  'WRITE VSC SCND 8 55 5 (FIELD' substr(perpg1,1,4,'_')
 if perpg2^='PERPG2' then
  'WRITE VSC SCND 8 61 5 (FIELD' substr(perpg2,1,4,'_')
 if percode^='PERCODE' then
  'WRITE VSC SCND 11 34 12 (FIELD' substr(percode,1,11,'_')
 if orart^='ORART' then
  Do
    'WRITE VSC SCND 13 34 47 (FIELD' substr(orart,1,46,'_')
```

```
    'WRITE VSC SCND 14 34 47 (FIELD' substr(orart,47,46,'_')
    'WRITE VSC SCND 15 34 47 (FIELD' substr(orart,93,46,'_')
    'WRITE VSC SCND 16 34 47 (FIELD' substr(orart,139,46,'_')
  End
if orper^='ORPER' then
 Do
   'WRITE VSC SCND  19 34 47 (FIELD' substr(orper,1,46,'_')
   'WRITE VSC SCND  20 34 47 (FIELD' substr(orper,47,46,'_')
 End
 if oryear^='ORYEAR' then
  'WRITE VSC SCND 22 34 5 (FIELD' substr(oryear,1,4,'_')
'ALARM VSC SCND'
Drop waitread.
'WAITREAD VSC SCND'
If waitread.1='PFKEY 12' then signal MENU
Do i=3 to waitread.0
   Parse var waitread.i 'DATA' x y z
   z = strip(strip(z,,'_'))
   Select
     When (x= 4) then pe.1 = z
     When (x= 5) then pe.2 = z
     When (x= 8) then select
                        When (y=34) then year = z
                        When (y=41) then pervol = z
                        When (y=47) then perno = z
                        When (y=55) then perpg1 = z
                        When (y=61) then perpg2 = z
                      end
     When (x=11) then percode = z
     When (x=13) then orar.1 = z
     When (x=14) then orar.2 = z
     When (x=15) then orar.3 = z
     When (x=16) then orar.4 = z
     When (x=19) then orpe.1 = z
     When (x=20) then orpe.2 = z
     When (x=22) then oryear= z
   End
End
per = ''
Do i=1 to 2
   if pe.i^='PE.'||i then per=per||pe.i
End
orart = ''
Do i=1 to 4
   if orar.i^='ORAR.'||i then orart=orart||orar.i
End
orper = ''
Do i=1 to 2
   if orpe.i^='ORPE.'||i then orper=orper||orpe.i
End
If waitread.1='PFKEY 10' then call prev
                  /* ERROR CONTROL */
If per = '' then
   Do
     'WRITE VSC SCND  2  1 55 (PRO HI FIELD *** Periyodiğin Adını Giriniz ***'
     'CURSOR VSC SCND 4 35'
     Signal READ_ART
```

```
   End


If (length(strip(year)) ^= 4) ! (datatype(year) ^= 'NUM') then
   Do
     'WRITE VSC SCND  2  1 55 (PRO HI FIELD *** Yayın Yılı Hatalı! Tekrar Giriniz
***'
     'CURSOR VSC SCND 8 37'
     Signal READ_ART
   End
If pervol = 'PERVOL' then
   Do
     'WRITE VSC SCND  2  1 55 (PRO HI FIELD *** Periyodiğin Cilt No.sunu Giriniz
***'
     'CURSOR VSC SCND 8 42'
     Signal READ_ART
   End
If perno = 'PERNO' then
   Do
     'WRITE VSC SCND  2  1 55 (PRO HI FIELD *** Periyodiğin Sayısını Giriniz ***'
     'CURSOR VSC SCND 8 48'
     Signal READ_ART
   End
If (perpg1 = 'PERPG1') ! (perpg2 = 'PERPG2') then
   Do
     'WRITE VSC SCND  2  1 55 (PRO HI FIELD *** Periyodiğin Sayfa No.sunu Giriniz
***'
     If perpg1='PERPG1' then 'CURSOR VSC SCND 8 56'
       else If perpg2='PERPG2' then 'CURSOR VSC SCND 8 62'
     Signal READ_ART
   End
If (datatype(perpg1) ^= 'NUM') ! (datatype(perpg2) ^= 'NUM') then
   Do
     'WRITE VSC SCND  2  1 55 (PRO HI FIELD *** Periyodiğin Sayfa No.su Hatalı!
Tekrar Giriniz ***'
     If datatype(perpg1)^='NUM' then 'CURSOR VSC SCND 8 56'
       else if datatype(perpg2)^='NUM' then 'CURSOR VSC SCND 8 62'
     Signal READ_ART
   End
If (oryear^='ORYEAR') ! (orart^='') ! (orper^='') then
 Do
  If orart = '' then
   Do
     'WRITE VSC SCND  2  1 55 (PRO HI FIELD *** Çeviri Makalenin Orijinal Adını
Giriniz ***'
     'CURSOR VSC SCND 13 35'
     Signal READ_ART
   End
  If orper = '' then
   Do
     'WRITE VSC SCND  2  1 55 (PRO HI FIELD  *** Orijinal Periyodiğin Adını Giriniz
***'
     'CURSOR VSC SCND 19 35'
     Signal READ_ART
   End
  If (length(strip(oryear)) ^= 4) ! (datatype(oryear) ^= 'NUM') then
```

```
   Do
     'WRITE VSC SCND  2  1 55 (PRO HI FIELD *** Yayın Yılı Hatalıl Tekrar Giriniz
***'
     'CURSOR VSC SCND 22 37'
     Signal READ_ART
   End
 End
Signal PROCESS


NOTICE:
If nodrop^='YES' then
 Do
  Drop symp sym. ins. inst city country notice notic. year notpgl notpg2
 End
Call WIN1 SCND
'WRITE VSC SCND  1 25' length(tn.0)+1 '(PRO FIELD' tn.0
'WRITE VSC SCND  1 37' length(tn.0)-17 '(PRO HI FIELD' substr(tn.0,13,length(tn.0)-18)
'WRITE VSC SCND  5  1 44 (PRO HI FIELD' tn.1
'WRITE VSC SCND  6 38 43 (FIELD' copies('_',42)
'WRITE VSC SCND  7 38 43 (FIELD' copies('_',42)
'WRITE VSC SCND 10  1 37 (PRO HI FIELD' tn.2
'WRITE VSC SCND 10 38 43 (FIELD' copies('_',42)
'WRITE VSC SCND 11 38 43 (FIELD' copies('_',42)
'WRITE VSC SCND 14  1 44 (PRO HI FIELD' tn.3
'WRITE VSC SCND 14 38 19 (FIELD' copies('_',18)
'WRITE VSC SCND 14 57  4 (PRO HI FIELD' ' / '
'WRITE VSC SCND 14 61 20 (FIELD' copies('_',19)
'WRITE VSC SCND 17  1 37 (PRO HI FIELD' tn.4
'WRITE VSC SCND 17 38 43 (FIELD' copies('_',42)
'WRITE VSC SCND 18 38 43 (FIELD' copies('_',42)
'WRITE VSC SCND 21  1 37 (PRO HI FIELD' tn.5
'WRITE VSC SCND 21 38  5 (FIELD 19__'
'WRITE VSC SCND 21 45  2 (PRO HI FIELD /'
'WRITE VSC SCND 21 49  5 (FIELD ____'
'WRITE VSC SCND 21 55  5 (FIELD ____'
Call Pfs
                /* READING SCREEN */
'CURSOR VSC SCND 6 39'
READ_NTC:
 if symp^='SYMP' then
 Do
   'WRITE VSC SCND 6 38 43 (FIELD' substr(symp,1,42,'_')
   'WRITE VSC SCND 7 38 43 (FIELD' substr(symp,43,42,'_')
 End
 if inst^='INST' then
 Do
   'WRITE VSC SCND 10 38 43 (FIELD' substr(inst,1,42,'_')
   'WRITE VSC SCND 11 38 43 (FIELD' substr(inst,43,42,'_')
 End
 if city^='CITY' then
  'WRITE VSC SCND 14 38 19 (FIELD' substr(city,1,18,'_')
 if country^='COUNTRY' then
  'WRITE VSC SCND 14 61 20 (FIELD' substr(country,1,19,'_')
 if notice^='NOTICE' then
 Do
```

```
   'WRITE VSC SCND 17 38 43 (FIELD' substr(notice,1,42,'_')
   'WRITE VSC SCND 18 38 43 (FIELD' substr(notice,43,42,'_')
 End
 if year^='YEAR' then
  'WRITE VSC SCND 21 38 5 (FIELD' substr(year,1,4,'_')
 if notpg1^='NOTPG1' then
  'WRITE VSC SCND 21 49 5 (FIELD' substr(notpg1,1,4,'_')
 if notpg2^='NOTPG2' then
  'WRITE VSC SCND 21 55 5 (FIELD' substr(notpg2,1,4,'_')
'ALARM VSC SCND'
Drop waitread.
'WAITREAD VSC SCND'
If waitread.1='PFKEY 12' then signal MENU
Do i=3 to waitread.0
   Parse var waitread.i 'DATA' x y z
   z = strip(strip(z,,'_'))
   Select
     When (x= 6) then sym.1=z
     When (x= 7) then sym.2=z
     When (x=10) then ins.1=z
     When (x=11) then ins.2=z
     When (x=14) then select
                        When (y=38) then city=z
                        When (y=61) then country=z
                      end
     When (x=17) then notic.1=z
     When (x=18) then notic.2=z
     When (x=21) then select
                        When (y=38) then year=z
                        When (y=49) then notpg1=z
                        When (y=55) then notpg2=z
                      end
   End
End
symp = ''
Do i=1 to 2
   if sym.i^='SYM.'!!i then symp=symp!!sym.i
End
inst = ''
Do i=1 to 2
   if ins.i^='INS.'!!i then inst=inst!!ins.i
End
notice = ''
Do i=1 to 2
   if notic.i^='NOTIC.'!!i then notice=notice!!notic.i
End
If waitread.1='PFKEY 10' then call prev
                  /* ERROR CONTROL */
If symp = '' then
   Do
     'WRITE VSC SCND  3  1 55 (PRO HI FIELD *** Kongre veya Sempozyumun Adını
Giriniz ***'
     'CURSOR VSC SCND 6 39'
     Signal READ_NTC
   End

If inst = '' then
```

```
   Do
     'WRITE VSC SCND  3  1 55 (PRO HI FIELD *** Düzenleyen Kurumun Adını Giriniz
***'
     'CURSOR VSC SCND 10 39'
     Signal READ_NTC
   End
If city = 'CITY' then
   Do
     'WRITE VSC SCND  3  1 55 (PRO HI FIELD *** Düzenlendiği Şehrin Adını Giriniz
***'
     'CURSOR VSC SCND 14 39'
     Signal READ_NTC
   End

If country= 'COUNTRY' then
   Do
      'WRITE VSC SCND  3  1 55 (PRO HI FIELD Düzenlendiği Ülkenin Adını Giriniz
***'
     'CURSOR VSC SCND 14 62'
     Signal READ_NTC
   End

If notice = '' then
   Do
     'WRITE VSC SCND  3  1 55 (PRO HI FIELD *** Bildirinin Yayınlandığı Yayın Adını
Giriniz ***'
     'CURSOR VSC SCND 17 39'
     Signal READ_NTC
   End
If (length(strip(year)) ^= 4) ! (datatype(year) ^= 'NUM') then
   Do
      'WRITE VSC SCND  3  1 55 (PRO HI FIELD *** Yayın Yılı Hatalıl Tekrar Giriniz
***'
     'CURSOR VSC SCND 21 41'
     Signal READ_NTC
   End

If (notpg1 = 'NOTPG1') ! (notpg2 = 'NOTPG2') then
   Do
     'WRITE VSC SCND  3  1 55 (PRO HI FIELD *** Yayının Sayfa No.sunu Giriniz ***'
     If notpg1='NOTPG1' then 'CURSOR VSC SCND 21 50'
        else If notpg2='NOTPG2' then 'CURSOR VSC SCND 21 56'
     Signal READ_NTC
   End
If (datatype(notpg1) ^= 'NUM') ! (datatype(notpg2) ^= 'NUM') then
   Do
     'WRITE VSC SCND  3  1 55 (PRO HI FIELD *** Yayının Sayfa No.su Hatalıl Tekrar
Giriniz ***'
     If datatype(notpg1)^='NUM' then 'CURSOR VSC SCND 21 50'
        else if datatype(notpg2)^='NUM' then 'CURSOR VSC SCND 21 56'
     Signal READ_NTC
   End
Signal PROCESS

NOTE:
If nodrop^='YES' then Do
                        Drop inst year print page ins.
```

```
                    End
Call WIN1 SCND
'WRITE VSC SCND  1 25' length(to.0)+1 '(PRO FIELD' to.0
'WRITE VSC SCND  1 37' length(to.0)-17 '(PRO HI FIELD' substr(to.0,13,length(to.0)-18)
'WRITE VSC SCND  5  1 19 (PRO HI FIELD' to.1
'WRITE VSC SCND  5 20 41 (FIELD' copies('_',40)
'WRITE VSC SCND  6 20 41 (FIELD' copies('_',40)
'WRITE VSC SCND  9  1 19 (PRO HI FIELD' to.2
'WRITE VSC SCND  9 20  5 (FIELD 19__'
'WRITE VSC SCND 12  1 19 (PRO HI FIELD' to.3
'WRITE VSC SCND 12 20  3 (FIELD __'
'WRITE VSC SCND 15  1 19 (PRO HI FIELD' to.4
'WRITE VSC SCND 15 20  5 (FIELD ____'
Call Pfs



                /* READING SCREEN */
'CURSOR VSC SCND 5 21'
READ_NOT:
 if inst^='INST' then
 Do
   'WRITE VSC SCND  5 20 41 (FIELD' substr(inst,1,40,'_')
   'WRITE VSC SCND  6 20 41 (FIELD' substr(inst,41,40,'_')
 End
 if year^='YEAR' then
  'WRITE VSC SCND  9 20 5 (FIELD' substr(year,1,4,'_')
 if print^='PRINT' then
  'WRITE VSC SCND 12 20 3 (FIELD' substr(print,1,2,'_')
 if page^='PAGE' then
  'WRITE VSC SCND 15 20 5 (FIELD' substr(page,1,4,'_')
'ALARM VSC SCND'
Drop waitread.
'WAITREAD VSC SCND'
If waitread.1='PFKEY 12' then signal MENU
Do i=3 to waitread.0
   Parse var waitread.i 'DATA' x y z
   z = strip(strip(z,,'_'))
   Select
     When (x= 5) then ins.1=z
     When (x= 6) then ins.2=z
     When (x= 9) then year=z
     When (x=12) then print=z
     When (x=15) then page=z
   End
End
inst = ''
Do i=1 to 2
   if ins.i^='INS.'!!i then inst=inst!!ins.i
End
If waitread.1='PFKEY 10' then call prev

                /* ERROR CONTROL */
If inst = '' then
   Do
```

```
     'WRITE VSC SCND  3  1 55 (PRO HI FIELD *** Yayınlayan Kurumun Adını Giriniz
***'
     'CURSOR VSC SCND  5 21'
     Signal READ_NOT
   End

If (length(strip(year)) ^= 4) | (datatype(year) ^= 'NUM') then
   Do
     'WRITE VSC SCND  3  1 55 (PRO HI FIELD *** Yayın Yılı Hatalı! Tekrar Giriniz
***'
     'CURSOR VSC SCND  9 23'
     Signal READ_NOT
   End

If print = 'PRINT' then
   Do
     'WRITE VSC SCND  3  1 55 (PRO HI FIELD *** Baskı Sayısını Giriniz ***'
     'CURSOR VSC SCND 12 21'
     Signal READ_NOT
   End

If (datatype(print) ^= 'NUM') then
   Do
     'WRITE VSC SCND  3  1 55 (PRO HI FIELD *** Baskı Sayısı Hatalı! Tekrar Giriniz
***'
     'CURSOR VSC SCND 12 21'
     Signal READ_NOT
   End
If page = 'PAGE' then
   Do
     'WRITE VSC SCND  3  1 55 (PRO HI FIELD *** Sayfa Sayısını Giriniz ***'
     'CURSOR VSC SCND 15 21'
     Signal READ_NOT
   End

If (datatype(page) ^= 'NUM') then
   Do
     'WRITE VSC SCND  3  1 55 (PRO HI FIELD *** Sayfa Sayısı Hatalı! Tekrar Giriniz
***'
     'CURSOR VSC SCND 15 21'
     Signal READ_NOT
   End
Signal PROCESS

BOOK:
If nodrop^='YES' then
Do
   Drop inst ins. year print page orbook orboo. oryear
End
Call WIN1 SCND
'WRITE VSC SCND  1 25' length(tb.0)+1 '(PRO FIELD' tb.0
'WRITE VSC SCND  1 37' length(tb.0)-17 '(PRO HI FIELD' substr(tb.0,13,length(tb.0)-18)
'WRITE VSC SCND  4  1 33 (PRO HI FIELD' tb.1
'WRITE VSC SCND  4 34 41 (FIELD' copies('_',40)
'WRITE VSC SCND  5 34 41 (FIELD' copies('_',40)
'WRITE VSC SCND  8  1 33 (PRO HI FIELD' tb.2
'WRITE VSC SCND  8 34  5 (FIELD 19__'
```

```
'WRITE VSC SCND 11  1 48 (PRO HI FIELD' tb.3
'WRITE VSC SCND 11 34  3 (FIELD __'
'WRITE VSC SCND 14  1 48 (PRO HI FIELD' tb.4
'WRITE VSC SCND 14 34  5 (FIELD ____'
'WRITE VSC SCND 17  1 33 (PRO HI FIELD' tb.5
'WRITE VSC SCND 17 34 41 (FIELD' copies('_',40)
'WRITE VSC SCND 18 34 41 (FIELD' copies('_',40)
'WRITE VSC SCND 19 34 41 (FIELD' copies('_',40)
'WRITE VSC SCND 20 34 41 (FIELD' copies('_',40)
'WRITE VSC SCND 22  1 33 (PRO HI FIELD' tb.2
'WRITE VSC SCND 22 34  5 (FIELD 19__'
Call Pfs
                    /* READING SCREEN */
'CURSOR VSC SCND 4 35'
READ_BOOK:
 if inst^='INST' then
 Do
   'WRITE VSC SCND  4 34 41 (FIELD' substr(inst,1,40,'_')
   'WRITE VSC SCND  5 34 41 (FIELD' substr(inst,41,40,'_')
 End
 if year^='YEAR' then
  'WRITE VSC SCND  8 34 5 (FIELD' substr(year,1,4,'_')

 if print^='PRINT' then
  'WRITE VSC SCND 11 34 3 (FIELD' substr(print,1,2,'_')
 if page^='PAGE' then
  'WRITE VSC SCND 14 34 5 (FIELD' substr(page,1,4,'_')
 if orbook^='ORBOOK' then
  Do
    'WRITE VSC SCND 17 34 41 (FIELD' substr(orbook,1,40,'_')
    'WRITE VSC SCND 18 34 41 (FIELD' substr(orbook,41,40,'_')
    'WRITE VSC SCND 19 34 41 (FIELD' substr(orbook,81,40,'_')
    'WRITE VSC SCND 20 34 41 (FIELD' substr(orbook,121,40,'_')
  End
 if oryear^='ORYEAR' then
  'WRITE VSC SCND 22 34 5 (FIELD' substr(oryear,1,4,'_')
'ALARM VSC SCND'
Drop waitread.
'WAITREAD VSC SCND'
If waitread.1='PFKEY 12' then signal MENU
Do i=3 to waitread.0
   Parse var waitread.i 'DATA' x y z
   z = strip(strip(z,,'_'))
   Select
     When (x= 4) then ins.1=z
     When (x= 5) then ins.2=z
     When (x= 8) then year=z
     When (x=11) then print=z
     When (x=14) then page=z
     When (x=17) then orboo.1=z
     When (x=18) then orboo.2=z
     When (x=19) then orboo.3=z
     When (x=20) then orboo.4=z
     When (x=22) then oryear=z
   End
End
inst = ''
```

```
Do i=1 to 2
   if ins.i^='INS.'!!i then inst=inst!!ins.i
End
orbook = ''
Do i=1 to 4
   if orboo.i^='ORBOO.'!!i then orbook=orbook!!orboo.i
End
If waitread.1='PFKEY 10' then call prev

                    /* ERROR CONTROL */
If inst = '' then
   Do
     'WRITE VSC SCND  2  1 55 (PRO HI FIELD *** Yayınlayan Kurumun Adını Giriniz
***'
     'CURSOR VSC SCND  4 35'
     Signal READ_BOOK
   End

If (length(strip(year)) ^= 4) ! (datatype(year) ^= 'NUM') then
   Do
     'WRITE VSC SCND  2  1 55 (PRO HI FIELD *** Yayın Yılı Hatalı! Tekrar Giriniz
***'
     'CURSOR VSC SCND  8 37'
     Signal READ_BOOK
   End

If print = 'PRINT' then
   Do
     'WRITE VSC SCND  2  1 55 (PRO HI FIELD *** Baskı Sayısını Giriniz ***'
     'CURSOR VSC SCND 11 35'
     Signal READ_BOOK
   End
If (datatype(print) ^= 'NUM') then
   Do
     'WRITE VSC SCND  2  1 55 (PRO HI FIELD *** Baskı Sayısı Hatalı! Tekrar Giriniz
***'
     'CURSOR VSC SCND 11 35'
     Signal READ_BOOK
   End
If page = 'PAGE' then
   Do
     'WRITE VSC SCND  2  1 55 (PRO HI FIELD *** Sayfa Sayısını Giriniz ***'
     'CURSOR VSC SCND 14 35'
     Signal READ_BOOK
   End

If (datatype(page) ^= 'NUM') then
   Do
     'WRITE VSC SCND  2  1 55 (PRO HI FIELD *** Sayfa Sayısı Hatalı! Tekrar Giriniz
***'
     'CURSOR VSC SCND 14 35'
     Signal READ_BOOK
   End

If (oryear ^= 'ORYEAR') ! (orbook ^= '')  then
Do
  If (length(strip(oryear)) ^= 4) ! (datatype(oryear) ^= 'NUM') then
```

```
   Do
     'WRITE VSC SCND  2  1 55 (PRO HI FIELD *** Yayın Yılı Hatalıl Tekrar Giriniz
***'
     'CURSOR VSC SCND 22 37'
     Signal READ_BOOK
   End
  If orbook = '' then
   Do
     'WRITE VSC SCND  2  1 55 (PRO HI FIELD *** Çeviri Kitabın Orijinal Adını
Giriniz ***'
     'CURSOR VSC SCND 17 35'
     Signal READ_BOOK
   End
End
Signal PROCESS

REPORT:
If nodrop^='YES' then Do
                      Drop inst ins. way year page
                     End
Call WIN1 SCND
'WRITE VSC SCND  1 25' length(tr.0)+1 '(PRO  FIELD' tr.0
'WRITE VSC SCND  1 37' length(tr.0)-17 '(PRO HI FIELD' substr(tr.0,13,length(tr.0)-18)
'WRITE VSC SCND  5  1 19 (PRO HI FIELD' tr.1
'WRITE VSC SCND  5 20 41 (FIELD' copies('_',40)
'WRITE VSC SCND  6 20 41 (FIELD' copies('_',40)
'WRITE VSC SCND  9  1 19 (PRO HI FIELD' tr.2
'WRITE VSC SCND  9 20  5 (FIELD _'
'WRITE VSC SCND  9 33' length(tr.3)+1 '(PRO FIELD' tr.3
'WRITE VSC SCND 12  1 19 (PRO HI FIELD' tr.4
'WRITE VSC SCND 12 20  5 (FIELD 19__'
'WRITE VSC SCND 15  1 19 (PRO HI FIELD' tr.5
'WRITE VSC SCND 15 20  5 (FIELD ____'
Call Pfs

                /* READING SCREEN */
'CURSOR VSC SCND 5 21'
READ_REP:
 if inst^='INST' then
 Do
   'WRITE VSC SCND  5 20 41 (FIELD' substr(inst,1,40,'_')
   'WRITE VSC SCND  6 20 41 (FIELD' substr(inst,41,40,'_')
 End
 if way^='WAY' then
  'WRITE VSC SCND  9 20 2 (FIELD' way
 if year^='YEAR' then
  'WRITE VSC SCND 12 20 5 (FIELD' substr(year,1,4,'_')
 if page^='PAGE' then
  'WRITE VSC SCND 15 20 5 (FIELD' substr(page,1,4,'_')
'ALARM VSC SCND'
Drop waitread.
'WAITREAD VSC SCND'
If waitread.1='PFKEY 12' then signal MENU
Do i=3 to waitread.0
   Parse var waitread.i 'DATA' x y z
   z = strip(strip(z,,'_'))
   Select
```

```
     When (x= 5) then ins.1=z
     When (x= 6) then ins.2=z
     When (x= 9) then way=z
     When (x=12) then year=z
     When (x=15) then page=z
   End
End
inst = ''
Do i=1 to 2
   if ins.i^='INS.'!i then inst=inst!!ins.i
End
If waitread.1='PFKEY 10' then call prev
                    /* ERROR CONTROL */
If inst = '' then
   Do
     'WRITE VSC SCND  3  1 55 (PRO HI FIELD *** Yayınlayan Kurumun Adını Giriniz
***'
     'CURSOR VSC SCND  5 21'
     Signal READ_REP
   End

If way = 'WAY' then
   Do
     'WRITE VSC SCND  3  1 55 (PRO HI FIELD *** Yayınlanma Şeklini Giriniz ***'
     'CURSOR VSC SCND  9 21'
     Signal READ_REP
   End



If (way ^= 1) & (way ^= 2) then
   Do
      'WRITE VSC SCND  3  1 55 (PRO HI FIELD *** Yayınlanma Şekli Hatalı! Tekrar
Giriniz ***'
     'CURSOR VSC SCND  9 21'
     Signal READ_REP
   End

If (length(strip(year)) ^= 4) | (datatype(year) ^= 'NUM') then
   Do
     'WRITE VSC SCND  3  1 55 (PRO HI FIELD *** Yayın Yılı Hatalı! Tekrar Giriniz
***'
     'CURSOR VSC SCND 12 23'
     Signal READ_REP
   End

If page = 'PAGE' then
   Do
     'WRITE VSC SCND  3  1 55 (PRO HI FIELD *** Sayfa Sayısını Giriniz ***'
     'CURSOR VSC SCND 15 21'
     Signal READ_REP
   End

If (datatype(page) ^= 'NUM') then
   Do
     'WRITE VSC SCND  3  1 55 (PRO HI FIELD *** Sayfa Sayısı Hatalı! Tekrar Giriniz
***'
```

```
    'CURSOR VSC SCND 15 21'
    Signal READ_REP
  End
Signal PROCESS

THESIS:
        /* DROPPING VARIABLES IN CASE MULTIPLE RECORDING */
If nodrop^='YES' then Do
                      Drop year where inst
                    End
Call WIN1 SCND
'WRITE VSC SCND  1  5' length(tt.0)+1 '(PRO  FIELD' tt.0
'WRITE VSC SCND  1 17 16 (PRO HI FIELD' substr(tt.0,13,15)
'WRITE VSC SCND  1 38 15 (PRO HI FIELD' substr(tt.0,34,14)
'WRITE VSC SCND  5  1 27 (PRO HI FIELD' tt.1
'WRITE VSC SCND  5 28  5 (FIELD 19__'
'WRITE VSC SCND  8  1 27 (PRO HI FIELD' tt.2
'WRITE VSC SCND  8 28  2 (FIELD _'
'WRITE VSC SCND  8 38 29 (PRO FIELD' tt.3
'WRITE VSC SCND  9 38 29 (PRO FIELD' tt.4
'WRITE VSC SCND 12  1 29 (PRO HI FIELD' tt.5
'WRITE VSC SCND 12 28  2 (FIELD _'
'WRITE VSC SCND 12 38 29 (PRO FIELD <1> ==> Fen B.Ens.'
'WRITE VSC SCND 13 38 29 (PRO FIELD <2> ==> Sos.B.Ens.'
'WRITE VSC SCND 14 38 29 (PRO FIELD <3> ==> N.En. Ens.'
Call Pfs
                /* READING SCREEN */
'CURSOR VSC SCND 5 31'
READ_THES:
 if year^='YEAR' then
  'WRITE VSC SCND  5 28 5 (FIELD' substr(year,1,4,'_')
 if where^='WHERE' then
  'WRITE VSC SCND  8 28 2 (FIELD' where
 if inst^='INST' then
   'WRITE VSC SCND 12 28 2 (FIELD' inst
'ALARM VSC SCND'
Drop waitread.
'WAITREAD VSC SCND'
If waitread.1='PFKEY 12' then signal MENU
Do i=3 to waitread.0
   Parse var waitread.i 'DATA' x y z
   z = strip(strip(z,,'_'))
   Select
     When (x= 5) then year=z
     When (x= 8) then where=z
     When (x=12) then inst=z
   End
End
If waitread.1='PFKEY 10' then call prev
                /* ERROR CONTROL */
If (length(strip(year)) ^= 4) ! (datatype(year) ^= 'NUM') then
   Do
     'WRITE VSC SCND  3  1 55 (PRO HI FIELD *** Tezin Kabul Yılı Hatalı! Tekrar
Giriniz ***'
    'CURSOR VSC SCND  5 31'
    Signal READ_THES
   End
```

```
If where= 'WHERE' then
   Do
     'WRITE VSC SCND  3  1 55 (PRO HI FIELD *** Öğrencinin Okul Kodunu Giriniz
***'
     'CURSOR VSC SCND  8 29'
     Signal READ_THES
   End

If (where ^= 1) & (where ^= 2) then
   Do
     'WRITE VSC SCND  3  1 55 (PRO HI FIELD *** Okul Kodu Hatalı! Tekrar Giriniz
***'
     'CURSOR VSC SCND 8 29'
     Signal READ_THES
   End

If (inst = 'INST') & (type ^=8 ) then
   Do
     'WRITE VSC SCND  3  1 55 (PRO HI FIELD Ilgili *** Enstitünün Kodunu Giriniz
***'
     'CURSOR VSC SCND 12 29'
     Signal READ_THES
   End

If (inst ^= 1) & (inst ^= 2) & (inst ^= 3) & (type ^= 8) then
   Do
     'WRITE VSC SCND  3  1 55 (PRO HI FIELD Enstitü *** Kodu Hatalı! Tekrar Giriniz
***'
     'CURSOR VSC SCND 12 29' ; Signal READ_THES
   End
Signal PROCESS

Prev:
'DROP WIN SCND'
 Nodrop='YES'
 Signal READ
Return
 PROCESS:
                  /* SENDING DATA TO DBASE_VM VIA IUCV */

'IUCV SEND' dbase_vm 'RECORD'
'IUCV SEND' dbase_vm prtitle
'IUCV SEND' dbase_vm maxi
Do i=1 to maxi
  'IUCV SEND' dbase_vm title.i
  'IUCV SEND' dbase_vm name.i
  'IUCV SEND' dbase_vm surname.i
  'IUCV SEND' dbase_vm regno.i
End
'IUCV SEND' dbase_vm type
'IUCV SEND' dbase_vm lang
'IUCV SEND' dbase_vm maxs
Do i=1 to 5
   If scode.i^='SCODE.'!!I then 'IUCV SEND' dbase_vm scode.i
End
```

```
Select
  When type=1 then
      Do
        'IUCV SEND' dbase_vm per
        'IUCV SEND' dbase_vm year
        'IUCV SEND' dbase_vm pervol
        'IUCV SEND' dbase_vm perno
        'IUCV SEND' dbase_vm perpg1
        'IUCV SEND' dbase_vm perpg2
           If percode^='PERCODE' then
        'IUCV SEND' dbase_vm percode
           If oryear^='ORYEAR' then
              Do
                 'IUCV SEND' dbase_vm orart
                 'IUCV SEND' dbase_vm orper
                 'IUCV SEND' dbase_vm oryear
              End
      End
  When type=2 then
      Do
        'IUCV SEND' dbase_vm symp
        'IUCV SEND' dbase_vm inst
        'IUCV SEND' dbase_vm city
        'IUCV SEND' dbase_vm country
        'IUCV SEND' dbase_vm notice
        'IUCV SEND' dbase_vm year
        'IUCV SEND' dbase_vm notpg1
        'IUCV SEND' dbase_vm notpg2
      End
  When type=3 then
      Do
        'IUCV SEND' dbase_vm inst
        'IUCV SEND' dbase_vm year
        'IUCV SEND' dbase_vm print
        'IUCV SEND' dbase_vm page
      End


  When type=4 then
      Do
        'IUCV SEND' dbase_vm inst
        'IUCV SEND' dbase_vm year
        'IUCV SEND' dbase_vm print
        'IUCV SEND' dbase_vm page
           If oryear^='ORYEAR' then
              Do
                 'IUCV SEND' dbase_vm orbook
                 'IUCV SEND' dbase_vm oryear
              End
      End
  When type=5 then
      Do
        'IUCV SEND' dbase_vm inst
        'IUCV SEND' dbase_vm way
        'IUCV SEND' dbase_vm year
        'IUCV SEND' dbase_vm page
      End
```

```
  When (type=6)|(type=7) then
      Do
        'IUCV SEND' dbase_vm year
        'IUCV SEND' dbase_vm where
        'IUCV SEND' dbase_vm inst
        'IUCV SEND' dbase_vm type
      End
  Otherwise do ; say 'Hata var, tekrar deneyin!' ; signal EX ; end
End
                    /* RETURN TO THE MAIN MENU */
Signal MENU

          /* SEARCHING THE DATABASE FOR THE USER REQUEST */
SEARCH:
Call WIN1 THRD
'WRITE VSC THRD  1 21' length(ts.0)+1 '(PRO HI FIELD' ts.0
'WRITE VSC THRD  3 15' length(ts.1)+1 '(PRO FIELD' ts.1
'WRITE VSC THRD  4 15' length(ts.2)+1 '(PRO FIELD' ts.2
'WRITE VSC THRD  5 15' length(ts.3)+1 '(PRO FIELD' ts.3
'WRITE VSC THRD  6 15' length(ts.4)+1 '(PRO FIELD' ts.4
'WRITE VSC THRD  7 15' length(ts.5)+1 '(PRO FIELD' ts.5
'WRITE VSC THRD  8 15' length(ts.6)+1 '(PRO FIELD' ts.6
'WRITE VSC THRD  9 15' length(ts.7)+1 '(PRO FIELD' ts.7
'WRITE VSC THRD 14 15 12 (PRO HI FIELD Seçiminiz :'
'WRITE VSC THRD 24 10  6 (PRO HI FIELD' CR
'WRITE VSC THRD 24 17 26 (PRO FIELD' query
'WRITE VSC THRD 24 50  5 (PRO HI FIELD' PF
'WRITE VSC THRD 24 55 17 (PRO FIELD' menup
RE_SRCH:
'WRITE VSC THRD 14 27 2 (FIELD _'
'CURSOR VSC THRD 14 28'
Drop waitread.
'WAITREAD VSC THRD'
If waitread.1='PFKEY 12' then signal MENU
Parse var waitread.3 . . . key



Select
  When key=1 then call SRCH_SUR
  When key=2 then call SRCH_PRT
  When key=3 then call SRCH_YEAR
  When key=4 then call SRCH_YRTY
  When key=5 then call SRCH_LANG
  When key=6 then call SRCH_LNTY
  When key=7 then call SRCH_SCOD
  Otherwise
    Do
      txt='LÜTFEN GÖSTERİLEN SEÇENEKLERDEN BİRİNİ GİRİNİZ'
      'WRITE VSC THRD 21 15' length(txt)+1 '(HI PRO FIELD' txt
      'ALARM VSC THRD'
      Signal RE_SRCH
    End
End
Signal WAIT_SRCH

SRCH_SUR:
```

```
'WRITE VSC THRD 21 15 65 (PRO FIELD  '
'WRITE VSC THRD 18 15 17 (PRO HI FIELD Yazarın Soyadı :'
'WRITE VSC THRD 18 32 18 (FIELD' copies('_',17)
RE_SUR:
'WRITE VSC THRD 14 27  2 (PRO FIELD' key
'CURSOR VSC THRD 18 33'
Drop waitread.
'WAITREAD VSC THRD'
If waitread.1='PFKEY 12' then signal MENU
Parse var waitread.3 'DATA' . . z
surnamq = tran(strip(strip(z,,'_')))
upper surnamq
if surnamq='' then
   Do
        txt='LÜTFEN YAYINLARINI TARAMAK İSTEDİĞİNİZ KİŞİNİN SOYADINI
GİRİNİZ'
      'WRITE VSC THRD 22 15' length(txt)+1 '(HI PRO FIELD' txt
      'ALARM VSC THRD'
      Signal RE_SUR
   End
'IUCV SEND' dbase_vm 'SEARCH'
'IUCV SEND' dbase_vm 'SURNAME'
'IUCV SEND' dbase_vm surnamq
Return

SRCH_PRT:
'WRITE VSC THRD 21 15 65 (PRO FIELD  '
'WRITE VSC THRD 18 15 65 (PRO HI FIELD Yayın Adı -veya içinde geçen bir sözcük- :'
'WRITE VSC THRD 20  1 80 (FIELD' copies('_',79)
RE_PRT:
'WRITE VSC THRD 14 27  2 (PRO FIELD' key
'CURSOR VSC THRD 20 2'
Drop waitread.
'WAITREAD VSC THRD'
If waitread.1='PFKEY 12' then signal MENU
Parse var waitread.3 'DATA' . . z
prtq = strip(strip(z,,'_'))
upper prtq

if prtq='' then
   Do
      txt='LÜTFEN ARAŞTIRMAK İSTEDİĞİNİZ YAYIN ADINI GİRİNİZ'
      'WRITE VSC THRD 22 15' length(txt)+1 '(HI PRO FIELD' txt
      'ALARM VSC THRD'
      Signal RE_PRT
   End
'IUCV SEND' dbase_vm 'SEARCH'
'IUCV SEND' dbase_vm 'PRTITLE'
'IUCV SEND' dbase_vm prtq
Return

SRCH_YEAR:
'WRITE VSC THRD 21 15 65 (PRO FIELD  '
'WRITE VSC THRD 18 15 13 (PRO HI FIELD Yayın Yılı :'
'WRITE VSC THRD 18 28  5 (FIELD 19__'
Drop yearq
RE_YEAR:
```

```
'WRITE VSC THRD 14 27  2 (PRO FIELD' key
If yearq^='YEARQ' then 'WRITE VSC THRD 18 28 5 (FIELD' left(yearq,4,'_')
'CURSOR VSC THRD 18 31'
Drop waitread.
'WAITREAD VSC THRD'
If waitread.1='PFKEY 12' then signal MENU
Parse var waitread.3 'DATA 18' y z
If y=28 then yearq = strip(strip(z,,'_'))
If length(yearq) ^= 4 | datatype(yearq) ^= 'NUM' then
    Do
       txt='YAZILAN YAYIN YILI HATALI! TEKRAR GİRİNİZ'
       'WRITE VSC THRD 22 15' length(txt)+1 '(HI PRO FIELD' txt
       'ALARM VSC THRD'
       Signal RE_YEAR
    End
'IUCV SEND' dbase_vm 'SEARCH'
'IUCV SEND' dbase_vm 'YEAR'
'IUCV SEND' dbase_vm yearq
Return

SRCH_YRTY:
'WRITE VSC THRD 21 15 65 (PRO FIELD  '
'WRITE VSC THRD 17 15 13 (PRO HI FIELD Yayın Yılı :'
'WRITE VSC THRD 17 28  5 (FIELD 19__'
'WRITE VSC THRD 17 35 13 (PRO HI FIELD Yayın Tipi :'
'WRITE VSC THRD 17 48  2 (FIELD _'
'WRITE VSC THRD 19 40 41 (PRO FIELD (1)-Makale   (2)-Bildiri   (3)-Ders Notu'
'WRITE VSC THRD 20 40 41 (PRO FIELD (4)-Kitap    (5)-Rapor     (6)-Tez'
'CURSOR VSC THRD 17 31'
drop yearq typeq
RE_YRTY:
'WRITE VSC THRD 14 27  2 (PRO FIELD' key
If typeq^='TYPEQ' then 'WRITE VSC THRD 17 48 2 (FIELD' left(typeq,1,'_')
If yearq^='YEARQ' then 'WRITE VSC THRD 17 28 5 (FIELD' left(yearq,4,'_')
Drop waitread.
'WAITREAD VSC THRD'
If waitread.1='PFKEY 12' then signal MENU
Do i=3 to waitread.0
   Parse var waitread.i 'DATA 17' y z
   z = strip(strip(z,,'_'))
   Select
     When (y=28) then yearq=z
     When (y=48) then typeq=z
     Otherwise nop
   End
End
If length(yearq) ^= 4 | datatype(yearq) ^= 'NUM' then
    Do
       txt='YAZILAN YAYIN YILI HATALI! TEKRAR GİRİNİZ'
       'WRITE VSC THRD 22 15' length(txt)+1 '(HI PRO FIELD' txt
       'CURSOR VSC THRD 17 29'
       'ALARM VSC THRD'
       Signal RE_YRTY
    End
If typeq='' | verify(typeq,'123456')^=0 then
   Do
     'WRITE VSC THRD 22 15 55 (PRO HI FIELD Yayın Tipi Hatalı! Tekrar Giriniz'
```

```
    'CURSOR VSC THRD 17 49'
    'ALARM VSC THRD'
    Signal RE_YRTY
  End
'IUCV SEND' dbase_vm 'SEARCH'
'IUCV SEND' dbase_vm 'YRTY'
'IUCV SEND' dbase_vm yearq
'IUCV SEND' dbase_vm typeq
Return

SRCH_LANG:
'WRITE VSC THRD 21 15 65 (PRO FIELD  '
'WRITE VSC THRD 19 21'  length(T.14)+1 '(PRO FIELD' T.14
'WRITE VSC THRD 20 21'  length(T.15)+1 '(PRO FIELD' T.15
'WRITE VSC THRD 21 21'  length(T.16)+1 '(PRO FIELD' T.16
'WRITE VSC THRD 17 15 13 (PRO HI FIELD Yayın Dili :'
'WRITE VSC THRD 17 28  2 (FIELD _'
Drop langq
RE_LANG:
'WRITE VSC THRD 14 27  2 (PRO FIELD' key
'CURSOR VSC THRD 17 29'
If langq^='LANGQ' then 'WRITE VSC THRD 17 28 2 (FIELD' left(langq,1,'_')
Drop waitread.
'WAITREAD VSC THRD'
If waitread.1='PFKEY 12' then signal MENU
Parse var waitread.3 'DATA 17' y z
If y=28 then langq = strip(strip(z,,'_'))
If langq='' | verify(langq,'012345678')^=0 then
   Do
      txt='YAZILAN YAYIN DILI HATALI! TEKRAR GİRİNİZ'
      'WRITE VSC THRD 22 15' length(txt)+1 '(HI PRO FIELD' txt
      'ALARM VSC THRD'
      Signal RE_LANG
   End
'IUCV SEND' dbase_vm 'SEARCH'
'IUCV SEND' dbase_vm 'LANG'
'IUCV SEND' dbase_vm langq
Return

SRCH_LNTY:
'WRITE VSC THRD 21 15 65 (PRO FIELD  '
'WRITE VSC THRD 16 21'  length(T.14)+1 '(PRO FIELD' T.14
'WRITE VSC THRD 17 21'  length(T.15)+1 '(PRO FIELD' T.15
'WRITE VSC THRD 18 21'  length(T.16)+1 '(PRO FIELD' T.16
'WRITE VSC THRD 16  5 13 (PRO HI FIELD Yayın Dili :'
'WRITE VSC THRD 16 18  2 (FIELD _'
'WRITE VSC THRD 19  5 13 (PRO HI FIELD Yayın Tipi :'
'WRITE VSC THRD 19 18  2 (FIELD _'
'WRITE VSC THRD 19 23 41 (PRO FIELD (1)-Makale   (2)-Bildiri   (3)-Ders Notu'
'WRITE VSC THRD 20 23 41 (PRO FIELD (4)-Kitap    (5)-Rapor     (6)-Tez'
'CURSOR VSC THRD 16 19'
Drop langq typeq
RE_LNTY:
'WRITE VSC THRD 14 27  2 (PRO FIELD' key
If langq^='LANGQ' then 'WRITE VSC THRD 16 18 2 (FIELD' left(langq,1,'_')
If typeq^='TYPEQ' then 'WRITE VSC THRD 19 18 2 (FIELD' left(typeq,1,'_')
Drop waitread.
```

```
'WAITREAD VSC THRD'
If waitread.1='PFKEY 12' then signal MENU
Do i=3 to waitread.0
   Parse var waitread.i 'DATA' x '18' z
   z = strip(strip(z,,'_'))
   Select
     When (x=16) then langq=z
     When (x=19) then typeq=z
     Otherwise nop
   End
End
If langq='' | verify(langq,'012345678')^=0 then
    Do
       txt='YAZILAN YAYIN DILI HATALI! TEKRAR GİRİNİZ'
       'WRITE VSC THRD 22 15' length(txt)+1 '(HI PRO FIELD' txt
       'CURSOR VSC THRD 16 19'
       'ALARM VSC THRD'
       Signal RE_LNTY
    End
If typeq='' | verify(typeq,'123456')^=0 then
   Do
     'WRITE VSC THRD 22 15 55 (PRO HI FIELD Yayın Tipi Hatalı! Tekrar Giriniz'
     'CURSOR VSC THRD 19 19'
     'ALARM VSC THRD'
     Signal RE_LNTY
   End
'IUCV SEND' dbase_vm 'SEARCH'
'IUCV SEND' dbase_vm 'LNTY'
'IUCV SEND' dbase_vm langq
'IUCV SEND' dbase_vm typeq
Return

SRCH_SCOD:
'WRITE VSC THRD 21 15 65 (PRO FIELD  '
'WRITE VSC THRD 18 15 22 (PRO HI FIELD Doç.Bilim Dalı Kodu :'
'WRITE VSC THRD 18 37  8 (FIELD _______'
Drop scodq
RE_SCOD:
'WRITE VSC THRD 14 27  2 (PRO FIELD' key
If scodq^='SCODQ' then 'WRITE VSC THRD 18 37 8 (FIELD' left(scodq,7,'_')
'CURSOR VSC THRD 18 38'
Drop waitread.
'WAITREAD VSC THRD'
If waitread.1='PFKEY 12' then signal MENU
Parse var waitread.3 'DATA 18' y z
If y=37 then scodq = strip(strip(z,,'_'))
If datatype(scodq) ^= 'NUM' then
    Do
       txt='YAZILAN BILIM KODU HATALI! TEKRAR GİRİNİZ'
       'WRITE VSC THRD 22 15' length(txt)+1 '(HI PRO FIELD' txt
       'ALARM VSC THRD'
       Signal RE_SCOD
    End
'IUCV SEND' dbase_vm 'SEARCH'
'IUCV SEND' dbase_vm 'SCOD'
'IUCV SEND' dbase_vm scodq
Return
```

```
WAIT_SRCH:
'IUCV WAIT'
'CP SLEEP 1 SEC'
'IUCV GRAB'
x=1
first.x=1
st_sz=queued()
Do y=1 to st_sz
   Parse pull gln.y
   Inp.y=substr(gln.y,9)
   If ( inp.y='15'X ) & ( y^=st_sz ) then
            Do
               x=x+1 ; first.x=y+1
            End
End
maxl=x  ;  w=0
/*
nam.='' ; name.='' ; surname.='' ; type.='' ; lang.=''
*/
Do x=1 to maxl
   d=first.x
   Select
     When inp.d=1 then
       Do
          w=w+1
          Call ART_PARSE w
       End
     When inp.d=2 then
       Do
          w=w+1
          Call NTC_PARSE w
       End
     When inp.d=3 then
       Do
          w=w+1
          Call NOT_PARSE w
       End

     When inp.d=4 then
       Do
          w=w+1
          Call BOO_PARSE w
       End
     When inp.d=5 then
       Do
          w=w+1
          Call REP_PARSE w
       End
     When inp.d=6 then
       Do
          w=w+1
          Call THS_PARSE w
       End
     Otherwise nop
   End
End
```

```
Call WIN1 FRTH
'WRITE VSC FRTH 1 10 20 (RES PRO HI FIELD YAZARIN SOYADI, ADI'
'WRITE VSC FRTH 2  3 33 (RES PRO FIELD' copies('-',32)
'WRITE VSC FRTH 1 38 20 (RES PRO HI FIELD' center('YAYININ TIPI',19)
'WRITE VSC FRTH 1 60 11 (RES PRO HI FIELD' center('YAYIN DILI',10)
'WRITE VSC FRTH 1 73  8 (RES PRO HI FIELD FORM NO'
'WRITE VSC FRTH 2 38 20 (RES PRO FIELD' copies('-',19)
'WRITE VSC FRTH 2 60 11 (RES PRO FIELD' copies('-',10)
'WRITE VSC FRTH 2 73  8 (RES PRO FIELD -------'
'WRITE VSC FRTH -2  1 80 (RES PRO FIELD YAYIN HAKKINDA AYRINTILI BİLGİ
ALMAK IÇIN ILGILI SÜTUNA X YAZIP ENTER"A BASINIZ'
Call Pfcont FRTH
If w=0 then
'WRITE VSC FRTH 12 10 50 (PRO HI FIELD Arama kriterine uyan hiç bir yayın
bulunamadı!'
Do b=1 to w
If key=1 then
   Do c=1 to 5
      If index(surname.b.c,surnamq)^=0 then leave c
   End ; else c=1
   Nam.b = surname.b.c', 'name.b.c
  'WRITE VSC FRTH' b '1 2 (FIELD _'
  'WRITE VSC FRTH' b '3 33 (PRO FIELD' substr(nam.b,1,32)
  'WRITE VSC FRTH' b '38 20 (PRO FIELD' typ(type.b)
  'WRITE VSC FRTH' b '60 11 (PRO FIELD' lng(lang.b)
  'WRITE VSC FRTH' b '73  8 (PRO FIELD' center(form_no.b,7)
End
pp=1
'CURSOR VSC FRTH 1 2'
Again:
'POP WIN FRTH'
Drop waitread.
'WAITREAD VSC FRTH'
Select
  When waitread.1='PFKEY 12' then signal MENU
  When waitread.1='PFKEY  9' then Do
                                    'DROP WIN FRTH'
                                     signal SEARCH
                                  End

  When waitread.1='PFKEY  7' & pp>1 then do
                                          pp=pp-1
                                         'SCROLL BACK FRTH'
                                       end
  When waitread.1='PFKEY  8' & pp<1+(w-1)%20 then do
                                                    pp=pp+1
                                                   'SCROLL FORW FRTH'
                                                 end
  When waitread.1='ENTER' then do d=3 to waitread.0
                               Drop dm e f
                               Parse var waitread.d 'DATA' e f dm
                               Upper dm
                               If f=1 & dm='X' then
                                  do
                                     call SHOW e
                                    'WRITE VSC FRTH' e '1 2 (FIELD >'
                                  end
```

```
                        end
  Otherwise nop
End
signal Again


ART_PARSE:
arg w
type.w=1
call duty
ndx=ndx+1
Call pars(inp.ndx)
per.w     = pd.1
year.w    = pd.2
pervol.w  = pd.3
perno.w   = pd.4
perpg1.w  = pd.5
perpg2.w  = pd.6
if pd.0 = 7 then percode.w = left(pd.7,11)
ndx=ndx+1
If inp.ndx^ = '15'X then
   Do
      Call pars(inp.ndx)
      orart.w  = pd.1
      orper.w  = pd.2
      oryear.w = left(pd.3,4)
   End
Return

NTC_PARSE:
arg w
type.w=2
call duty
ndx=ndx+1
Call pars(inp.ndx)
symp.w     = pd.1
inst.w     = pd.2
city.w     = pd.3
country.w  = pd.4
notice.w   = pd.5
year.w     = pd.6
notpg1.w   = pd.7
notpg2.w   = pd.8
Return

NOT_PARSE:
arg w
type.w=3
call duty
ndx=ndx+1
Call pars(inp.ndx)
inst.w  = pd.1
year.w  = pd.2
print.w = pd.3
page.w  = pd.4
Return
```

```
BOO_PARSE:
arg w
type.w=4
call duty
ndx=ndx+1
Call pars(inp.ndx)
inst.w  = pd.1
year.w  = pd.2
print.w = pd.3
page.w  = pd.4
if pd.0 ^= 4 then
   Do
      orbook.w = pd.5
      oryear.w = pd.6
   End
Return

REP_PARSE:
arg w
type.w=5
call duty
ndx=ndx+1
Call pars(inp.ndx)
inst.w = pd.1
way.w  = pd.2
year.w = pd.3
page.w = pd.4
Return

THS_PARSE:
arg w
call duty
ndx=ndx+1
Call pars(inp.ndx)
year.w  = pd.1
where.w = pd.2
inst.w  = pd.3
type.w  = pd.4
Return


DUTY:
ndx=1+first.x
call pars(inp.ndx)
form_no.w=pd.1
prtitle.w=pd.2
lang.w=pd.3
Do m=4 to pd.0
   r = m-3
   scode.w.r= pd.m
End
maxs.w=r
ndx=ndx+1
Call pars(inp.ndx)
q=0
Do i=1 to pd.0 by 4
   q=q+1 ; j=i+1  ;  k=i+2  ;  l=i+3
```

```
   title.w.q   = pd.i
   name.w.q    = pd.j
   surname.w.q = pd.k
   regno.w.q   = pd.l
End
maxi.w=q
Return

Pars: procedure expose pd.
arg str
str=strip(str,,'15'X)
drop pd.
S=0
Do while str^=''
   S = s+1
   Parse var str pd.s '15'x str
End
pd.0=s
Return

SHOW:
                /* SHOWING THE INCOMING DATA ON THE SCREEN */
arg e
Call WIN1 FIVE
'DROP WIN FRTH'
lnt = length(prtitle.e) ; lt = 80*(1+(lnt-1)%80)
if lnt>80 then lnt=80
'WRITE VSC FIVE  1 36 10 (PRO HI FIELD Yayın Adı'
'WRITE VSC FIVE  2  1 81 (PRO    FIELD' center(copies('-',lnt),80)
'WRITE VSC FIVE  3  1 161 (PRO   FIELD' center(prtitle.e,lt)
'WRITE VSC FIVE  8  1 45 (PRO HI FIELD' center('Yazar(lar)',44)
'WRITE VSC FIVE  9  1 45 (PRO FIELD' copies('-',44)
'WRITE VSC FIVE  8 48 26 (PRO HI FIELD Kurumu (ITÜ ise Sicil No)'
'WRITE VSC FIVE  9 48 26 (PRO FIELD' copies('-',25)
'WRITE VSC FIVE 18  3'  length(T.17)+1 '(PRO HI FIELD' T.17
'WRITE VSC FIVE 24  1 6 (PRO HI FIELD' cr
'WRITE VSC FIVE 24  7 43 (PRO FIELD : Yayın hakkındaki diğer bilgileri göster'
'WRITE VSC FIVE 24 55 5 (PRO HI FIELD PF10'
'WRITE VSC FIVE 24 60 20 (PRO FIELD : Önceki ekrana dön'

Do y=1 to maxi.e
   merg=left(tit(title.e.y),9)' 'surname.e.y', 'name.e.y
  'WRITE VSC FIVE' y+9 ' 1 45 (PRO FIELD' merg
  'WRITE VSC FIVE' y+9 '48 26 (PRO FIELD' regno.e.y
End
  Do s=1 to maxs.e
    'WRITE VSC FIVE 20' s*10-6 '8 (FIELD' substr(scode.e.s,1,7)
  End
RPT:
'WAITREAD VSC FIVE'
Select
  When waitread.1='ENTER' then call show_rest e
  When waitread.1='PFKEY 10' then return
  Otherwise signal Rpt
End
If result='RE_SHOW' then call show e
Return
```

```
SHOW_REST:
arg e
Call win1 LAST
'WRITE VSC LAST -1  1  4 (RES PRO HI FIELD PF9'
'WRITE VSC LAST -1  5 28 (RES PRO FIELD : Yayın Tarama menüsüne dön'
'WRITE VSC LAST -1 35  5 (RES PRO HI FIELD PF10'
'WRITE VSC LAST -1 40 20 (RES PRO FIELD : Önceki ekrana dön'
'WRITE VSC LAST -1 61  6 (RES PRO HI FIELD' cr
'WRITE VSC LAST -1 67 14 (RES PRO FIELD : Listeye dön'
Select
  When type.e=1 then
    Do
      'WRITE VSC LAST  4  1 33 (PRO HI FIELD' ta.1
      'WRITE VSC LAST  4 34 47 (PRO FIELD' substr(per.e,1,46)
      'WRITE VSC LAST  5 34 47 (PRO FIELD' substr(per.e,47,46)
      'WRITE VSC LAST  7  1 43 (PRO HI FIELD' ta.2
      'WRITE VSC LAST 8 34 5 (PRO FIELD' year.e
      'WRITE VSC LAST 8 41 4 (PRO FIELD' substr(pervol.e,1,3)
      'WRITE VSC LAST 8 47 5 (PRO FIELD' substr(perno.e,1,4)
      'WRITE VSC LAST 8 55 5 (PRO FIELD' substr(perpg1.e,1,4)
      'WRITE VSC LAST 8 61 5 (PRO FIELD' substr(perpg2.e,1,4)
       if percode.e^='PERCODE.'!le then
         Do
           'WRITE VSC LAST 10  1' length(ta.3)+1 '(PRO HI FIELD' ta.3
           'WRITE VSC LAST 11  1 33 (PRO HI FIELD' ta.4
           'WRITE VSC LAST 11 34 12 (PRO FIELD' percode.e
         End
       if orart.e^='ORART.'!le then
        Do
          'WRITE VSC LAST 13  1 33 (PRO HI FIELD' ta.5
          'WRITE VSC LAST 18  1' length(ta.6)+1 '(PRO HI FIELD' ta.6
          'WRITE VSC LAST 22  1 33 (PRO HI FIELD' ta.7
          'WRITE VSC LAST 13 34 47 (PRO FIELD' substr(orart.e,1,46)
          'WRITE VSC LAST 14 34 47 (PRO FIELD' substr(orart.e,47,46)
          'WRITE VSC LAST 15 34 47 (PRO FIELD' substr(orart.e,93,46)
          'WRITE VSC LAST 16 34 47 (PRO FIELD' substr(orart.e,139,46)
          'WRITE VSC LAST  19 34 47 (PRO FIELD' substr(orper.e,1,46)
          'WRITE VSC LAST  20 34 47 (PRO FIELD' substr(orper.e,47,46)
          'WRITE VSC LAST 22 34 5 (PRO FIELD' oryear.e
        End
    End
  When type.e=2 then
    Do
      'WRITE VSC LAST  5  1 44 (PRO HI FIELD' tn.1
      'WRITE VSC LAST 10  1 37 (PRO HI FIELD' tn.2
      'WRITE VSC LAST 14  1 44 (PRO HI FIELD' tn.3
      'WRITE VSC LAST 14 57  4 (PRO HI FIELD' ' / '
      'WRITE VSC LAST 17  1 37 (PRO HI FIELD' tn.4
      'WRITE VSC LAST 21  1 37 (PRO HI FIELD' tn.5
      'WRITE VSC LAST 21 45  2 (PRO HI FIELD /'
      'WRITE VSC LAST 6 38 43 (PRO FIELD' substr(symp.e,1,42)
      'WRITE VSC LAST 7 38 43 (PRO FIELD' substr(symp.e,43,42)
      'WRITE VSC LAST 10 38 43 (PRO FIELD' substr(inst.e,1,42)
      'WRITE VSC LAST 11 38 43 (PRO FIELD' substr(inst.e,43,42)
      'WRITE VSC LAST 14 38 19 (PRO FIELD' substr(city.e,1,18)
      'WRITE VSC LAST 14 61 20 (PRO FIELD' substr(country.e,1,19)
```

```
    'WRITE VSC LAST 17 38 43 (PRO FIELD' substr(notice.e,1,42)
    'WRITE VSC LAST 18 38 43 (PRO FIELD' substr(notice.e,43,42)
    'WRITE VSC LAST 21 38 5 (PRO FIELD' substr(year.e,1,4)
    'WRITE VSC LAST 21 49 5 (PRO FIELD' substr(notpg1.e,1,4)
    'WRITE VSC LAST 21 55 5 (PRO FIELD' substr(notpg2.e,1,4)
  End
When type.e=3 then
  Do
    'WRITE VSC LAST  5  1 19 (PRO HI FIELD' to.1
    'WRITE VSC LAST  9  1 19 (PRO HI FIELD' to.2
    'WRITE VSC LAST 12  1 19 (PRO HI FIELD' to.3
    'WRITE VSC LAST 15  1 19 (PRO HI FIELD' to.4
    'WRITE VSC LAST  5 20 41 (PRO FIELD' substr(inst.e,1,40)
    'WRITE VSC LAST  6 20 41 (PRO FIELD' substr(inst.e,41,40)
    'WRITE VSC LAST  9 20 5 (PRO FIELD' substr(year.e,1,4,)
    'WRITE VSC LAST 12 20 3 (PRO FIELD' substr(print.e,1,2)
    'WRITE VSC LAST 15 20 5 (PRO FIELD' substr(page.e,1,4)
  End
When type.e=4 then
  Do
    'WRITE VSC LAST  4  1 33 (PRO HI FIELD' tb.1
    'WRITE VSC LAST  8  1 33 (PRO HI FIELD' tb.2
    'WRITE VSC LAST 11  1 48 (PRO HI FIELD' tb.3
    'WRITE VSC LAST 14  1 48 (PRO HI FIELD' tb.4
    'WRITE VSC LAST  4 34 41 (PRO FIELD' substr(inst.e,1,40)
    'WRITE VSC LAST  5 34 41 (PRO FIELD' substr(inst.e,41,40)
    'WRITE VSC LAST  8 34 5 (PRO FIELD' substr(year.e,1,4)
    'WRITE VSC LAST 11 34 3 (PRO FIELD' substr(print.e,1,2)
    'WRITE VSC LAST 14 34 5 (PRO FIELD' substr(page.e,1,4)
     if orbook.e^='ORBOOK.'lle then
        Do
          'WRITE VSC LAST 17  1 33 (PRO HI FIELD' tb.5
          'WRITE VSC LAST 22  1 33 (PRO HI FIELD' tb.2
          'WRITE VSC LAST 17 34 41 (PRO FIELD' substr(orbook.e,1,40)
          'WRITE VSC LAST 18 34 41 (PRO FIELD' substr(orbook.e,41,40)
          'WRITE VSC LAST 19 34 41 (PRO FIELD' substr(orbook.e,81,40)
          'WRITE VSC LAST 20 34 41 (PRO FIELD' substr(orbook.e,121,40)
          'WRITE VSC LAST 22 34 5 (PRO FIELD' substr(oryear.e,1,4)
        End
  End
When type.e=5 then
  Do
    'WRITE VSC LAST  5  1 19 (PRO HI FIELD' tr.1
    'WRITE VSC LAST  9  1 19 (PRO HI FIELD' tr.2
    'WRITE VSC LAST 12  1 19 (PRO HI FIELD' tr.4
    'WRITE VSC LAST 15  1 19 (PRO HI FIELD' tr.5
    'WRITE VSC LAST  5 20 41 (PRO FIELD' substr(inst.e,1,40)
    'WRITE VSC LAST  6 20 41 (PRO FIELD' substr(inst.e,41,40)
     If way.e=1 then tx='BASKI' ; else tx='TEKSIR'
    'WRITE VSC LAST  9 20 7 (PRO FIELD' tx
    'WRITE VSC LAST 12 20 5 (PRO FIELD' substr(year.e,1,4)
    'WRITE VSC LAST 15 20 5 (PRO FIELD' substr(page.e,1,4)
  End
When type.e=6 ! type.e=7 then
  Do
    'WRITE VSC LAST  5  1 27 (PRO HI FIELD' tt.1
    'WRITE VSC LAST  8  1 27 (PRO HI FIELD' tt.2
```

```
      'WRITE VSC LAST 12  1 29 (PRO HI FIELD' tt.5
       Select
         When inst.e=1 then ins='Fen B.Ens.'
         When inst.e=2 then ins='Sos.B.Ens.'
         Otherwise          ins='N.En. Ens.'
       End
       If where.e=1 then wh='İTÜ"den' ; else wh='Başka Üniversiteden'
      'WRITE VSC LAST  5 28 5  (FIELD' substr(year.e,1,4)
      'WRITE VSC LAST  8 28 22 (FIELD' wh
      'WRITE VSC LAST 12 28 41 (FIELD' ins
    End
  Otherwise nop
End
Rpt_l:
'WAITREAD VSC LAST'
Select
  When waitread.1='ENTER' then return
  When waitread.1='PFKEY 10' then return 'RE_SHOW'
  When waitread.1='PFKEY  9' then Do
                                  'DROP WIN LAST'
                                   signal SEARCH
                                 End
  Otherwise signal Rpt_l
End
Return

Typ: procedure
Arg t
Select
  When t=1 then ty='BİLİMSEL MAKALE'
  When t=2 then ty='YAYINLANMIŞ BİLDİRİ'
  When t=3 then ty='DERS NOTU'
  When t=4 then ty='KİTAP'
  When t=5 then ty='YAYINLANMIŞ RAPOR'
  When t=6 then ty='Y.LİSANS TEZİ'
  When t=7 then ty='DOKTORA TEZİ'
  Otherwise nop
End
Return ty

Lng: procedure
Arg l
Select
  When l=0 then lg='TÜRKÇE'
  When l=1 then lg='ALMANCA'
  When l=2 then lg='ARAPÇA'
  When l=3 then lg='FARSÇA'
  When l=4 then lg='FRANSIZCA'
  When l=5 then lg='İNGİLİZCE'
  When l=6 then lg='İSPANYOLCA'
  When l=7 then lg='İTALYANCA'
  When l=8 then lg='RUSÇA'
  Otherwise nop
End
Return lg

Tit: procedure
```

```
Arg v
Select
  When v=1  then ti='Prof.Dr.'
  When v=2  then ti='Prof.'
  When v=3  then ti='Doç.Dr.'
  When v=4  then ti='Doç.'
  When v=5  then ti='Y.Doç.'
  When v=6  then ti='Y.Doç.Dr.'
  When v=7  then ti='Dr.'
  When v=8  then ti='Araş.Gör.'
  When v=9  then ti='Uzman'
  When v=10 then ti='Ögr.Gör.'
  When v=0  then ti='Ögrenci'
  Otherwise      ti='          '
End
Return ti


TRAN: procedure
Parse arg coming
Return translate(coming,'IĞÜŞIÖÇ','ığüşiöç')

                /* EXIT FROM THE PROGRAM */
EX:
Call DEL_ALL
Exit

       * PROCEDURE WHICH DELETES ALL WINDOWS ANS VSCREENS *
DEL_ALL:
'SET CMSTYPE HT'
'DROPBUF'
'Q WINDOW (STACK'
Do queued()   ; pull . name .
'DELETE WINDOW' name ; end
'Q VSCREEN (STACK'
Do queued()   ; pull . name .
'DELETE VSCREEN' name ; end
'SET CMSTYPE RT'
Return


           /* PROCEDURE WHICH CLEARS VSCREEN  scrn   */
WIN1: arg scrn
 'CLEAR VSC' scrn
 Do i=1 to 24
    'WRITE VSC' scrn i '1 80 (PRO FIELD  '
 End
'SHOW WINDOW' scrn 'ON' scrn '1 1'
Return

           /* PROCEDURES WHICH DESIGNATES PF_Keys */
Genpf:
arg vscr
'WRITE VSC' vscr '24 1  6 (PRO HI FIELD' CR
'WRITE VSC' vscr '24 7 26 (PRO FIELD' next
'WRITE VSC' vscr '24 58 5 (PRO HI FIELD' PF
'WRITE VSC' vscr '24 63 17 (PRO FIELD' menup
```

```
Return

Pfs:
'WRITE VSC SCND 24 1  6 (PRO HI FIELD' Cr
'WRITE VSC SCND 24 7 23 (PRO FIELD' save
'WRITE VSC SCND 24 30 5 (PRO HI FIELD' Pfr
'WRITE VSC SCND 24 35 21 (PRO FIELD' back
'WRITE VSC SCND 24 58 5 (PRO HI FIELD' Pf
'WRITE VSC SCND 24 63 17 (PRO FIELD' menup
Return

Pfcont:
arg vscr
'WRITE VSC' vscr '-1  1  4 (RES PRO HI FIELD' pfb
'WRITE VSC' vscr '-1  5 15 (RES PRO FIELD' back
'WRITE VSC' vscr '-1 22  4 (RES PRO HI FIELD' pff
'WRITE VSC' vscr '-1 26 16 (RES PRO FIELD' next
'WRITE VSC' vscr '-1 44  4 (RES PRO HI FIELD PF9'
'WRITE VSC' vscr '-1 48 16 (RES PRO FIELD' menuq
'WRITE VSC' vscr '-1 65  5 (RES PRO HI FIELD' PF
'WRITE VSC' vscr '-1 70 11 (RES PRO FIELD' menum
Return

ERROR:
Rs=rc
If rs=1 & index(sourceline(sigl),'SCROLL')^=0 then signal again
'SET CMSTYPE RT'
'VMFCLEAR' ; Do 10 ; say '' ; end
 If rs=40 ! (length(rs)=4  &  substr(rs,1,3)=101) then
   Do
     no_op='Veri Tabanını sağlayan' dbase_vm 'makinası çalışmıyor'
         E='YAYIN  VERI  TABANINI  CALISTIRAN'  DBASE_VM  'MAKINASINI
AUTOLOG KOMUTUYLA LUTFEN ACINIZ'
     'EXECIO 0 CP (STRING MSG OPERATOR' E
     if rc=0 then info=', Operatöre haber verildi'  ; else info = ''
     say no_op''info
   End ;                                  else
   Say sigl 'nolu satırda hata var! Lütfen Faruk''u arayın'
 Do 10 ; say '' ; end
Exit
```

CURRICULUM VITAE

Faruk KOCABIYIK was born on February 20, 1966 in Istanbul. He graduated from the Kocasinan Lisesi in 1983. He received a BS degree in computer engineering from the Istanbul Technical University in 1988. He has been a research assistant at I.T.U. Data Processing Center since then.