

## BÖLÜM 4

### ANA BELLEK YÖNETİMİ

#### 10.1. Ana Bellek Yönetimi İşlevleri

Verilerin içerikleri genellikle kullanıcılar tarafından tanımlanır ve günlenir. Ayrıca içerik işlevinin her komut işletiminde benzer biçimde yinelendiği düşünülürse, bu işlevin sistem açısından çok seçenekli bir yönü olmadığı ortaya çıkar. ( Şekil-28 ) 'de 'A' ve 'B' işlevlerinin zaman içinde ne aşamada ve nasıl çözümlendiği ise ( Yazılım / Donanım, Yazılım + Donanım, ...) bir bilgisayar sisteminin niteliğini saptayan en önemli seçeneklerdir.

Kaynak Program					
	0	---		Ana	
		---	4000	Bellek	
	Simgesel	---	Yerdeğisir		
LU 4,AA(3)		XX32			işletim
	Çevirici	4834	Yükleyici	XX32	
		1000		4834	
		---		5000	
AA DC . . .		---			
	1000	0003			
		1000	5000	0003	
		0005		1000	
	A		B	0005	I

Şekil-28. Bir Programa Uygulanan Bellek Yönetimi

#### 10.1.1. Adlandırma İşlevi

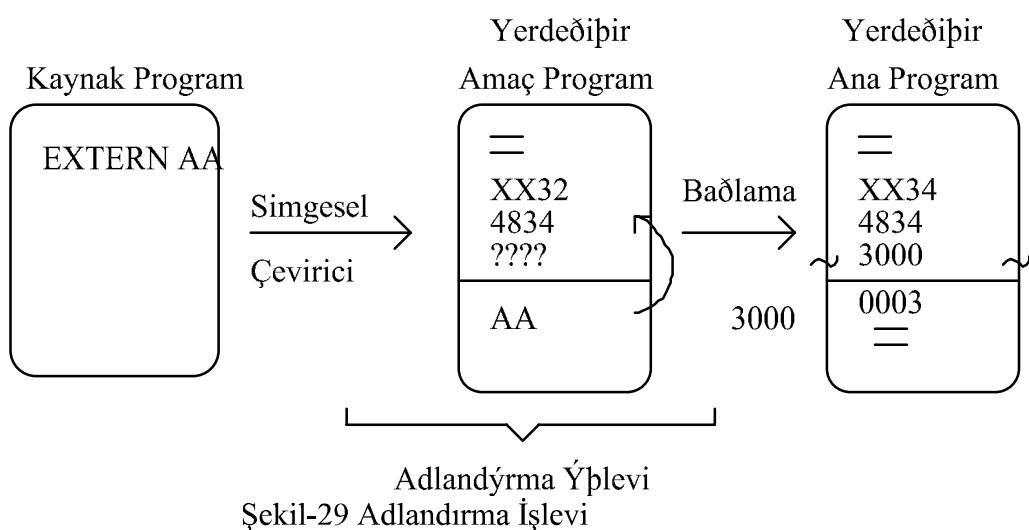
Adlandırma işlevi kullanıcıların sorunlarına esnek, kullanımı kolay ve etkin çözüm sağlama için tanınan ayırt edici simgesel evrenler nedeniyle ortaya olmuş olup; bir program içinde değişik ortamlarda kullanılan benzer simgesel adların ayrı kimliklere dönüştürülmüdür. Blok yapılı dillerde ( PL/1, Pascal gibi ) aynı simgenin ayrı bloklarda kullanımı, bir bilgisayar sisteminde hizmet alan kullanıcıların program, kütük vb. adlarını birbirinden etkilenmeden seçmeleri adlandırma işlevine verebileceğimiz örneklerdir.

Bir program içinde simgesel adların (değişken, kütük adı,...) kimliklere dönüştürülmesi üç yoldan, ayrı ayrı ya da bu yolların bir kesiminin birlikte uygulanması ile gerçekleştirilebilir. Bunlar;

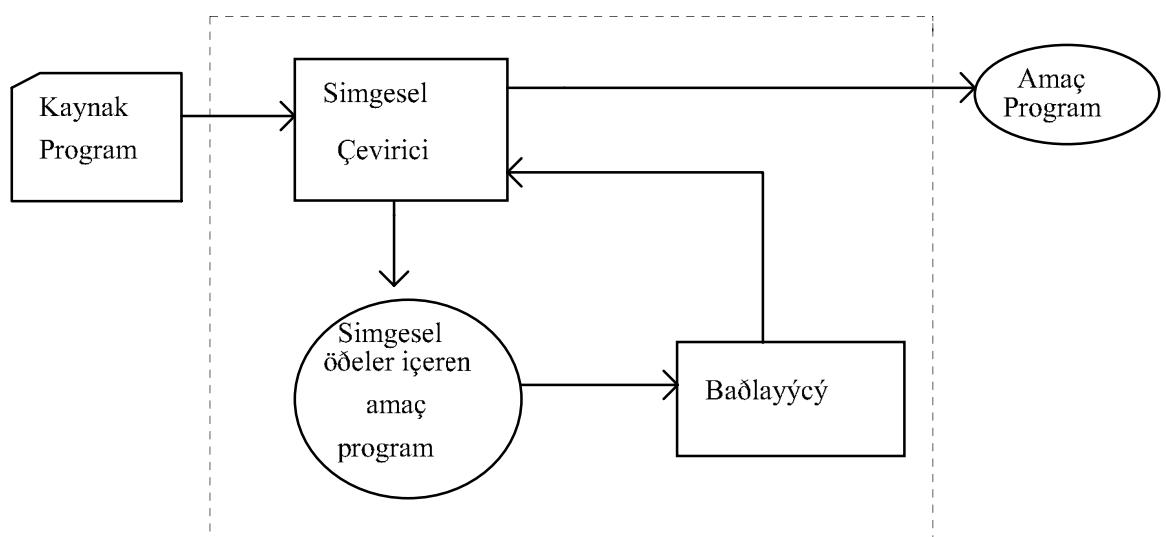
- a) Derleme : Simgesel çeviri süresince,

- b) Yükleme** : Öncesinde ya da sonrasında bağlama süresince
- c) İşletim aşamasında**

( Şekil-28 )'de verilen kodlanmış program örneğinde adlandırılma işlevi için 'a' yolu tümüyle uygulanmaktadır. Aynı örnekte 'AA' simgesinin bir dış ilişki ( External Reference ) simgesi olması başka bir deyişle bu veri alanının başka bir program kesimi içinde yer alması, çeviri süreci sonunda üretilen amaç programda henüz kimliklerine kavuşmamış simgelerin de yer almamasına sebep olacaktır. Adlandırma işleminin tamamlanabilmesi için ayrı bir bağlama sürecine gerek vardır. ( Şekil : 29 )



Amaç programındaki simgeler gerekirse çeviri / bağlama süreçleri bir çok kez yinelerek adreslere dönüştürülür. Bu yaklaşım durgun ( Statik ) bağlama işlevi olarak adlandırılır. ( Şekil-30 )



### **Şekil-30. Durgun Adlandırma İşlevi**

İşletim aşamasında bağlama ise, simgesel öğeleri içeren programları işletime alıp, bu öğelere gereksinim duyulduğu zaman bağlama işlevini çözümlemeyi öngörür. Bu yöntem sağladığı etkinliğe karşın getirdiği yük nedeniyle az benimsenen bir seçenektır.

#### **10.1.2. Bellek İşlevi**

Program içinde görelî adreslerle ana bellekteki gerçek adresler arasında bağlantıyi kuran bellek işlevi, bir bilgisayar sisteminde bellek donanımını belirleyen en büyük etkendir. Bu işlev bir yerdeğîr yükleyici yazılım tarafından, işletim başlamadan bir kez gerçekleştiriliyorsa, ana bellek için 'Durgun Bir Bellek Yönetimi' uygulanmaktadır. Başka bir deyişle ana belleğin belirli bir konumuna yüklenmiş bir program, işletimi sonuçlanıncaya deyin aynı konumu koruyacaktır. Ana belleğin bir kaynak sorunu yaratmadığı sistemlerde bu yönetim, yalnız olması nedeniyle başarı ile uygulanır. Bu yönetimin sakincası önceden planlanmış bir stratejinin donmuş yönlerinden kaynaklanan sorunlardır. Ana belleğin işletim aşamasında ortaya çıkan gereksemelere göre yeniden düzenlenmesi, kaynak sorununu bir ölçüde çözümleyen esnek bir yaklaşımıdır. Bu olgu programlar açısından bellek işlevinin bir çok kez yinelenebilir olmasına bağlıdır. Bu tür bellek yönetimini ön gören sistemleri 'Dinamik Bellek Yönetimi' uygulayan sistemler olarak adlandırıyoruz.

Bellek işlevi, uygulamada ana belleğin düzenlenmesi ve atanmasını içeren iki ayrı işleve dayanmaktadır. Ana belleğin düzenlenmesini konu kapsamında ayrıntılı olarak inceleyeceğiz. Bellek atama işlevi ise 'Veri Yapıları' konusu içinde ayrıntılı olarak incelenir, İşletim Sistemleri kapsamında ana belleği bölüşülen bir kaynak olarak ele alacağız ve atama işlevinin politikası ile ilgileneceğiz. Bellek atamada kullanılan yöntem ne olursa olsun, bu işlevin belli başlı dört görevi içerdigi söylenebilir. Bunlar ;

- a)** Ana belleğim kullanımda / serbest olan kesimlerinin sayışımını yapmak, kullanımda olan kesimlerin ne nitelikte verileri ne süreyle kapsadığını izlemek,
- b)** Kullanıcıların bellek istemlerini karşılamak,
- c)** Kullanımı biten bellek kesimlerini serbest kaynaklar arasına katmak,
- d)** Gerekliyorsa ana belleği yeniden düzenlemektir.

#### **10.2. Ana Belleğin Düzenlenmesi**

Ana belleğin yönetim tekniklerini, bu kaynağı düzenlemekte kullanılan yöntemlere göre söyle sıralayabiliriz.

- |                                      |                            |
|--------------------------------------|----------------------------|
| <b>1)</b> Single Contiguous          | ( Tek ve Kesintisiz )      |
| <b>2)</b> Partitioned                | ( Bölümlere Ayırarak )     |
| <b>3)</b> Relocatable Partitioned    | ( Yer Değişir Bölümlerle ) |
| <b>4)</b> Paged                      | ( Sayfalı )                |
| <b>5)</b> Demand-Paged               | ( İstenebilir-Sayfalı )    |
| <b>6)</b> Segmented                  | ( Kesimleme )              |
| <b>7)</b> Segmented And Demand Paged | ( Kesimleme ve Sayfalama ) |

Bu tekniklerden ilk ikisi 'Durağan', diğerleri 'Dinamik' bellek yönetimi olarak adlandırılır. Ayrıca, ilk dört bellek yönetimi 'Gerçek', diğerleri 'Görüntü' bellek yönetimi olarak sınıflandırılır.

Bu teknikleri incelerken şu 4 olgu üzerinde duracağız;

- 1) Tekniğin bellek yönetimine yaklaşımı, yani kavramlar ve prensipler,
- 2) Tekniğin gerektirdiği özel donanım,
- 3) Teknikle ilgili yazılım algoritmaları ve işlemlerin ( processing ) tanıtılp incelenmesi,
- 4) Teknikle ilgili stratejilerin avantajlarının ve dezavantajlarının tartışılması.

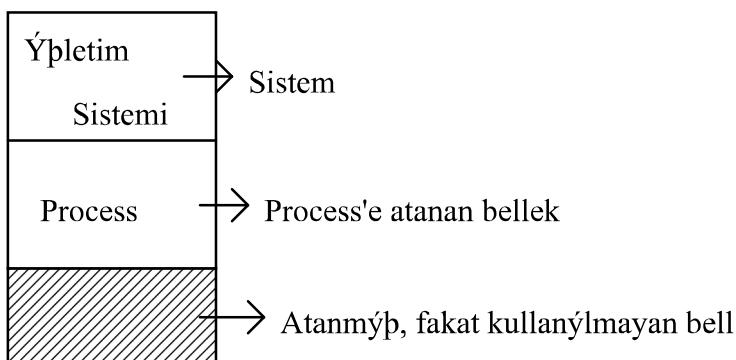
### 10.2.1. Gerçek Ana Bellek Yönetimi

#### 10.2.1.1. Single Contiguous ( Tek ve Kesintisiz ) Bellek Yönetimi

Single Contiguous atama, bellek yönetimi teknikleri içinde en basit olanı ve dolayısıyla hiçbir özel donanım gereksinimi olmayan bir tekniktir. Bu teknik, IBM-1130 Disk Monitor System, Honeywell 316, PDP-8 ve 9 Monitor Sistemler gibi mini bilgisayarlarda uygulanmaktadır.

Bu tip sistemlerde Multi Programming söz konusu olmadığından user, job, job-step ve process olarak tanımlanan kavramlar arasında bire-bir bağlılık bulunmaktadır. Bu nedenle Single-Contiguous bellek yönetiminden söz ederken user, job, job-step ve process arasındaki farkları belirtmeksızın sadece process kelimesini kullanabiliriz.

Bu process'e Single-Contiguous atama ( Şekil-31 )' deki gibi yapılmaktadır.



Şekil-31. Single-Contiguous Atama

Kavram olarak ( Şekil-31 )' de bellek contiguous ( Kesintisiz ) olarak üç bölüme ayrılmıştır. Bunlardan biri kalıcı olarak işletim sistemine atanmıştır. Geri kalan iki ve üçüncü bölümler tümüyle tek bir process'e atanmıştır. Aslında ( Şekil-31 )' de görüldüğü gibi process sadece ikinci bölümünü kullanmakta ve üçüncü bölüm hiçbir şey için kullanılmamaktadır.

Örneğin 256 K byte'lık bir bellek düşünelim ve bunun 32 K'luk bölümünde basit bir işletim sistemi yerleştirilmiş olsun. Geriye kalan 224 K'luk bellek tek bir process'e atanmaktadır. Bu process'in 64 K'luk bir sahada çalışabileceğini varsayırsak, geriye kalan 160 K'luk bellek, bu tüm belleğin %60'ı, hiçbir işe yaramamaktadır.

Single-Contiguous teknigine daha önce sözünü ettigimiz Bellek Yönetim Fonksiyonları açısından bakacak olursak, bu teknığın basitliğine karşın sağladığı avantajları görebiliriz.

- 1) Kaynağın, yani belleğin izlenmesi büyük bir sorun değildir, çünkü bellek tümüyle tek bir process'e atanmaktadır.
- 2) Bellek atama politikası son derece basittir, bir process'in işlenmesine karar verildiğinde ( Scheduled ) bellek tümüyle o process'e verilmektedir.
- 3) Atama işlemi son derece basittir.
- 4) Process'in bellekte olan işi bittiğinde tüm bellek başka bir process'e atanabilir.

### ***Donanım Desteği***

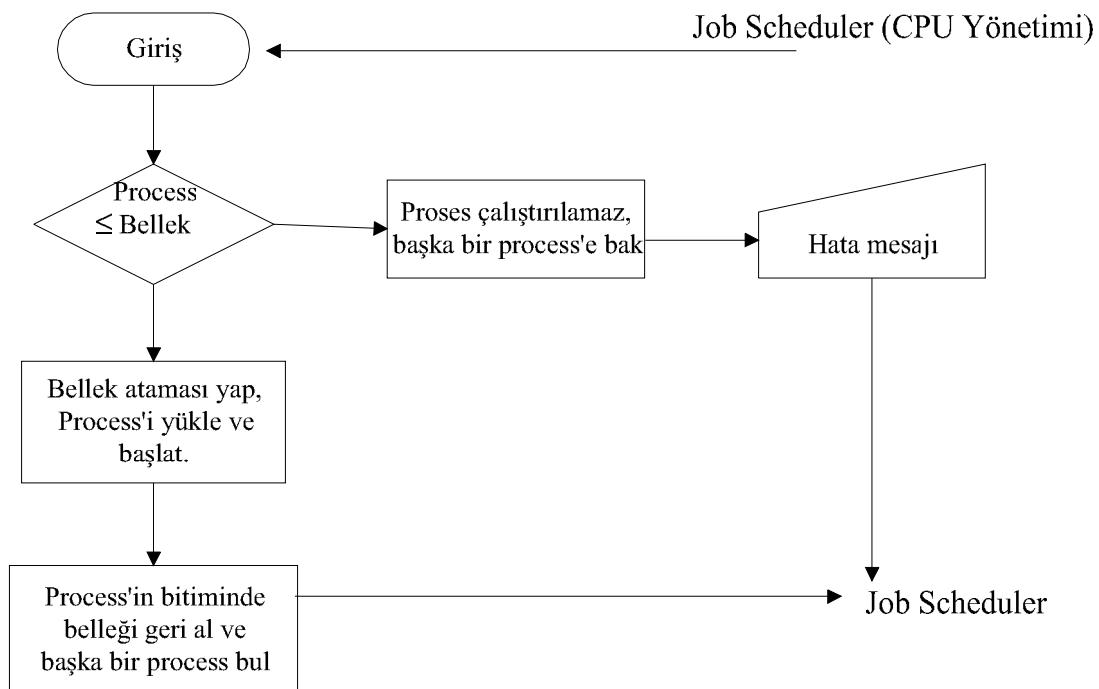
Genel olarak 'Single-Contiguous' atama teknığının özel donanıma gereksinimi yoktur. Ancak bazı sistemlerde process'lerin işletim sisteminin çalıştığı sahaya müdahale etmelerini önlemek amacıyla basit bir koruma ( Protection ) mekanizması bulunmaktadır.

Bu mekanizma sınırlama registerleri ve CPU'nun supervisor modunda çalışmalarını içermektedir. Sınırlama registerleri adında da anlaşılacağı gibi işletim sisteminin bulunduğu sahanın bitiş adresi ile yükldür. Bu register kanalıyla process'lerin bu sahaya erişmeleri önlenmektedir.

CPU genel olarak process ve supervisor olmak üzere iki modda çalışır. Process modunda çalışırken donanım process tarafından adreslenen her sahanın korunmaya alınmış sahanın içinde olup olmadığını kontrol eder. Eğer adresleme sahasının dışına çıkılacak olursa bir interrupt verilir ve kontrol işletim sistemine geçer. Supervisor modunda çalışırken ise işletim sistemi belleğin her yerini adresleyebilir ve sınırlama registerlerini set edebilir. Bu moddan diğer bir moda geçiş yalnızca işletim sistemi kanalıyla ve genellikle interrupt'larla yapılmaktadır.

### ***Yazılım Desteği***

Yazılım yönünden Single-Contiguous atama teknığının tek bir algoritma ile gerçekleştirilebileceğini söyleyebiliriz. Bu algoritmanın ana hatlarını akış şemasıyla (Şekil-32)'deki gibi gösterebiliriz.



Şekil-32. Single-Contiguous Yazılım Algoritması

( Şekil-32 )’de görüldüğü gibi algoritmaya giriş, CPU yönetimindeki Job Scheduler mekanizması tarafından işlenmesi istenilen bir iş olduğunda yer almaktadır. Bellek atanması yapıldıktan sonra, atama yapılan process'in işi bitinceye kadar algoritmaya başka bir çağrıda bulunulamaz.

### *Avantajlar*

Single-Contiguous atama tekniğinin en büyük avantajı şüphesiz çok basit olmasıdır. Böyle bir teknik normal olarak 1 K'luk bir bellekte gerçekleştirilebilir, daha karmaşık bir sistemde bellek yönetimi 100-150 K'luk bir bellek kaplayabilir. Diğer bir avantaj ise, sistemin kullanımındaki kolaylık ve anlaşılabilirlik olarak gösterilebilir.

### *Dezavantajları*

Bu teknin en büyük dezavantajı bellekten tümüyle yararlanılmamasıdır. Bu konuda üç yetersizlik söz edilebilir. Bunlar;

- 1) Belleğin bir bölümü tümüyle harcanmaktadır.
- 2) G/C kanallarının kullanıldığı bir sistem söz konusu ise, bazı durumlarda belleğin hiçbir yeri aktif olarak kullanılmamaktadır. Şöyleki; çalıştırılmakta olan process bir G/C işlemi başlatabilir olsa, o G/C işleminin kanal tarafından yürütüldüğü süre içinde CPU beklemek zorunluğunda kalacak ve process'in bu süre içinde bellekte bulunup bulunmaması hiçbir şey ifade etmeyecektir.

CPU'nun G/Ç bekleme zamanının yüzdesinin ne olduğu sorusu akla gelebilir. Doğal olarak bu yüzde programdan programa değişir. Örneğin IBM 360-65 modeli bir makinada tipik bir iş için bu oran %60-70 dolayında olmaktadır. Dolayısıyla bellek kadar CPU'da verimsiz bir şekilde kullanılmaktadır.

**3)** Çoğu programların adresleme sahası içinde bulunan, ancak, ya çok az ya da hiç kullanılmayan bilgiler taşıyan bölümleri bulunmaktadır. Örneğin hata mesajları veren alt-programlar, tanımlanması gereken fakat az veya hiç kullanılmayan sabit değerler ve buffer sahaları gibi.

Single-Contiguous atama tekniğinin dezavantajları,

- 1)** Verimsiz bellek kullanımı,
  - 2)** Verimsiz CPU kullanımı,
  - 3)** Process'lerin eldeki bellekten daha küçük bir sahaya yerleştirilebilmesi zorunluluğu ,
- şeklinde özetlenebilir.

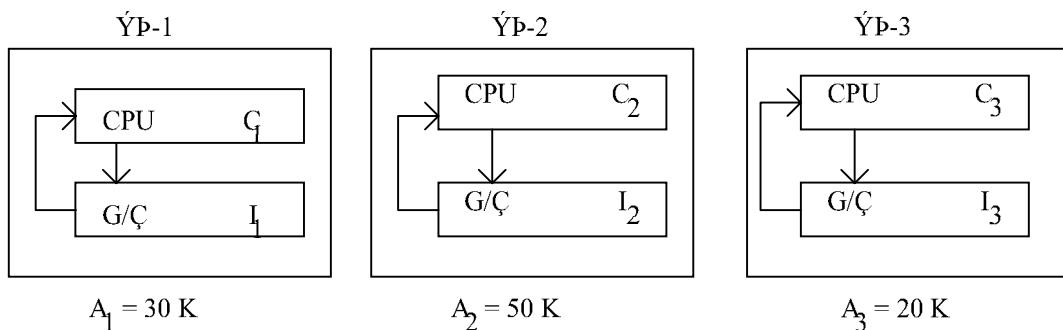
İncelenecək olursa Single-Contiguous atama tekniğinin verimsiliği, bir bilgisayar sisteminin 'Değişmez' olarak nitelendireceğimiz kaynakları ile o sistemi kullananların 'Değişken' olarak nitelendireceğimiz istekleri arasında gereken uyumu sağlayamamaktan doğmaktadır.

Bir bilgisayar sisteminin fiziksel (Donanım) kaynakları genellikle yılda bir veya iki kez değişikliğe uğramaktadır. Bunun yanında kullanıcıların sistemden istekleri sık sık değiştiği gibi istekler arasında da büyük farklılıklar olabilmektedir. Doğal olarak bu uyumsuzluğu gidermek için bütün kullanıcıların aynı uzunlukta program yazmaları ve aynı kaynakları kullanmaları istenemez. Bunu yapmak yerine birden fazla programın aynı zamanda işlenebilmesini sağlama ve kaynakları gereği şekilde paylaşımı olaraq kullanma olanağı sağlanabilir. Bu çalışma şecline Multi-Programming adını vermiştık.

### ***Multi-Programming***

Daha önce sözü edilen Multi-Programming, aslında CPU yönetimi kapsamına giren bir konudur. Ancak bellek yönetimiyle ilgili diğer tekniklerin incelenmesi için Multi-Programming ile ilgili temel kavramların verilmesi gereklidir.

Örnek olarak elimizde birbirinden bağımsız üç ayrı işin bulunduğu düşünelim. Her iş için gereken CPU zamanlarını  $C_1$ ,  $C_2$ ,  $C_3$  ve her iş için gereken G/Ç kanalı kullanım zamanlarını ise  $L_1$ ,  $L_2$ ,  $L_3$  olarak gösterelim. Ayrıca her işin adres sahاسını veya başka bir deyimle bellek gereksinimlerini  $A_1$ ,  $A_2$ ,  $A_3$  ile gösterelim. ( Şekil-33 ).



Şekil-33. Örnek 3 İşin Gösterimi.

Modelimizi basitleştirmek amacıyla har işin önce bir süre CPU kullandığını, ardından bir süre G/Ç yaptığını ve tekrar CPU kullanımına döndüğünü varsayalım. Elimizdeki belleğin kapasitesi 100 K byte olsun. Bu durumda eğer yalnız İş-1 çalıştırılacak olsa 30 K'luk bellek kullanılacak, geriye kalan 70 K'luk bellek boş kalacaktır. Aynı zamanda CPU'nun  $I_1/(C_1+I_1)$  oranındaki zamanı G/Ç işlemlerinin bitmesini beklemekle geçecektir. Burada  $I_1$  ne kadar büyürse CPU bekleme oranında o derece büyüyecektir.

Buna karşın, '(30K+50K+20K)=100K atanabilir bellek kapasitesi' olduğundan, bütün işleri (İş-1, İş-2, İş-3) bellekteki adres sahalarına yerlestirebiliriz. Bu belleğin tümüyle kullanılır hale geçmesini sağlar.

İşler işlemeye başlandığında CPU yönetimi CPU'yu İş-1'e atayacaktır.  $C_1$  kadar bir süre geçtikten sonra CPU yönetimi CPU'yu durdurup  $I_1$  süresini beklemektense, CPU'yu İş-2'ye atayabilecektir. Aynı şekilde İş-2,  $C_2$ 'yi bitirip  $I_2$ 'ye geldiğinde CPU yönetimi CPU'yu  $I_2$  için beklemektense İş-3'e atayabilecektir.

$C_3$  'ün bitiminde iki durum söz konusudur;

- 1)  $I_1 \leq C_2 + C_3$  ise CPU hiç bekletilmeden İş-1'e yeniden atanabilir,
- 2)  $I_1 > C_2 + C_3$  ise CPU  $I_1$  kadar bir süre beklemektense  $I_1 - (C_2 + C_3)$  kadar bir zaman bekletilmiş olacaktır.

Göründüğü gibi Multi-Programming CPU'nun verimliliğini tek bir işin çalıştırılması durumundan çok daha arttırmıştır. Bazı durumlarda iki veya daha fazla işi aşağı yukarı tek bir işin alacağı süre içinde bitirmek mümkündür.

### G/Ç Bekleme Yüzdesinin Ölçülmesi

Buraya kadar kullandığımız "Multi-Programming" modelinde her iş için I'ların ve C'lerin sabit olduğunu ve her I'yı bir C'nin, her C'yi ise bir I'nın izlediğini varsayımiştik.

Aslında gerçek bir process'e ilişkin I ve C değerleri birbirlerine çok daha karmaşık bir şekilde bağlıdır. Dolayısıyla bir işin toplam G/C bekleme yüzdesi, ki biz bunu  $\mu$  ile

gösterebiliriz, daha anlamlı ve daha gerçege yakın bir ölçü ortaya koymaktadır. Bir işin toplam G/C bekleme zamanı oranını,

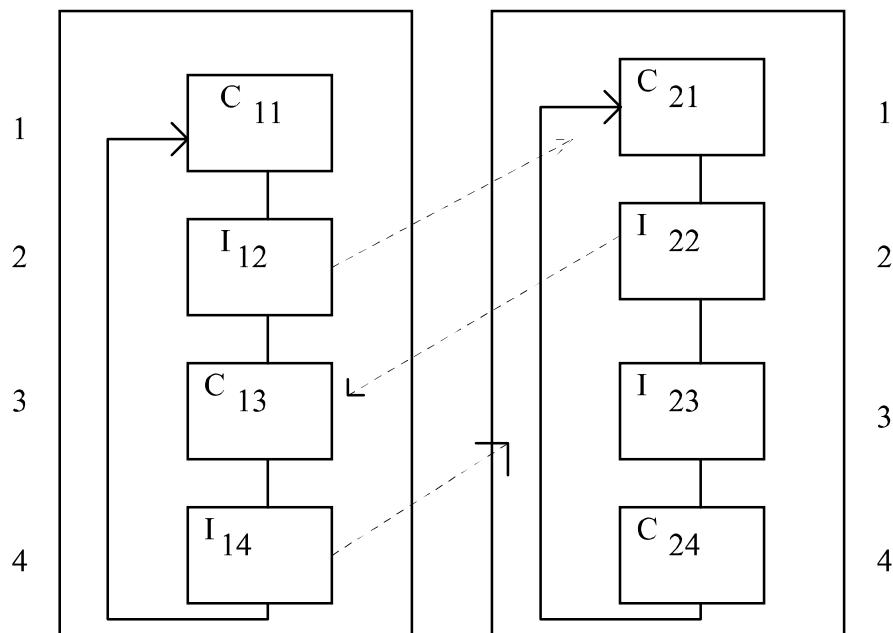
$$\mu = \frac{T_{toplum}.G/C.Bekleme.Zamaný}{T_{toplum}.G/C + CPU.Zamaný}$$

şeklinde gösterebiliriz. Bu oran bir işin tek başına çalıştırılabileceği bir sistemde CPU'nun WAIT statüsüne girdiği zamanları kaydetmek yoluyla kolayca hesaplanabilir. İşletim sistemleri üzerinde yapılan araştırmalarda, orta ve büyük boy (IBM 360/65 veya IBM 370/158 gibi) sistemlerde çalıştırılan tipik işlerin  $\mu$  değerinin %60-%70 dolaylarında olduğu gözlenmiştir. Doğal olarak bu oran işten işe değişebilir.

"Uniprogramming" yoluyla  $\mu$  değerinin %50 olduğunu saptandığı iki işi "MultiProgramming" ile işleyecek olursak sistemin efektif G/C bekleme yüzdesinin ( $\mu'$ ) sıfıra düşeceği söylenebilir mi?

"MultiProgramming" modeline tekrar dönecek olursak, bunun anlamının  $C_1 = I_1 = C_2 = I_2$  olduğunu görebiliriz. Dolayısıyla  $I_1$  ile  $C_2$  ve  $I_2$  ile  $C_1$  işlemleri çakıştırılabilir ve buradanda ( $\mu'$ )nün sıfıra düşeceği söylenebilir.

Ancak daha önce belirtildiği gibi bu model yeteri kadar gerçekçi değildir. Daha iyİ bir modellemeyi (Şekil-34)'deki gibi yapabiliriz.



$$\mu_1 = \%50$$

$$\mu_2 = \%50$$

Şekil-34. Multiprogramming Modeli

(Şekil-34)'de verilen modelde her periyodun  $C_{11}, I_{12}, C_{13}, \dots$  birbirine eşit olduğu varsayılmıştır. Eğer bu iki işi multi programlamaya tabi tutacak olursak  $I_{12}$  ile  $C_{21}$ 'i ve  $I_{22}$  ile  $C_{13}$ 'ü çakıştırabiliriz. Ancak İş-1  $I_{14}$  durumuna geldiğinde İş-2 de  $I_{23}$  durumunda G/Ç yapmak için bekliyor. Bu nedenle CPU ya  $I_{14}$  veya  $I_{23}$  işleminin yapılması için boş beklemek zorundadır. Görüldüğü gibi  $\mu$  değeri %50 olan iki işin multiprogramlanması, sistemin efektif G/Ç beklemeye yüzdesi  $\mu'$ nün sıfıra inmesi anlamına gelmez..

Bu ikinci multiprogramming modelini genelleştirip, olasılık teorisine başvuracak olursak sistemin efektif G/Ç beklemeye yüzdesine ilişkin daha anlamlı bilgiler edinebiliriz. Şöyleki, sistem G/Ç işlemlerinin bitmesini ancak bütün işlerin G/Ç yapmak için uğraştığı bir zamanda beklemek zorluğunda kalır. Eğer  $n$  tane iş multiprogramlanıyorsa ve her işin G/Ç beklemeye yüzdesi ( $\mu$ ) birbirine eşit ise, sistemin toplam G/Ç beklemeye yüzdesi ( $\mu'$ ) yaklaşık olarak  $\mu' = \mu^n$  alınabilir.

Örneğin : Eğer  $\mu = \%50$  ise,

<u>İş Sayısı ( n )</u>	<u>Sistem G/Ç beklemeye yüzdesi ( <math>\mu'</math> )</u>
1	$(0.5)^1 = \%50$
2	$(0.5)^2 = \%25$
3	$(0.5)^3 = \%12.5$
4	$(0.5)^4 = \%6.3$
5	$(0.5)^5 = \%3.1$
6	$(0.5)^6 = \%1.6$

Ancak bu tabloda verilen değerler, her işin her periyod içinde bir adım ileri gittiği varsayılarak elde edilmiştir. Eğer sistem  $N$  CPU'dan ve  $N$  G/Ç kanalından oluşmuş olsaydı bu değerler doğru olacaktı. Fakat bu durumda  $\mu'$  anlamı,  $N$  CPU'nun aynı zamanda boş kalma olasılığı olacaktır. Bu olasılığı  $N$  tane parayı havaya atıp hepsinin yazı veya tura gelme olasılığına benzetebiliriz.

Söz konusu bilgisayar sisteminde sadece bir CPU'nun ve birkaç multiplexor G/Ç kanalının bulunduğu varsayıacak olursak  $N$  tane G/Ç işleminin paralel olarak yürütülebileceğini de varsayıabilirmiz. Doğal olarak tek bir CPU bulunduğuundan CPU'ya gelen işlerde bir yığılma söz konusu olacaktır.

Bu gibi durumlarda olasılık teorisinde "Birth-and-Death Markov Process" i adı verilen matematiksel bir teknikle çözümlenebilir. Burada sadece "Birth-and-Death Markov Process" tekniği kullanarak sistemin efektif G/Ç beklemeye zamanına daha geçerli yanıt verecek bir formülü tanıtmakla yetineceğiz.

$$\mu' = \frac{\left(\frac{\mu}{1-\mu}\right)^n}{n! \sum_{i=0}^n \frac{\left(\frac{\mu}{1-\mu}\right)^i}{i!}}$$

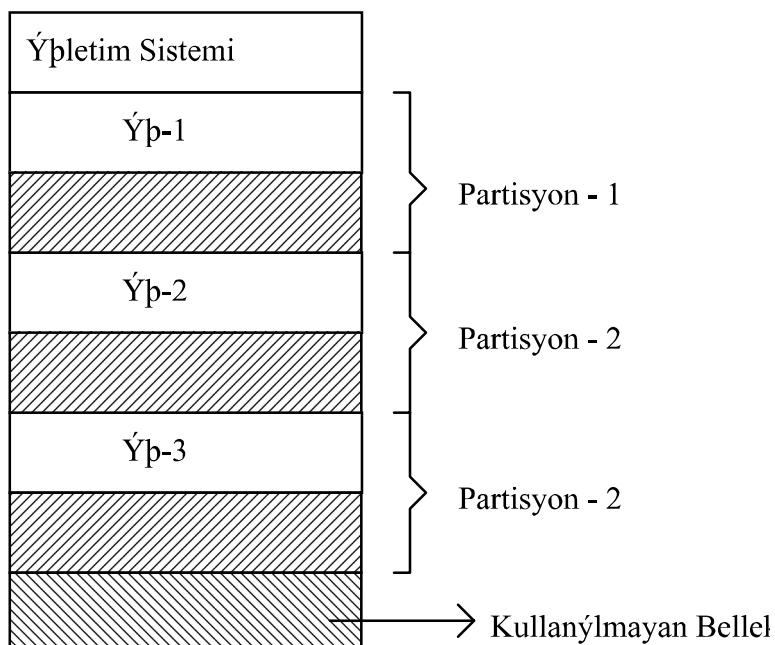
Bu formül  $\mu'$  için  $\mu' = \mu^n$  formülünden daha düşük değerler vermektedir. Nedeni ise  $n$  tane CPU yerine sadece tek bir CPU'nun kullanılması varsayımdır.

Burada belirtilmesi gereklidir ki, G/C bekleme zamanı oranı ile ilgili formüller bizleri gerçekle yaklaştıracı nitelikte formüllerdir. Önemli olan nokta G/C bekleme zamanının, uygulanacak olan "Multi-Programming" tekniğinin derecesine göre düşüş göstermesidir. Bundan sonra ele alacağımız bellek yönetim teknikleri "Multi-Programming" tekniğinin uygulanmasını ön planda tutmaktadır.

#### 10.2.1.2. Partitioned ( Bölümlere Ayırarak ) Bellek Yönetimi

"Multi-Programming" tekniğinin uygulanmasında kullanılabilen en basit bellek yönetim tekniklerinden biri Partitioned ( Partisyonlu ) bellek yönetimi tekniğidir.

Adında da anlaşılacağı gibi partisyonlu bellek yönetiminde bellek birbirinden bağımsız partisyonlara yani bölümlere ayrılmaktadır ve her bölüm ayrı bir işin adresleme alanını oluşturmaktadır. (Şekil-35 ).

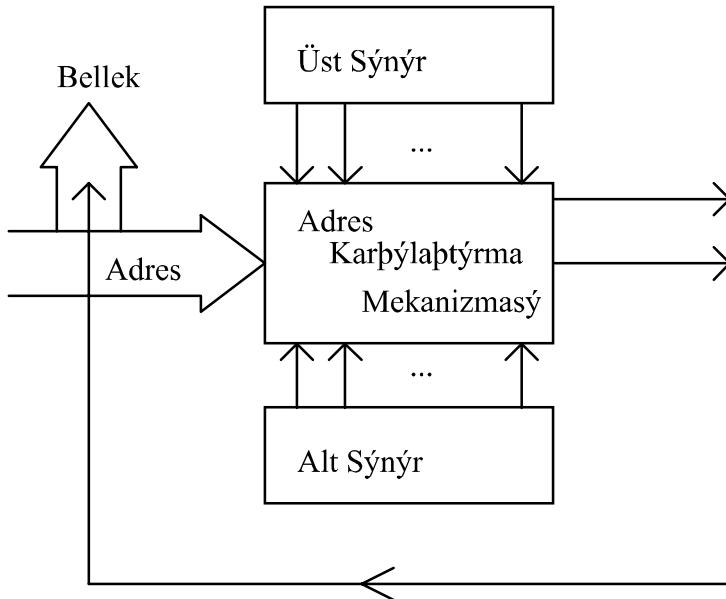


Şekil-35. Partisyoned Bellek Yönetimi.

#### Donanım Desteği

Genellikle partisyonlu bellek yönetimi az ve basit donanım desteği gerektirmektedir. Bu donanım özellikle bir işin diğer bir işe ait adresleme alanına veya işletim sisteminin bulunduğu alana müdahalesinin önlemek amacını gitmektedir. Single-Contiguous bellek yönetimini incelerken sözünü ettigimiz gibi, bu görev bir

protection (koruma) mekanizması tarafından yapılmaktadır. Yalnız partisyonlu bellek yönetiminde her partisyon için iki sınırlama registerinin kullanılması gerekmektedir. Bu registerlerden biri partisyonun üst sınırını diğer ise alt sınırını belirler. ( Şekil-36 )



Şekil-36. Koruma mekanizması

Multi-Programming yapan bir sistemde G/C kanallarının da adresleyebilecekleri alanlar sınırlanmalıdır. Örneğin üçüncü partisyonda bulunan bir iş birinci partisyonındaki bilgi okuması için G/C kanalının adresleyebileceği alanlar da iki sınırlama registeriyle belirlenmelidir. Buna karşın G/C kanallarına ilişkin adresleme işleri işletim sistemi tarafından yani yazılım kanalıyla denetlenebilir. Ancak böyle bir yaklaşım adresleme zamanı açısından sakıncalı olabilir.

### **Yazılım Desteği**

Uygulamada partisyonlu bellek yönetimi önumüze çeşitli şekillerde çıkmaktadır. Bunları genel olarak Statik Partisyon'lu ve Dinamik Partisyon'lu teknikler olarak iki sınıfa ayıralım.

Statik Partisyon'lu uygulamalarda partisyonların yerleri ve büyüklükleri sistem kullanıma açılmadan önce belirlenmelidir. Örneğin IBM'in OS/370 MFT ( Multi-Programming with a fixed number of tasks ) adlı işletim sisteminde partisyonların sayısı, yerleri ve büyüklükleri opetatör tarafından sistem on-line duruma gelmeden önce belirlenmelidir. Örneğin;

Partisyon No	Kapasitesi	Başlangıç Adresi	Statüsü
1	8K	312K	1 (Kullanılıyor)
2	32K	320K	1
3	32K	352K	0 (Kullanılmıyor)
4	120K	384K	0
5	520K	504K	1

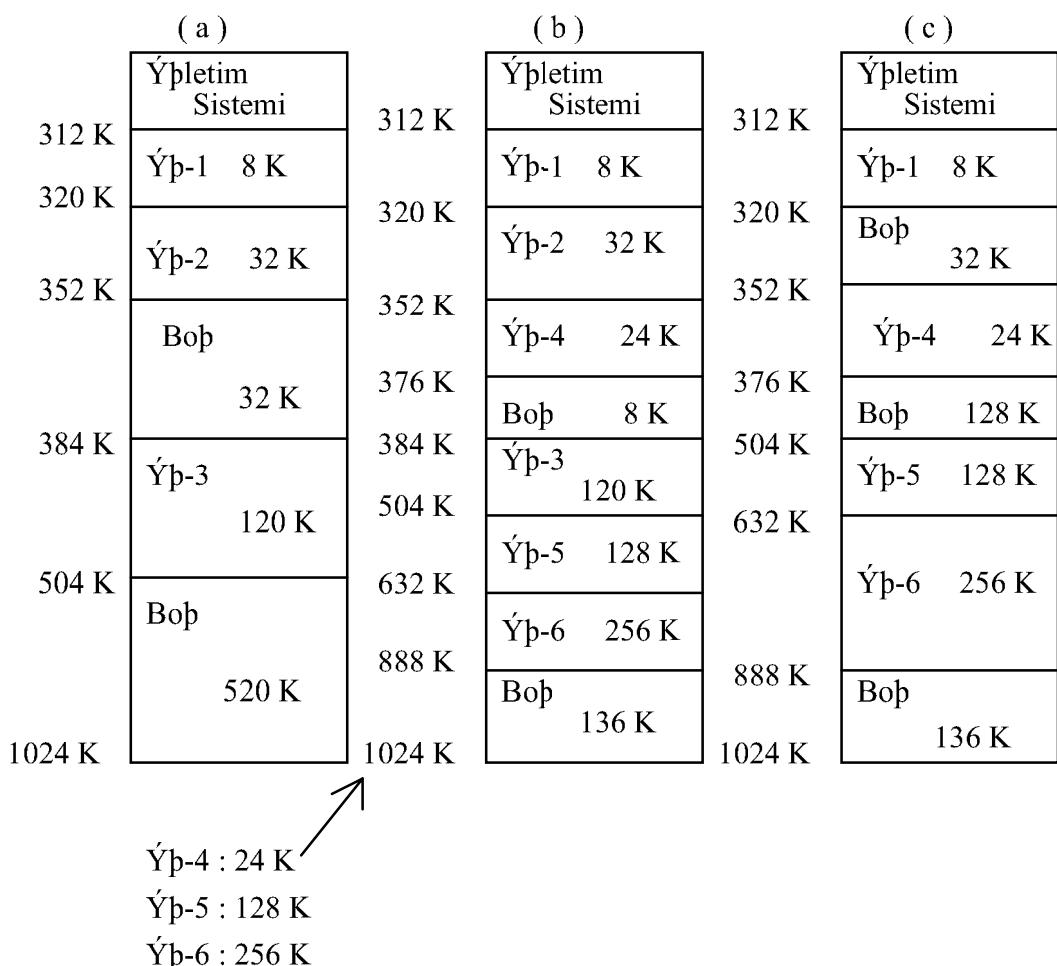
Statik Partisyon'lu bellek yönetimi çalıştırılacak programların büyülüklerinin ve işlem frekanslarının bilindiği durumlarda oldukça elverişli bir tekniktir. Fakat programların büyülükleri ve işlem frekanslarının büyük farklılıklar gösterdiği durumlarda belleğin verimsiz bir şekilde kullanılmasına yol açmaktadır. Örneğin; yukarıda partisyon konumunu belirlediğimiz belleği kullanarak 1 K, 2\*9 K, 33 K ve 121 K büyülüğündeki 5 iş için yapılacak bellek atamaları şu şekilde olabilir;

Partisyon No	Kapasitesi	İş Büyüklüğü	Statüsü
1	8K	1K	1 (Kullanılıyor)
2	32K	9K	1
3	32K	9K	0 (Kullanılmıyor)
4	120K	33K	0
5	520K	121K	1
	712K	173K	539K

Göründüğü gibi işlerin partisyon kapasitelerine en uygun bir şekilde yerleştirilmelerine rağmen 712 K'lık belleğin yalnız 173 K'lık bölümü, yani %25'i kullanılmakta geri kalan %75'i ise hiçbir işe yaramamaktadır. Bu örnek abartılmış gibi görünüyorsa da gerçek uygulamalarda benzeri oranda bellek harcanması kolaylıkla söz konusu olabilmektedir.

Statik Partisyon'lu uygulamalarda önumüze çıkan bu gibi iş büyülüğü partisyon kapasitesi uyumsuzluğunu gidermek amacıyla Dinamik Partisyon atama teknikleri geliştirilmiştir.

Dinamik Partisyon'lu uygulamalarda partisyonların sayısı büyülükleri ve bulundukları yerler sisteme girilen işlere bellek gereksinimlerine göre yani dinamik olarak belirlenmektedir. ( Şekil-37 )



Şekil : 37. Dinamik Partisyonlu Uygulamalarda Bellek Atama.

Belleğin herhangi bir andaki durumunu ( Şekil : 37 a )'da gösterildiği gibi varsayalım. Burada bulunan üç iş için o işlerin büyüklüklerine eşit büyüklükte üç partisyon atanmış olsun. Bir süre sonra üç ayrı işin daha sisteme girilmesi istendiğini varsayalım. Bu yeni işin belleğe yerleştirilmesi ( Şekil : 37 b )'de gösterildiği gibi olabilir. Yine bir süre sonra İş-2 ve İş-3 ün bittiğini varsayırsak belleğin bu işlerin devreden çıkışlarından sonraki durumu da ( Şekil : 37 c )'de gösterildiği gibi olacaktır.

Göründüğü gibi yeni bir işe partisyon ataması yapılırken izlenen yöntem;

- 1)** Kapasitesi en az işin büyülüğüne eşit olan bir partisyonun bulunması,
- 2)** Eğer partisyon kapasitesi işten büyükse, işe yetecek sahanın atanması ve geriye kalan kısmın ise boş bırakılması,

dır.

Benzer olarak işi biten partisyonların tekrar kullanılmak için sisteme kazandırılmasında izlenen yöntem ise; herhangi bir partisyon boşaltığında o partisyonla komşu olan partisyonların kullanılıp kullanılmadığının kontrol edilmesi, eğer kullanılmayan varsa boşalan partisyonun onlarla birleştirilmesidir.

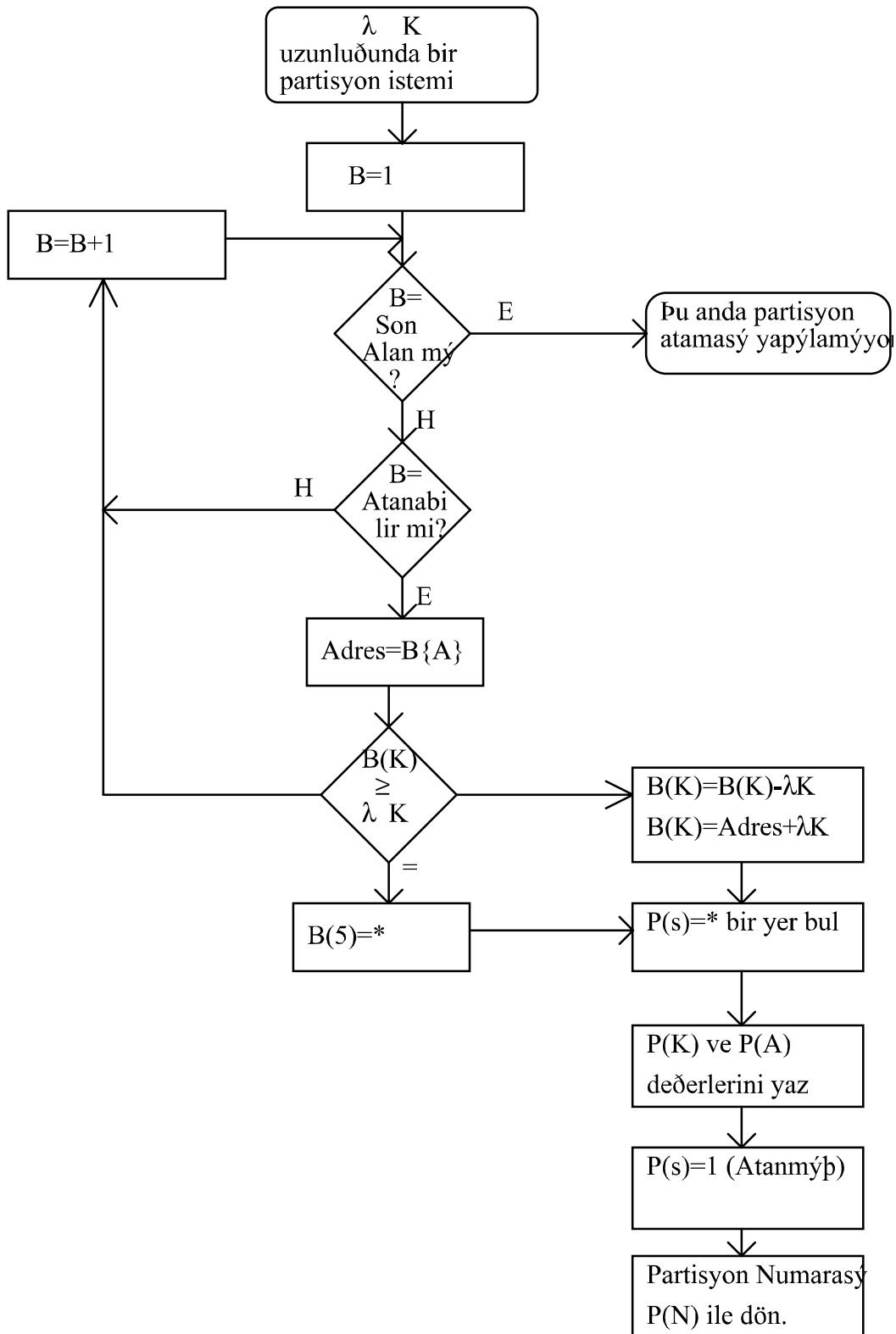
Buraya kadar söylenenlerden anlaşılacağı üzere, Dinamik Partisyon atama uygulamalarında belleğin hem kullanılan hemde kullanılabilecek durumda olan böülümleriyle ilgili statü bilgilerinin tutulması gereklidir. Örneğin, belleğin ( Şekil-37 a )daki durumundaki statüsünü şu şekilde tutabiliyoruz.

#### Atanmış Partisyonların Statüsü

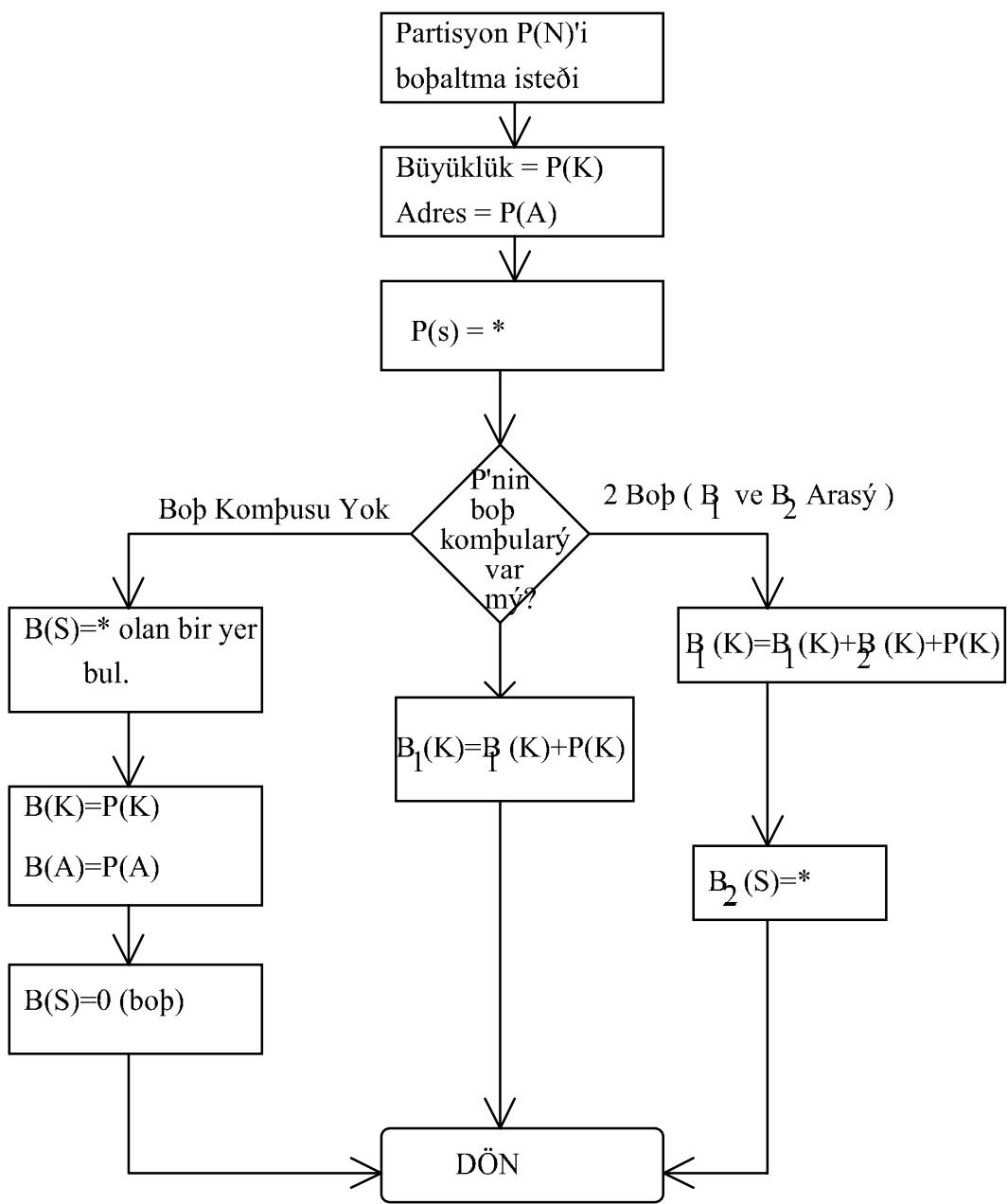
Partisyon No	Kapasite(K)	Adres(A)	Statüsü
1	8K	312K	1 (Atanmış)
2	32K	320K	1
3	-	-	* (Boş Bilgi alanı)
4	120K	384K	1
5	-	-	*
.	.	.	*
.	.	.	*
.	.	.	*

Bellek statüsü ile ilgili bilgilerin bu şekilde tutulduğunu varsayıarak Dinamik Partisyon'lu bellek yönetiminde partisyon atamalarının ( Kaynak Atanması ) ve işi biten partisyonların tekrar sisteme kazandırılması ( Kaynağın Geri Alınması ) işlemlerinin nasıl yapıldığını algoritmik olarak inceleyelim.

## Dinamik Partisyon Atama Algoritması



## Dinamik Partisyon Geri Alma Algoritması



Gerçek sistemlerde Dinamik Partisyon atama uygulamaları iki ayrı şekilde karşımıza çıkmaktadır. Bunlardan biri First-Fit ( ilk-uygun ), diðeri ise Best-Fit ( En-uygun ) atama teknikleri adları ile bilinmektedir.

Algoritmik yönden ele alınacak olunursa her iki teknikte yukarıda incelediðimiz algoritmanın aynısını incelemektedir. Ancak First-Fit tekniðinde kullanılan boş alanlar listesi, bu alanların adreslerine göre dizilmişlerdir; yani en küçük adreslidен en büyük adresliye doğru sıralanmıştır. Yeni bir partisyon istendiðinde arama en işlemi en küçük adresli boş alandan başlatılmakta ve karşılaþılan ilk yeterli boş alan istenilen partisyonu oluþturmaktadır.

First-Fit tekniğinin belli başlı iki avantajı vardır;

**1)** Partisyonu geri alma algoritmasında belirtilen " bu partisyon'a komşu başka boş alan var mı? " sorusuna yanıt verebilmek için ortalama olarak boş alanlar listesinin sadece yarısını taramak gerekiyor. Bunun yanında eğer varsa küçük adresli boş komşu alan, ilk bulunan alan olmaktadır.

**2)** Bu teknik mümkün olduğu kadar belleğin küçük adresli böülümlerindeki boş alanların kullanılmasını sağlamaktadır. Böylelikle kapasitesi büyük olan boş alanlar belleğin büyük adresli böülümlerde oluşup, birikim özelliği göstermektedirler. Dolayısıyla büyük bir iş için partisyon istendiğinde belleğin bu bölümünde istenilen büyülükte bir partisyon oluşturulması şansı büyümektedir.

Best-Fit tekniğinde ise boş alanlar listesi, alanların kapasitesine göre sıralanmıştır; yani en küçük kapasiteli boş alan listenin başında, en büyük kapasiteli boş alan ise listenin sonunda yer almaktadır. Böylelikle bir iş için partisyon istenildiğinde yapılacak iş, listenin başından başlayarak o iş için yeterli olabilecek ilk boş alanın hangisi olduğunu bulmaktır. Bulunan bu boş alan aynı zamanda o iş için en uygun ( Best-Fitted ) partisyon olmaktadır. Best-Fit tekniğinin belli başlı üç avantajı vardır. Bunlar;

**1)** Ortalama olarak Best-Fit sağlayan boş alanı bulmak için boş alanlar listesinin sadece yarısını taramak gerekiyor,

**2)** Eğer tam istenilen kapasitede olan bir boş alan varsa Best-Fit tekniği bu alanı bulmaktadır. First-Fit tekniğinde aynı seçimin yapılması söz konusu olmayabilir.

**3)** Eğer tam istenilen kapasitede bir boş alan yoksa ona yakın ( en-uygun ) boş alan seçilir ki, bu da First-Fit tekniğinde görülen gereğinden büyük boş alanların parçalanmasını önler ve bu gibi alanların daha büyük işler için kullanılması şansını arttırmaktadır.

Ancak, burada şunuda belirtmek gerekiyor ki, Best-Fit kriteri sağlanırken, her zaman istenilen boş alana en yakın seçildiğinden bellekte hiçbir işe yaramayacak kadar küçük boş alanlar oluşmaktadır. Bu durumda Best-Fit tekniğinin bir dezavantajı olarak vurgulamak gerekiyor.

Partisyonlu bellek yönetimi uygulamarının bir sonucu olarak bellekte birbirinden ayrı irili ufaklı boş sahaların oluşması olayına Fragmentasyon denir.

(Şekil-37c)'de verdigimiz Dinamik Partisyon atama örneğine dönecek olursak atanabilecek en büyük partisyon alanı 136 K büyülüğündedir. Bu durumda 137 K'lık bir partisyon istenecek olsa, şimdije kadar incelediğimiz algoritmaların "Yer Yok!" yanıtını alacaktık; bellekte toplam 296 K'lık boş alan bulunmasına rağmen isteğimiz karşılanmayacaktır.

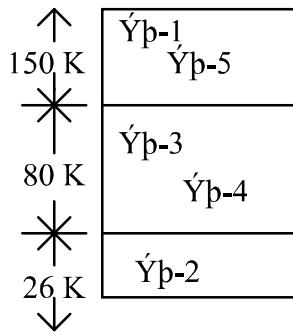
Partisyonlu bellek yönetiminde algoritma seçiminin dikkatli yapılmasıyla fragmentasyon sorununun etkisi minimum bir düzeyde tutulabilir. Şimdije kadar

incelediğimiz statik partisyon, dinamik First-Fit ve dinamik Best-Fit atama tekniklerinden herbirinin fragmentasyon sorununun etkilerini minimum bir düzeyde tutacak uygulamalar bulmak mümkündür.

Örneğin şöyle bir iş akışı söz konusu olsun;

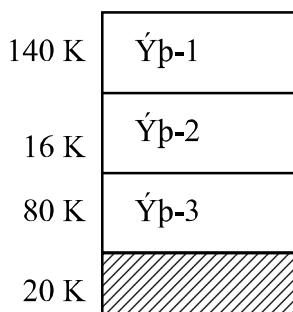
İş-1	140 K
İş-2	16 K
İş-3	80 K
İş-1	Sonuçlanıyor
İş-3	Sonuçlanıyor
İş-4	80 K
İş-5	128 K
.	
.	
.	

Şimdi elimizde 256 K'lık belleğin bulunduğu varsayıarak üç algoritmanın hangisinin bu iş akışı için elverişli olacağını araştıralım( Şekil-38 ).



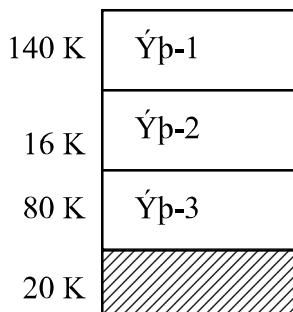
Týkanma yok, Algoritma geçerli

a) Statik Partisyon Atama.



Týkanma var, Yp-5 için partisyon yok. Yp-2 veya Yp-4'ün sonuçlanması beklemek gerekiyor.

b) Dinamik First-Fit.



Týkanma yok,  
Algoritma geçerli

c) Dinamik Best-Fit.

### Þekil : 38. Partisyonlu Bellek Yönetim Algoritmalarýný Karþýlaþtýrýlmasý

(Şekil-38)'de görüldüğü gibi, dinamik First-Fit tekniği statik partisyon atama teknigine göre daha mantıklı bir teknik olmasına karşın ele aldığımız iş akışı için statik partisyon atama teknığının daha elverişli ( tikanma yaratmayan ) bir teknik olduğunu söyleyebiliriz. Sonuç olarak bir teknığın diğerinden daha iyi olup olmadığını

söyledebilmek için sisteme girilecek işlerin sırasını, büyüklüklerini ve işlem frekanslarını bilmek veya bu konuda tahminler ( istatistikler ) yürütmek gerekmektedir.

## Avantajlar

Partisyonlu bellek yönetiminin belli başlı üç avantajı vardır. Bunlar;

- 1) Multi-Programming'e elverişli bir bellek yönetim tekniğidir, dolayısıyla CPU ve G/Ç ünitelerinin daha verimli bir şekilde kullanılmasını sağlar,
- 2) Özel amaçlı ve karmaşık bir donanım gerektirmez,
- 3) Teknik ile ilgili yazılım algoritmaları oldukça basittir.

## Dezavantajlar

- 1) Partisyonlu bellek yönetiminin en büyük dezavantajı fragmentasyondur. Fragmentasyonun etkisi kullanılacak olan bellek atama algoritmasına ve sisteme girilecek işlerin sıralanmasına başlı olarak değişir. Öyle iş sıralanmaları düşünülebilir ki, bellek kullanımı %10 un altına düşmektedir.
- 2) Single-Contiguous bellek yönetimine kıyasla hem daha büyük bir işletim sistemi hem de daha fazla bellek gerektirir.
- 3) Bir iş için gerekecek partisyonun büyüklüğü bellek kapasitesi ile sınırlanmaktadır. Ayrıca Single-Contiguous tekniğinde olduğu gibi bellekte çok az ve hatta hiç kullanılmayacak olan bir takım bilgilerde tutulmaktadır.

## Bellek Gereksinimi

Genelde ne kadar bellek satın alınmalıdır ? sorusuna yanıt verirken iki faktör göz önünde tutulmalıdır;

- 1) Sisteme girilecek işlerin ortalama uzunlukları,
- 2) İstenilen CPU kullanım yüzdesi.

1970-71 yılında LEHMAN tarafından IBM 360/65 ve 370/155 sistemleri üzerinde yapılan bir araştırmaya göre tipik bir iş 128 KB'lık bir bellek istemekte ve tek başına çalıştırıldığında, bu iş toplam zamanının %65 ini G/Ç beklemekle geçirmektedir.

Multi-Programming konusunu incelerken gördüğümüz gibi, efektif sistem G/Ç oranını %10 un altına düşürebilmek için en azından dört işin multi-programlanması gereklidir.  $\mu'$  ile ilgili tabloya bakılacak olunursa;

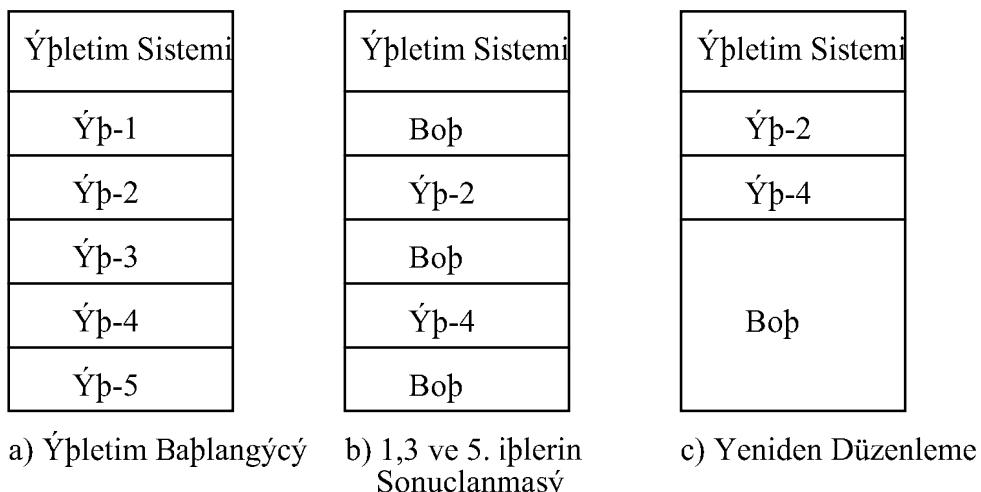
$$\begin{array}{llll} \mu = \%65 & N=4 & \text{für} & \mu' = 8.1 \\ & N=5 & \text{für} & \mu' = 2.9 \\ & N=6 & \text{für} & \mu' = 0.9 \end{array}$$

N=5 alınacak olunursa bellek gereksinimi  $5 * 128 = 640$  K olur. Doğal olarak partisyonlu bellek yönetiminde fragmentasyon söz konusu olduğundan extra belleğe

gereksinim vardır. Fragmentasyon ile belleğin 1/3 ünün harcadığını varsayıacak olursak, bellek kapasitesinin  $640+320=960$  K olması gerekmektedir. Buna işletim sistemi için gerekecek belleği de eklemek gerekir, bu da yaklaşık olarak 256 K ile 512 K arasında değişebilir.

#### **10.2.1.3. Relocatable Partitioned ( Yer Değişir Bölümlü ) Bellek Yönetimi**

Belleği bölümleyerek ( Partitioned ) düzenlemeye yönteminin en büyük sakıncası, ana belleğin parçalanmasıdır( Fragmentasyon ). İşletim sırasında işlerin nasıl davranışacağı tam kestirilemediği için, yapılan planlamalarda yetersiz kalmaya mahkumdur. Sorun bellek yönetiminde uygulanan politikanın durgun olmasından kaynaklanmaktadır. Başka bir deyişle işlerin, işletime başladıkları yerde bilinmeleri gerekmektedir. Oysa parçalanmaların arttığı dönemlerde kullanılan alanların ana bellekte kaydırılıp ard arda dizilmelerine olanak tanınsaydı serbest alanların bütünlüğüne gerçekleştirilebilirdi. (Şekil-39)



Şekil : 39. Belleğin Düzenlenmesi ( Compaction )

Ana bellekte işletim aşamasında yapılan bu kaydırma, dinamik bir bellek yönetim politikasını gerekmektedir. Çünkü yeri değiştirilen programların içerdikleri adreslerin yeniden belirlenen bir başlangıç adresine göre düzenlemeleri yeterli olmayacağındır. İşletim sırasında hesaplanan veri adresleri, işletim başlangıcındaki konuma göreli değer olacaklardır.

Böyle bir işlem prensip olarak basit olmasına karşın uygulamada çeşitli sorunlara yol açmaktadır. Şöyleki, bir işin adresleme alanı değiştirildiğinde o işin yeni adresleme alanında da doğru olarak çalışacağı söylemeyecez. Çünkü her işte veya programda base register'ler, parametre listeleri, data yapıları( Listeler,Kuyruklar,Stackler ) gibi adres duyarlı elemanlar bulunmaktadır. Dolayısıyla yerdeğişiminden sonra bir programın doğru olarak çalışmasına devam etmesini sağlamak için o programa ilişkin adres duyarlı elemanlarında gerektiği şekilde yeniden düzenlenmesi gereklidir. İşte adres duyarlı elemanlarının yer değiştirmesi işlemine Relocation adı verilmektedir. Ancak relocation

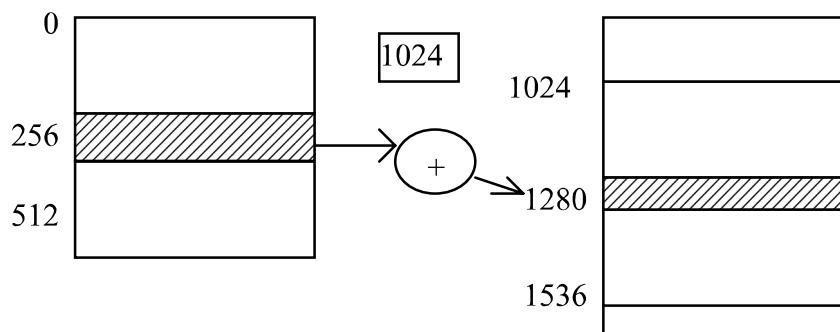
işlemi göründüğünden çok daha büyük bir sorun olarak karşımıza çıkmaktadır. Bir programda kullanılan adres ve pointerlardan hangilerinin değiştirilmesi gerektiğini bir işletim sistemi nasıl tayin edebilir? Programın içinde 364000 gibi sayısal bir değerin olduğunu varsayıyalım, acaba bu relocation sırasında değiştirilmesi gereken bir adres midir yoksa programa veri olarak verilmiş bir telefon numarası mıdır?

Bu tip ayırmayı yapabilmek için özellikle Reiter (1966-67) tarafından çeşitli yazılım teknikleri geliştirilmiştir. Ancak bu teknikler verimsiz ve kısıtlayıcı olmaları nedeniyle geniş çapta uygulanamamaktadırlar.

Relocation sorununa başka bir yaklaşımla yer değiştirecek programların yeniden yüklenmesi ve işlemlerine yeniden başlatılması olabilir. ( Relocatable Loader ) Ancak bu yaklaşımla CPU'nun etkin kullanımı söz konusu olamayacağı gibi, bazı durumlarda bir işe yeni baştan başlamamızda mümkün olmayabilir.

Relocation sorununa en uygun çözüm 'Donanım' veya 'Mapped memory' teknolojisinden kaynaklanmaktadır. Bu teknolojinin temel kavramını şu şekilde özetleyebiliriz;

Bir programın kendisine ait olarak gördüğü adresleme alanı o programın çalıştırıldığı belleğin fiziksel (gerçek) adresleme alanıyla aynı olmayabilir. (Şekil-40)



Şekil : 40. Taban Yazmacına Göreli Adresleme

Aslında bu kavram Virtual Memory veya Görüntülü Bellek adını verdığımız kavramın temelini oluşturmaktadır.

### **Donanım Desteği**

Relocatable Partitioned bellek yönetiminin gerektirdiği relocation işlemine donanım yönünden iki ayrı şekilde yaklaşılabilir.

Bunlardan biri Data Tipi kavramı üzerine kurulu olup, bellekteki her değerin ne tip data olduğunu fiziksel olarak kaydetmeye dayanmaktadır. Örneğin her bellek kelimesine(Word) iki extra tag biti eklenerek kelimelerdeki data tipini bu bitler yardımıyla belirtebiliriz. Örneğin;

- 00 : Integer
- 01 : Floating Point
- 10 : Karakter
- 11 : Adres

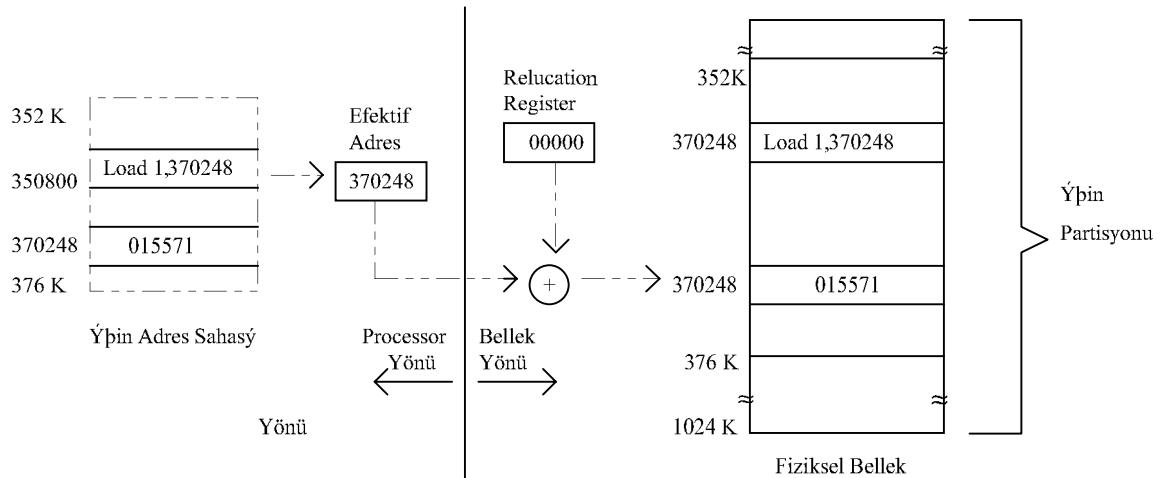
Bu bitler programcılar tarafından görülmemekte ve donanım tarafından otomatik olarak belirlenip işlenmektedir. Örneğin "A" adres değeri taşıyan, "I" ise integer değer taşıyan değişkenler olsun.  $I=A$  gibi bir deyim işlendiğinde A'ın değeri I'ya taşıdığı gibi A'nın tipini belirten (11) değeride I'nın tipini belirten (00) değerinin yerine taşınmaktadır. Böylece adres değerlerinin relocation amacıyla yer değiştirmesi ve işlenmesi sağlanmış olmaktadır.

Bir işe ilişkin partisyonun yer değiştirmesi gerekiğinde donanım, data tipini belirten bitlere bakarak adres tipini bulmaka ve bunlar üzerinde gereken relocation işleminin yapmaktadır. Burroughs 5500 ve 6700 serisi bilgisayarlarda bu tip bir relocation donanım mekanizması kullanılmaktadır.

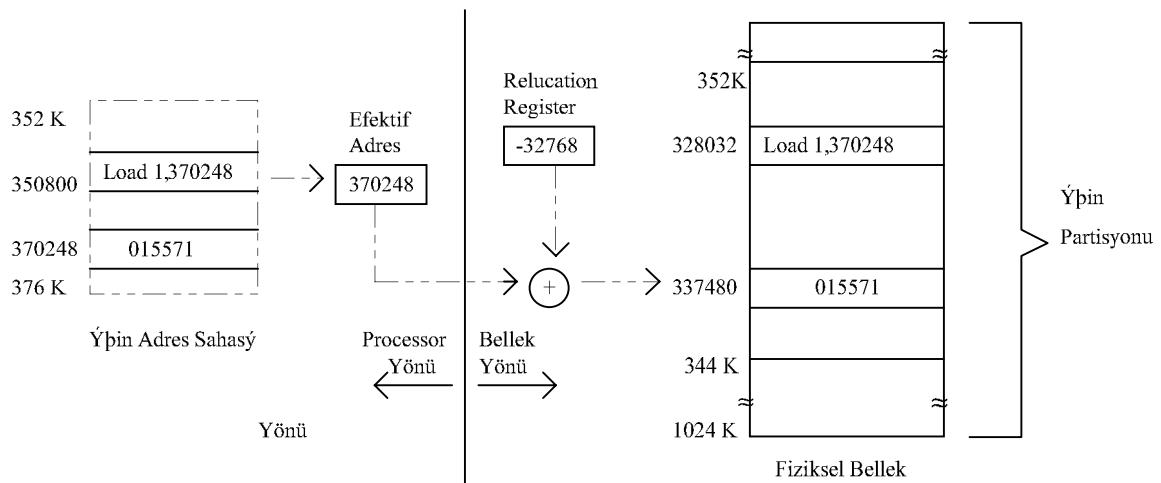
Data tiplerini belirleyerek relocation yapmanın belli başlı üç dezavantajı vardır. Bunlar;

- 1) Her kelime için iki extra tag bitinin sağlanması,
- 2) Her kelimenin data tipi kontrol edildiğinden relocation işlemi yavaş olmaktadır,
- 3) Günümüz bilgisayarlarının çoğunun genel yapısından ayıralık gösteren bir yapı oluşturduğundan "incompatibility" veya uyumsuzluk sorunları ortaya çıkmaktadır.

Diğer bir relocation tekniğinde ise sistem, Base Relocation ve Bounds registerleri olarak bilinen iki özel amaçlı registerle donatılmıştır. Dec PDP-10, Univac 1108 ve Honeywell 6000 serisi bilgisayarlarda kullanılan bu teknikle relocation işleminin nasıl yapıldığını bir örnekle inceleyebiliriz. Bellekte 352 K ile 376 K adresleri arasında yerleştirilmiş bir iş düşünelim ( Şekil -41 )



#### A) Relocationdan Önce



#### B) Relocationdan Sonra

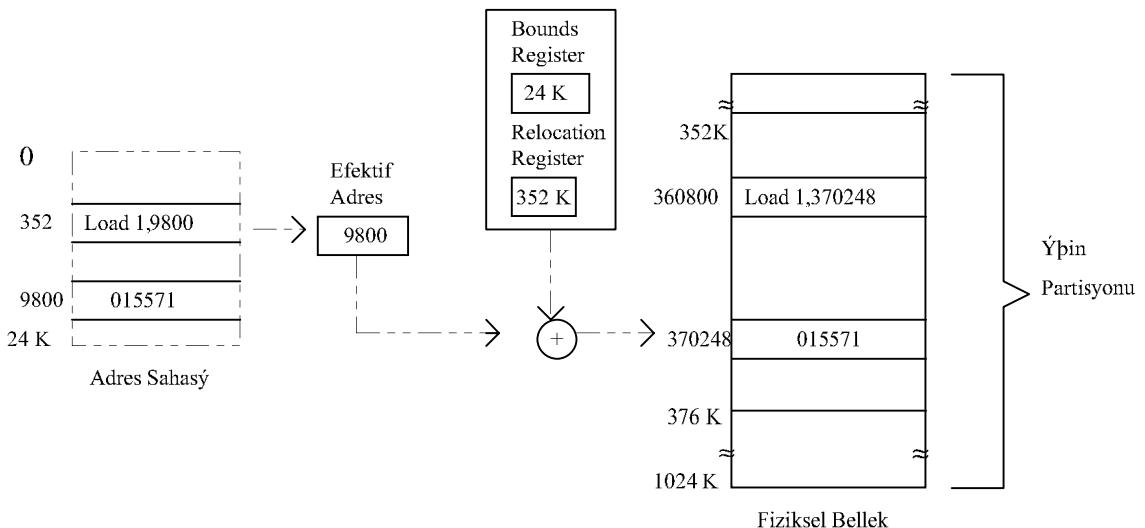
Şekil 41. Relocation registerin kullanýmý

(Şekil- 41)'de görüldüğü gibi program içinde belirlenmiş olan adreslerde herhangi bir değişiklik yapılmadığı gibi, program bellekteki fiziksel (geçer) konumunun da farkında olmadan çalıştırılabilir olmuştur.

Relocation işleminin otomatik olarak ve her komutun işlenmesi sırasında yapılması nedeniyle bu teknique Dinamik Relocation adı verilmektedir. Bounds registeri ise koruma amacıyla sınırlama registerlerinde olduğu gibi kullanılmaktadır.

## Yazılım Desteği

(Şekil-41)'den anlaşılacağı gibi dinamik relocation teknigi kullanıldığında, bir işe ait adresleme alanı o işin bellekteki gerçek konumundan bağımsız olarak düşünülebilir. Dolayısıyla her işin adresleme alanı sıfırdan başlatılabilir (Şekil-42).



Şekil 42. Yarıadresleme alaný sýfýrdan baþlatýlýyor.

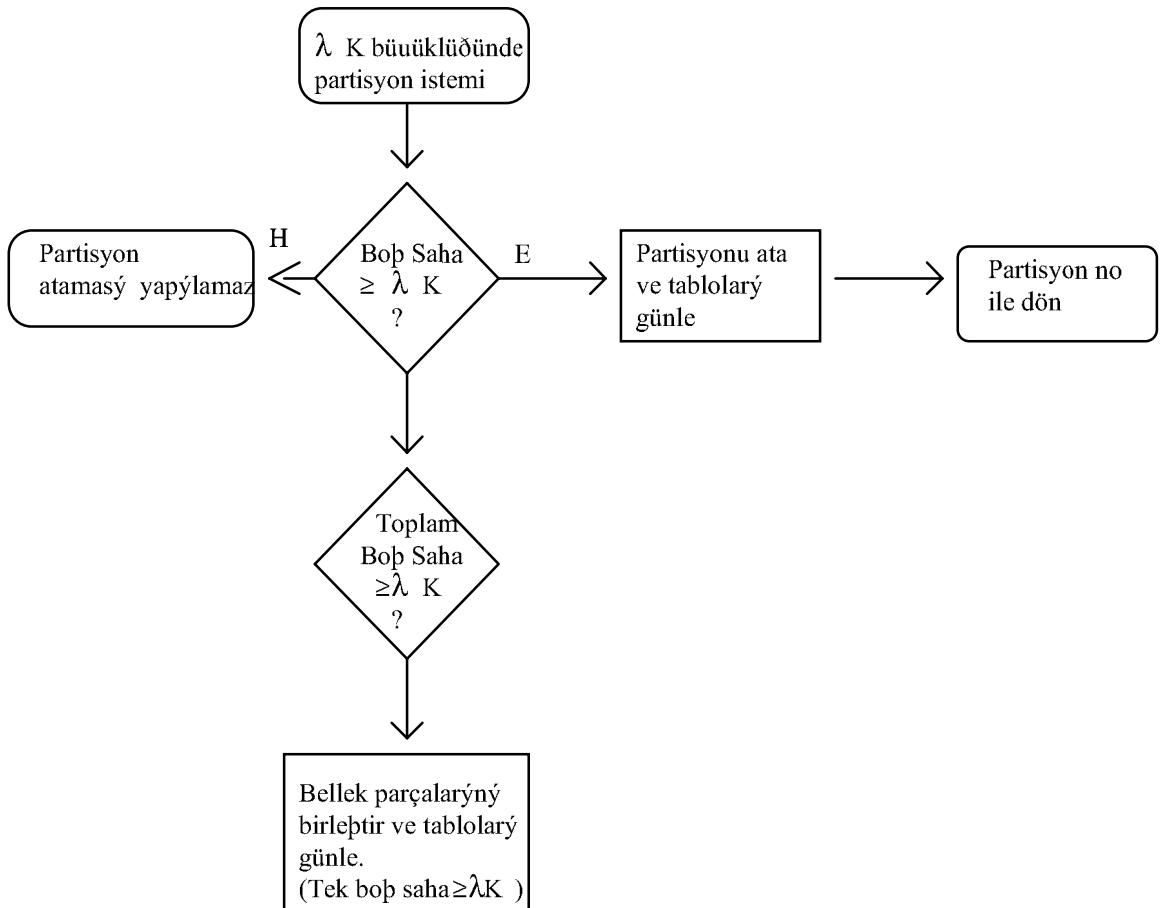
Relocatable partisyonlu bellek yönetiminde kullanılan algoritmalar, temel olarak dinamik partisyonlu bellek yönetimini anlatırken incelediğimiz algoritmaların aynıdır. Ancak burada ek olarak bir de "Compaction ne zaman yapılacak?" sorusuna yanıt vermek gerekiyor. Bu soruyu yanıtlamak için iki ayrı yol izlenebilir.

Bunlardan birinde compaction işlemi bir partisyon boşaldığında hemen yapılmakta ve fragmentasyona fırsat verilmemektedir. Böyle bir yaklaşımla boş alanların tümünün bir arada tutulması sağlandığı gibi boş alanlar listesi üzerinde yapılacak işlemler de basitleştirilmiş olmaktadır. Ancak compaction işleminin her partisyon başladığında yapılmasının CPU zamanı açısından sakıncaları vardır. Örneğin bir partisyonu bir yerden diğer bir yere taşıırken saniyede bir milyon byte transfer edebileceğimizi düşünecek olursak (IBM-370 MVC veya MVCL komutları ile), 1024 K'lık bir belleğin ortalarına doğru olan kççk bir partisyonun yerini değiştirmek için 1/2 saniyelik CPU zamanı gerekecektir. İş CPU zamanından büyük olabilir.

Diğer bir yöntem ise recompaction işleminin sadece yeni gelen bir iş için gereken yeterli boş bi alanın bulunmadığı zamanlarda yapmak olabilir. Böylelikle compaction için kullanılacak olan CPU zamanında önemli derecede düşüş olacaktır. Fakat buna karşın boş alanlar listesi ve partisyonlar listesi üzerinde yapılacak olan işlemler daha karmaşık ve zaman alıcı olacaktır.

Göründüğü gibi bu iki yöntem arasında bir seçim yapabilmek için sisteme girilecek işlerin büyüklükleri ve işlem frekansları ile ilgili bilgiler göz önünde

tutulmalıdır. Relocatable partition bellek yönetiminde partisyon atama algoritması (Şekil-43) 'de verilmiştir.



Şekil 43. Relocatable partition atama algoritması

## Avantajlar

Relocation partisyonlu bellek yönetiminin belli başlı avantajlarını şu şekilde sıralayabiliriz;

- 1)** Fragmentasyon sorununu ortadan kaldırır,
- 2)** Daha çok sayıda partisyonun atanmasını sağlar,
- 3)** Multi-Programming uygulama düzeyini yükseltir,
- 4)** Bellek ve CPU kullanım oranını arttırmır.

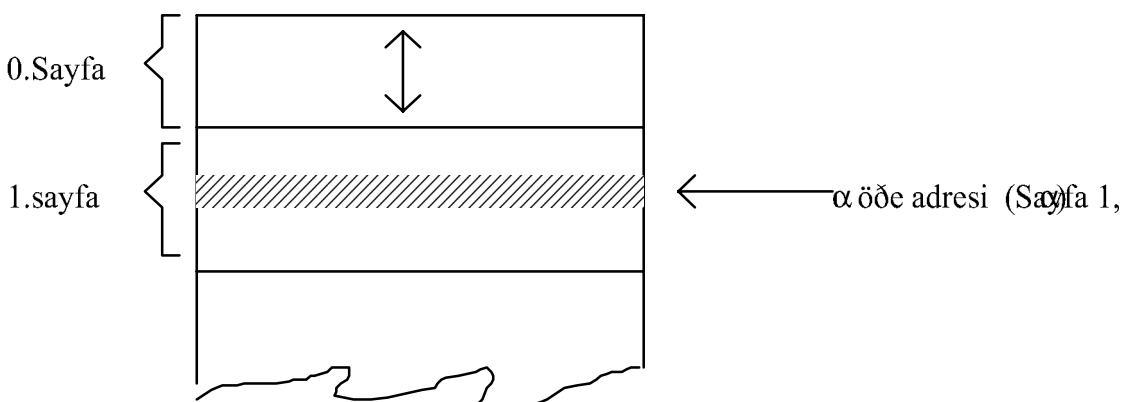
## Dezavantajları

- 1) Relocation donanımı sistemin fiyatını yükseltir,
- 2) Relocation donanımı sistem hızını azaltır,
- 3) Compaction zamanı yüksek olabilir,
- 4) Compaction yapılmasına rağmen bazı bellek bölümleri kullanılmayacaktır  
(Ya iş girişi olmadığından veya yeteri kadar boş bir alan bulunmadığından ),
- 5) Atanabilecek partisyon kapasitesi bellek kapasitesi ile sınırlanmaktadır.

### 10.2.1.4. Paged ( Sayfalı ) Bellek Yönetimi

Ana belleği yerdeğiştirir bölümlerle düzenlemek, parçalanmayı önlemesine karşın işletimin zaman zaman kesilerek bölümlerin ana bellekte kaydırılmaları nedeniyle pahalı bir çözümdür. Yöntemi bu açıdan incelediğimizde bunu, işlere atanacak kesimin tek bir bütün olarak işlenmesinden kaynaklandığını kolayca görebiliriz. İşlere atanacak bellek kesiminin devamlı olması zorunlu ortadan kaldırıldığından parçalanmayı önleyici daha etkin teknikler geliştirilmiştir. Bunlardan biri paging ( Sayfalama ) bellek yönetim tekniğidir.

Sayfalama yönteminin temelini adresleme alanı ile fiziksel alanın birbirinden bağımsız olabileceği düşüncesi oluşturmuştur. Bağımsızlık kavramını fragmentasyon ve devamlılık açısından en iyi bir şekilde değerlendirebilmek için adresleme alanı ve fiziksel alan eşit uzunlukta küçük parçalara bölünmüştür. Adresleme alanının küçük parçalarına "page", fiziksel alanın küçük parçalarına ise "blok" adı verilmektedir. Böyle bir yapı içinde bir ögenin adresi, içinde bulunduğu sayfa no ve sayfa başına göreli konumu olarak saptanır. ( Şekil : 40 )



Şekil:40. Bir öðenin görelî konumu

Mantıksal açıdan bakıldığından öğelerin program başlangıcına göreli konumları değişmektedir. Örneğin sayfa boyu  $x$  bellek birimi ise,  $\alpha$  ögesinin program başına göreli adresi;

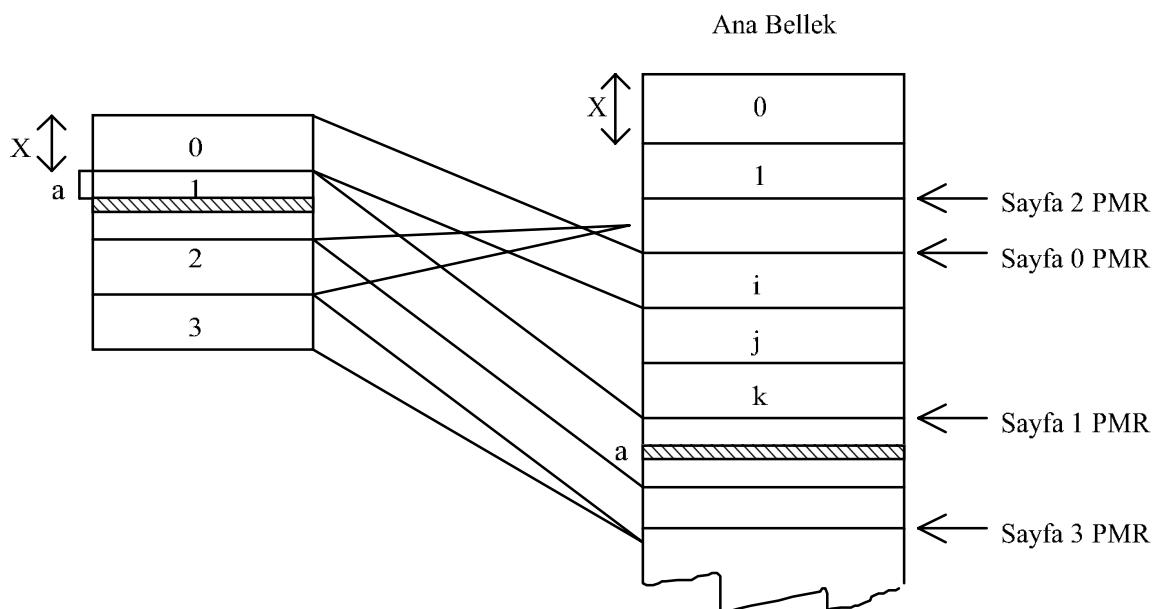
Sayfa-No \*  $x$  +  $\partial$     yada " $x+\partial$ "    dır.

Dinamik bellek yönetiminin ilkesini "programların adres alanlarının ana belleğin fiziksel konumlarından bağımsızlaşdırılmak" diye tanımlamıştık. Sayfalama yönteminde, bu ilkeyi "bir sayfa içindeki konumlarından bağımsızlaşdırılmak" olarak genişletiyoruz.

Relocatable bellek yönetiminden hatırlanacağı gibi gereken önlemler alındığında (Base Register) partisyonların bellekteki fiziksel konumu işlerin adresleme alanlarını etkilememektedir. Bu gerçek göz önünde tutularak page'ler ile blok'lar arasında öyle bir bağlantı kurulabilir ki (Page Mapping), page'ler mantıksal olarak devamlılık özelliğini koruyabilir. Ancak blok'larda devamlılık özelliğinin aranması gerekmektedir.

Page  $\longrightarrow$  Page-Mapping  $\longrightarrow$  Blok

Adresleme alanındaki page'lerin fiziksel alandaki blok'lara yansıtılması (Mapping) için her page'e ilişkin ayrı bir registerin bulundurulması gerekmekte veya belleğin belirli bir bölümü bu amaçla ayrılmalıdır. Bu registerlere page-map-registers (Şekil-41), bellek bölümüne page-map-tables adı verilmektedir.

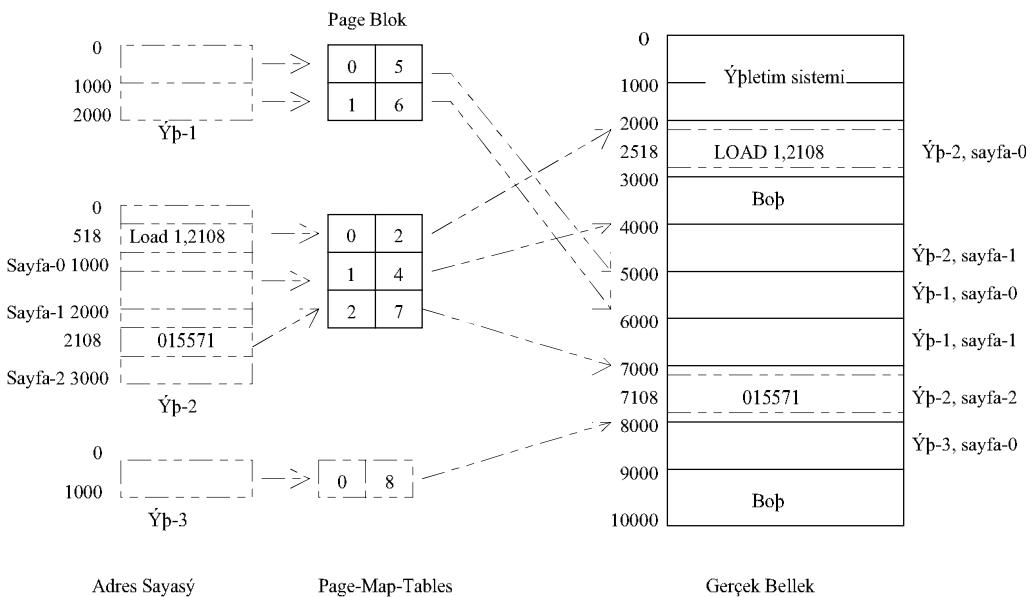


Şekil:41. Page-Map-Registerleri

Programlardaki her sayfa ana bellekte eşit uzunlukta yer kapsadığı için, boş / serbest konumların tutulma işlemi de kolaylaşacaktır. (Örneğin her sayfa için bir bit ayrılmış bir dizi biçiminde).

Paging ile devamlılık sorunu, bir partisyonun tümüyle devamlı olmasından bir sayafanın devamlı olması sorununa indirgenmiş olmaktadır. Dolayısıyla sayfa boyunun seçimi, bu yöntemin başarısındaki önemli etmenlerden biridir. Sayfa boyalarını çok büyük tutarsak, ana belleği yerdeğişir bölgelerle yönetmedeki sakıncalar bu çözümde de kendilerini gösterirler (Örneğin programın sayfa boyuna göre çok ufak olmasından kaynaklanan bellek kaybı). Sayfa boyu çok küçük olduğunda, bellek kesimlerinin sayısı artacağından page-map-register sayısı artacak ve dolayısıyla sistemin hızı düşecek, maliyet artacaktır. Sayfa boyunu saptarken; ana belleğin kapasitesini, üretilen amaç programların ana bellek birimi oranı, kullanılan verilerin bu birimlere oranları gözetilir. Günümüzdeki sistemlerde yaygın olarak kullanılan boyutlar 1K, 2K, 4K ve 8K bayt kapasitesindedir.

Sayfa boyunun uzunluğu 1K olduğu bir sistemde adresleme alanı ile fiziksel alan arasındaki bağlar (Şekil-42)'de gösterilmiştir.



Şekil-42: Adresleme alanı ile Fiziksel alan ilişkisi

(Şekil-42)'de gösterilen paging teknigini uygulayan bilgisayar sistemlerine örnek olarak XDS 940, XDS Sigma 7 ve CDC 3300 sistemleri verilebilir.

Bellek yönetiminin karşılaması gereken dört işlevi vardır;

- 1) Ana belleğin durumunun iki ayrı tür tablo ile izlenmesi;
  - a) Her iş için adres alanı-bellek ilişkisini kuracak registerlerin düzenlenmesi
  - b) Tüm bellekteki serbest-boş alanların belirlenmesi.
- 2) Ana belleğin kime atanacağını saptamak( İş yönetimi içinde çözülür.)
- 3) Boş sayfaları seçilen işe atamak üzere gerekli düzenlemeleri yapmak.)

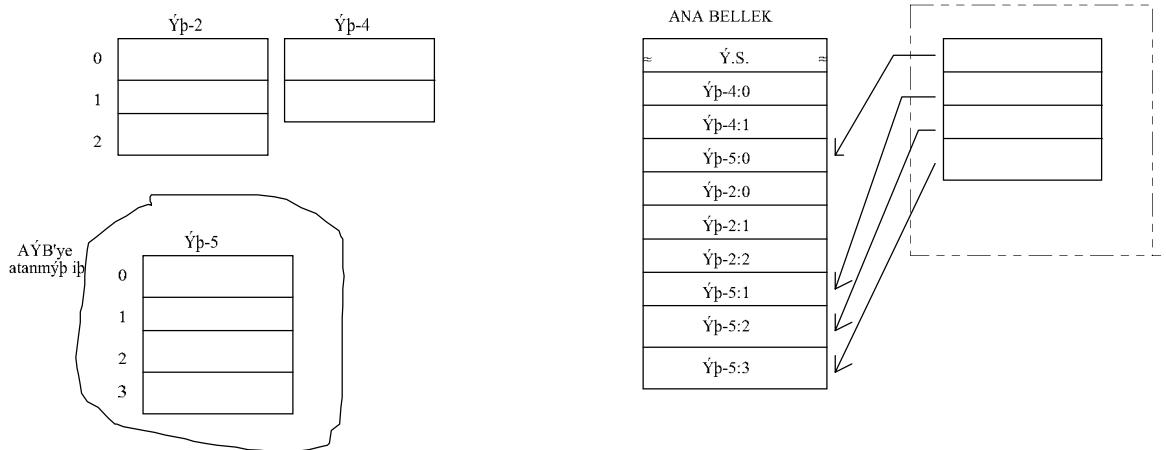
#### 4) İş bitiminde kullanılan alanların serbest bırakılması.

#### Donanım Desteği.

Programların işletiminde adres alanı-fiziksel adres arasındaki geçiş ya özel amaçlı registerler (page-map-register) ya da ana bellekte bu iş için tutulan tablolarda gerçekleştirilir.

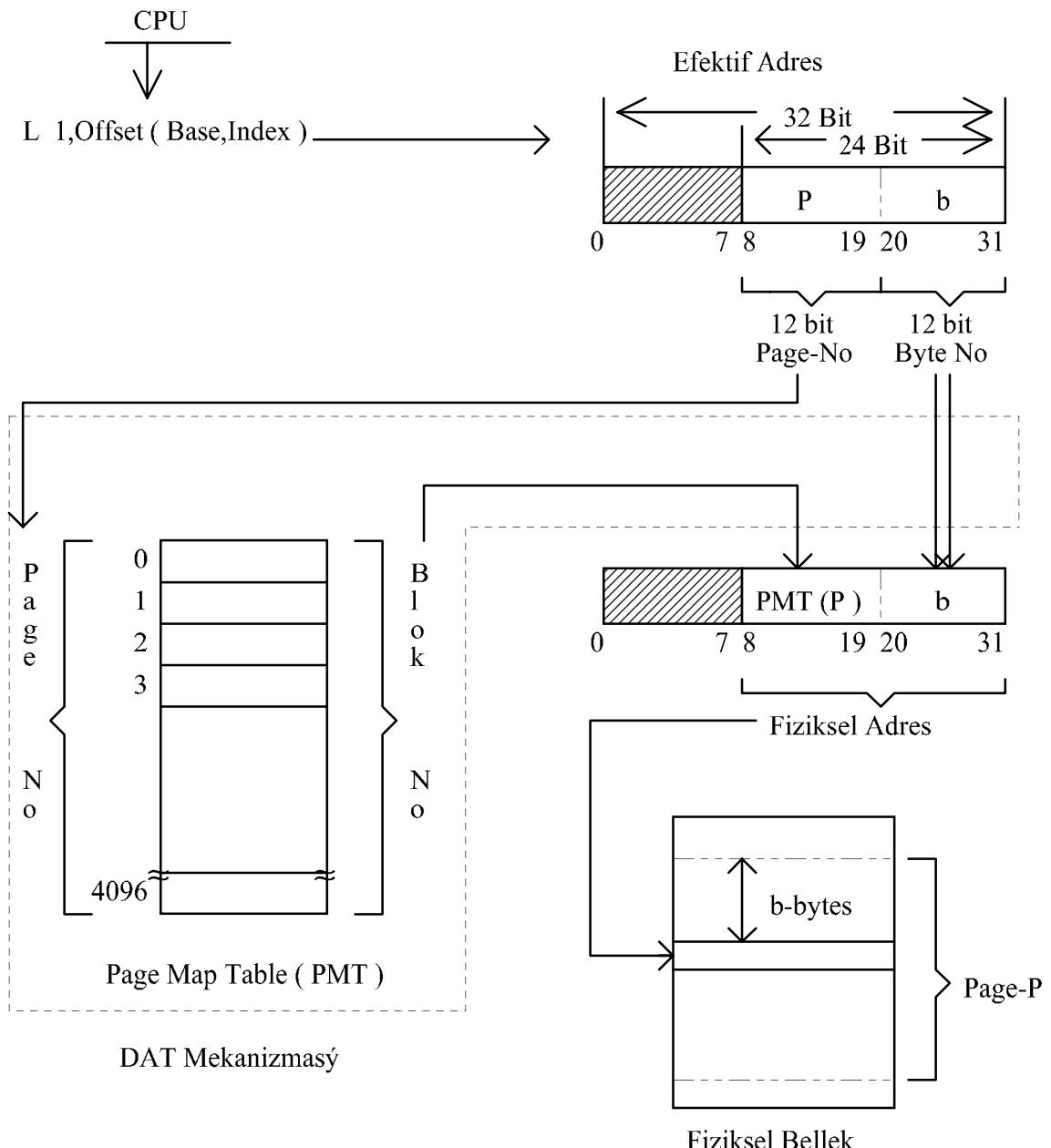
Özel amaçlı register kullanımı pahalı bir yaklaşımdır. Ana belleği adreslemek üzere, her işe (Bellek büyülüklüğü/ Sayfa büyülüklüğü) kadar register ayırmak gereklidir. İş maliyetinin artması sistemin maliyetini doğrudan artırır etken olmaktadır. Bu nedenle çoğu sistemlerde bu registerlerin adedi düşürülür (20-25 tane) ve herhangi bir anda sadece bir işin aktif olduğu gerçeği göz önünde tutularak kullanıcılar arasında paylaşımlı olarak kullanılırlar. Sistemde her yeni görev AİB 'ni kullanmak için anahtarlandığında bu registerlerin de yeni adres sahası ile günlenmesi gereklidir (Şekil-43). Örneğin XDS 940 serisi bilgisayarda bu yöntem uygulanmaktadır.

AİB taban register kümlesi



Şekil 43: Registerlerin Günlenmesi

Bu şekilde 25 tane registerin kullanıldığını varsayıp olursak, page uzunluğunun 4K olduğu bir sistemde çalıştırılabilen en büyük program 100 K olacaktır. Gerek daha büyük adresleme alanları oluşturmak gerekse registerlerin devreye giren her iş için yeniden yüklenip registerlerdeki eski bilgilerin saklanması büyük zaman kaybına neden olmaktadır. Bu sakıncayı önlemek üzere uygulanan çözüm, adresleri ana bellekte tablolardır (page-map-tables) biçiminde oluşturmaktır. Page-map-tables kullanımını açıklamak için IBM-370 serisi bilgisayarlarda kullanılan page-mapping veya Dynamic Address Translation (DAT) teknigi incelenebilir (Şekil-44). Bu yaklaşımda AİB'ye atanmış işin adres tablosunun bellek konumunu belirlemek üzere yalnız bir taban registerinin kullanımı yeterlidir (DAT registeri).

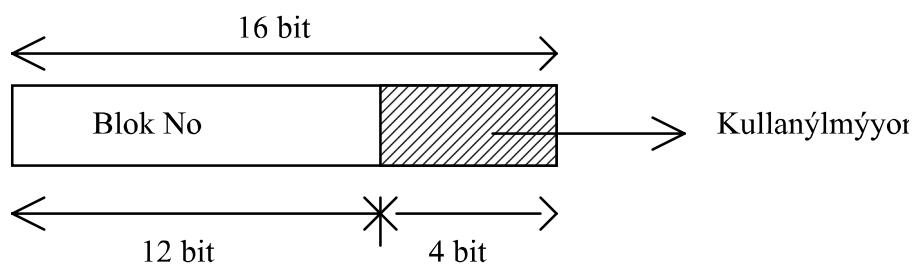


Şekil : 44. Dinamic Address Translation Tekniği

IBM-370 makinalarında page uzunluğu 1K-2K veya 4K olabilmektedir. Yine 370 komut yapısı hatırlanacak olursa efektif adres 24 bit uzunluğundadır. DAT mekanizması 24 bitlik efektif adresi otomatik olarak iki bölüme ayırmaktadır. Bunlardan biri 8-19. bitler arasında yer alan 12 bitlik page numarasına, diğer ise 20-31. bitler arasında yer alan o page'deki offseti belirtmektedir. DAT mekanizmasının bir bölümü olarak gösterilebilen Page-Map-Table efektif adresden elde edilen 12 bitlik page numarasının bellekteki bir blok'a yansıtılmasını sağlamaktadır. Eğer page-map-table bellekte tutuluyorsa, donanımın bu bilgilerin belleğin neresinde bulunduğu bilmesi

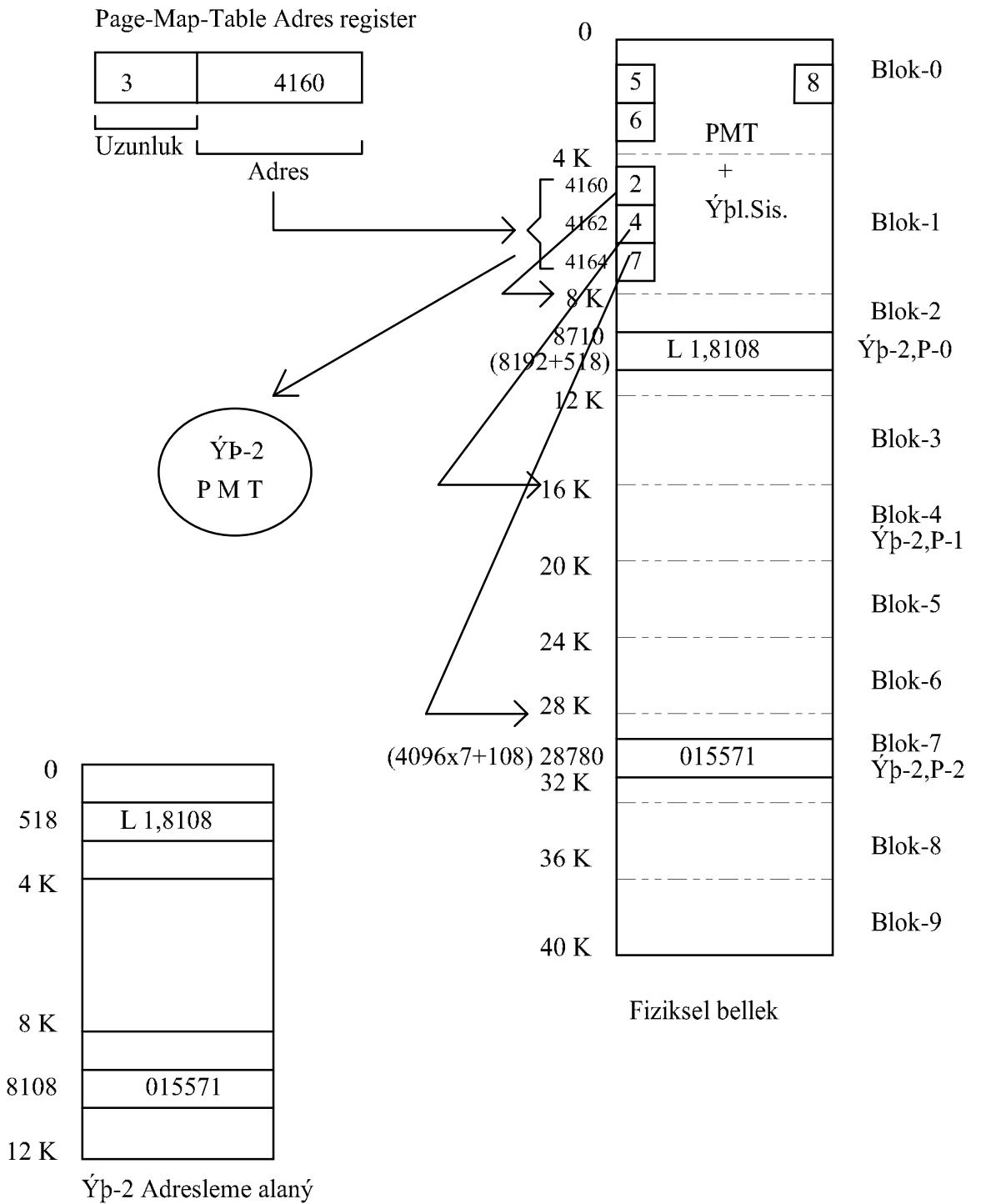
gerekir. Yani PSW, CAW, CSW'de olduğu gibi bu bilgilerin sabit bir yerde tutulması gereklidir. Ancak böyle bir yaklaşımla CPU'nun bir işten diğer bir işe geçişinde PMT'nin de yeniden düzenlenmesi gereklidir. Bu nedenle 370 sistemlerinde PMT'ler belleğin herhangi bir yerinde saklanabilir bir şekilde düzenlenmişlerdir ve PMT'lerin bulunduğu yerde Page-Map-Table-Address-Register (PMTAR) adlı bir registerde tutulmaktadır. İş değişiminde sadece PMTAR'yi yeniden yüklemek gereklidir. Dolayısıyla CPU kontrolü bir işten diğer bir işe geçtiğinde yapılacak işlem, yeni işin PMT'sini belirten adresin PMTAR registerine yüklenmesi olmaktadır.

PMT içindeki bilgiler çeşitli şekillerde düzenlenebilir. 370 sistemlerinde blok sayısı 4096 olabileceğinden blok numarası için 12 bitlik bir alan gereklidir, bu da 2 byte veya half-word içine (Şekil-45)'deki gibi yerleştirilmektedir.



Şekil:45. Blok Numarasýnýn Yerleþimi

Page, Blok, PMT ve PMTAR arasındaki ilişkiler (Şekil-46)'da gösterilmiştir.



Şekil: 46. Page,Blok, PMT ve PMTAR arasındaki ilişkiler

IBM-370 sistemlerinde bir işin adresleme alanı 16 Mbyte ( $4906 * 4K$ ) olabilir. Eğer her iş için PMT 4096 girişli bir liste olacak olursa, her iş için PMT 'in 8192 byte uzunluğunda olması gereklidir. Bu sorunu çözebilmek için de PMT'larının değişik

uzunluklarda, yani gerektiği kadar uzunlukta, yapılması öngörülmüş ve PMT 'nin uzunluğu PMTAR'nin bir bölümünde belirtilmiştir. Bu uzunluk parametresi bir yerde sınırlama registeri şeklinde görev yapmakta ve bağlı olduğu işin adresleme alanını sınırlamaktadır.

Eğer DAT işlemi bellekte oluşturulmuş PMT ler ile gerçekleştirilecek olursa bu sistemin % 50 hızı ile çalışmasına neden olur. Başka bir deyişle DAT % 100 yük getirir. Çünkü PMT 'ler bellekte saklanacak olursa her bellek erişimi söz konusu olduğunda önce PMT 'ye daha sonra da bellekte istenilen yere erişmek için iki bellek erişimi gerekecektir. CPU'nun yavaşlamasına neden olan bu sorunun gidermek amacıyla, PMT 'lerin ve page-map-registerlerinin beraber kullanılmasından Hybrit Page-Mat-Table'ları ortaya çıkmıştır.

Bu amaçla örneğin 16 tane özel amaçlı PMR bulundurulmakta ve bunlar PMT 'lerin bir kısmının tuttuğu bilgileri içermektedirler. PMR'lerin PMT 'lerden yüklenmeleri otomatik olarak yapılmakta, ne işletim sistemi ne de işler PMR-PMT ilişkisinin farkında olmamaktadır. Bu teknikle yük %100 den %10 un altına düşmektedir. PMR ler ve bunlarla ilgili donanıma Associative Memory veya Table Look-Aside Buffer adı verilmektedir.

### **Yazılım Desteği.**

İşletim sistemi temel olarak 3 ayrı tablo üzerinde işlem yaparak Paged Bellek Yönetimini gerçekleştirmektedir. Bunlar;

#### **1) Job Table (İş Tanım Tablosu)**

Sistemde çalışan işlerin tanımlandığı bu tabloda her işin uzunluğu, PMT 'sinin adresi ve statü bilgiler tutulmaktadır.

#### **2)Memory Block Table (Bellek Öbek Tablosu)**

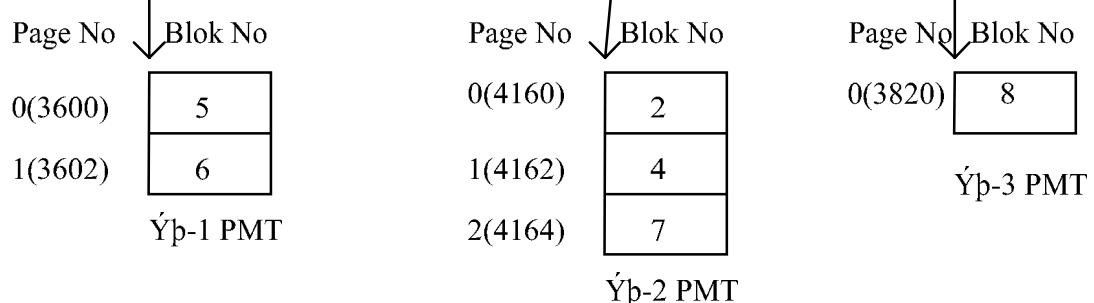
Her işin adres sahاسını belirleyen bilgiler (bellekteki her blok için statü bilgileri) tutmaktadır.

#### **3)Page Map Table**

Yazılım açısından tabloların kullanımı (Şekil-47)' de gösterilmiştir.

Ýþ No	Ýþ Uzunluðu	PMT Adresi	Statü
1	8 K	3600	Atanmýþ
2	12 K	4160	Atanmýþ
3	4 K	3820	Atanmýþ
4			Böþ

Job Table



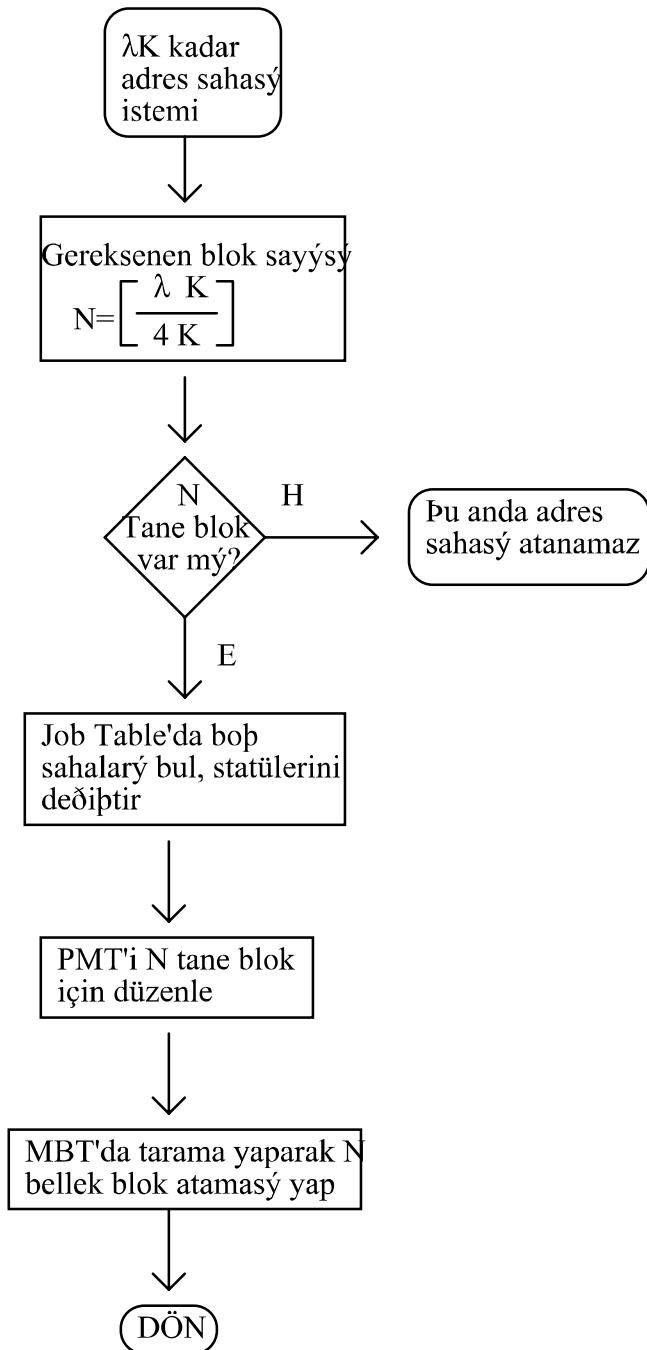
Page Map Tables

Blok No	Statü
0	Ý.S.
1	Ý.S.
2	Ýþ-2
3	Böþ
4	Ýþ-2
5	Ýþ-1
6	Ýþ-1
7	Ýþ-2
8	Ýþ-3
9	Böþ

Memory Blok Table

Þekil:47. Paged Bellek Yönetimi Tablolarý

Adres Sahası Atama Algoritması:

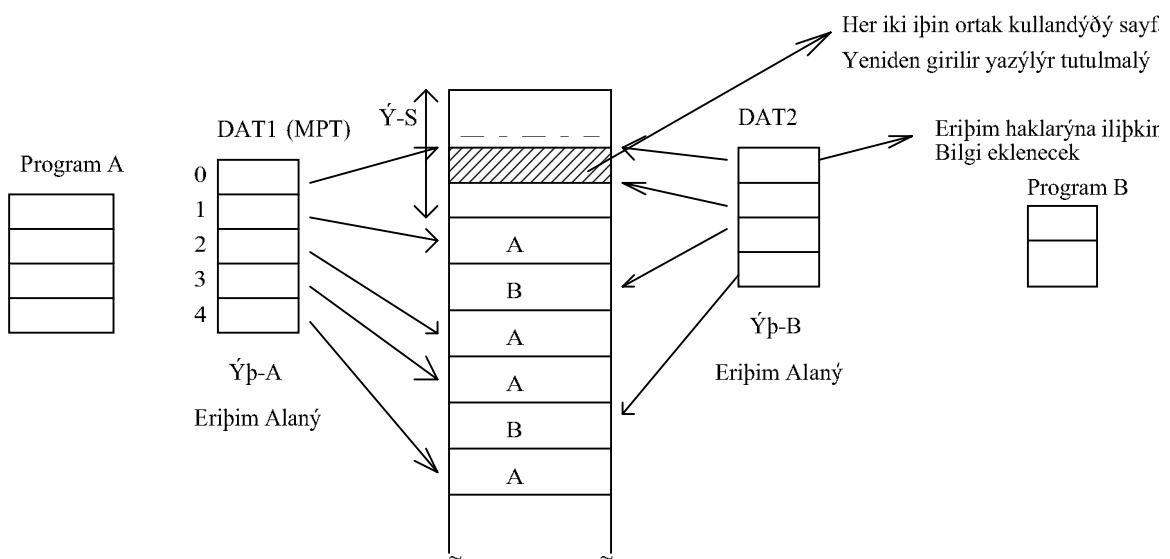


Adres Sahasının Kesişmesi

Şimdiye deðin tanımlanan dinamik bellek düzenleme yöntemlerinde işlerin adres sahalarının birbirinden bağımsız oldukları varsayılmıştır. Bu nedenle bunları karşılayan fiziksel konumlar da ana bellekte ayrı ayrı atanmıştır.

Her işin erişim alanının bir diðeriyle kesişmesi bu işler arasında doğal bir koruma mekanizması oluşturur. Ancak ana bellekteki bazı öğelerin (Program, veriler) ortak kullanımı arzulanan bir seçenektedir. Adres alanlarında çakışmanın işletim sistemi

ve işler arasında gerçekleştirilmesi çoğu kez yeterli bir esnekliktir. Adres alanlarını çakıştırmada kullanılabilecek bir yöntem de her istekti adres alanının belirli kesiminin (Örneğin ilk sayfa) ortak olarak tanımlara ayrılmışdır. (Şekil-48)



Şekil:48. Adres alanlarıının kesimmesi

Sistemde ortak kullanılan sayfaların korunması, paylaşımın verdiği genel sorundur. DAT (MPT)'larda yada özel taban registerlerinde kullanılan erişim haklarını belirleyen ve kısıtlayan göstergelere yer verilir.(oku/yaz,salt oku,işletim vb).

### **Avantajları:**

- 1) Ana belleğin parçalanmasını öner.
- 2) Belleği eşit parçalara böldüğü için diğer yöntemlere göre atama yöntemi yalındır.
- 3) Belleğin daha iyi kullanımını sağladığı için çok iş düzende başarı düzeyi artar.

### **Dezavantajları:**

- 1) Komut işletim süresini ve kullanılan ek donanım nedeniyle sistem fiyatını arttırmır.
- 2) İşletim sisteminde kullanılan veri yapılarının karmaşıklığı artar.
- 3) Program boyalarının sayfa kapasitesinin tam bir katsayı olmaması sebebiyle son sayfalarda "kırıntı" diye adlandırdığımız boş alanlar oluşur.
- 4) Ana belleğe yeni bir iş yüklenliğinde yeterli boş sayfaların bulunmaması bu işlemi engeller, böylece bazı sayfalar boş olmalarına karşın kullanılmadan dururlar.

### **10.2.2. Görüntü Ana Bellek Yönetimi**

Gerçek ana bellek yönetimi olarak tanımladığımız bellek düzenleme yöntemlerinde işler, gereksendikçe tüm bellek alanlary atanana deðin çakışmamaktadır. Bu yapı içinde çalışmak için bekleyen işlerin olmasına karþın ana belleğin bir kesiminin kullanılmaması olası bir durumdur. Bu olsunun yanı sıra her işe ayrılan erişim alanı, programlar ve kullanıldıkları veriler üzerine kapasite kısıtlamaları getirmektedir. Sözü edilen yöntemlerde bir işin adres alanı ana belleğin kapasitesi ile sınırlıdır. Ancak çok iş düzeni nedeniyle bu alan gerçekte sistem kapasitesinin çok altında tutulur. Günümüzde ana bellek teknolojisinin gelişmesiyle bu kaynakta ekonomik nedenlerle sınırlamalar giderek azalmaktadır. Yakın tarihlere kadar ana belleğin sistem değerinin önemli oranını tutması nedeniyle bu kaynak üzerinde yoğun olarak çaba harcanmış ve birçok düzenleme yöntemi geliştirilmiştir. Bunlardan ana belleği en ekonomik olarak kullananları "Görüntü Bellek Düzenleme Yöntemleri" adı altında toplanmaktadır.

Görüntü ana bellek düzeni bir işin adres alanına işe atanabilecek fiziksel bellek kesimini açma olanaðını sağlayan yöntem olarak tanımlanabilir. Başka bir deyiþle bir programın işletimi için tümünün ana bellekte bulunması gerekmektedir.

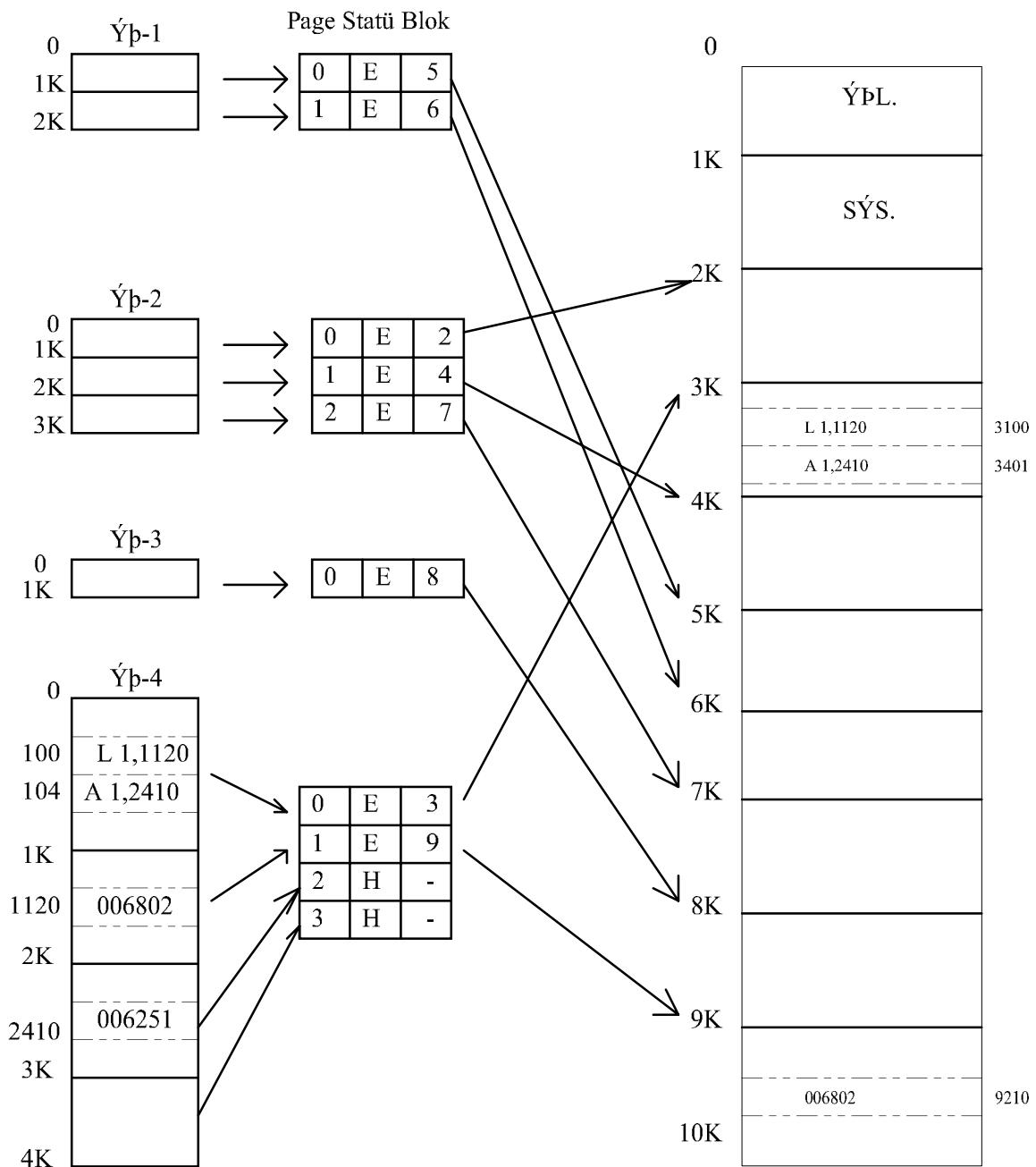
Zaman içinde bir programın çalışmasını izlediðimizde işletiminde gereksenen komut veri kümesinin belirli sınırları içinde kalması olasılıðının büyük olduğunu görürüz. Buna göre bir programın çalışabilmesi için bazı kesimlerinin ana bellekte bulunması yeterlidir. Ancak her an gereksinim bulunup da ana bellekte bulunmayan ögelere erişilebileceği sorunu çözme gerekmektedir.

Görüntü bellek düzenlemeye yaygın olarak uygulanan teknikler Demand-Paged, Segmented and Demand-Paged olarak sıralanabilir.

#### **10.2.2.1. Demand-Paged (İstenebilir-Sayfalı) Bellek Yönetimi**

Buraya kadar sözünü ettiðimiz bellek yönetimi teknikleriyle bir programın adresleme alanının tümü fiziksel alana yerleştirilmekçe o programın çalıştırılması söz konusu olamıyordu. Demand-Paged bellek yönetim tekniðinin temelini oluþtururan kavram ise bir programa ilişkin adresleme alanının tümüyle fiziksel alana yerleştirilmesi zorunluluðunun olmamasıdır. (Şekil-49).

Bu yöntemde programlar ve veriler sayfalama yönteminde olduğu gibi derleyici/çeviriciler aracılığı ile eşit uzunlukta sayfalara bölünürler. İşletimi istenen her işe gereksediði tüm bellek alanını karşılayacak kadar ikincil bellek atanır. Ögelere erişim sayfalama yönteminde olduğu gibi registerler yada dat aracılığı ile gerçekleştirilir. Ancak bu yapılarda erişilen sayfanın ana bellekte bulunup bulunmadığını belirleyen bir de belirteç yer alır. Adresleme sürecinde erişilmek istenen sayfa ana bellekte değilse bu gözterge aracılığı ile süreci durdurmak üzere bir kesilme üretilir. İşletim sistemi kesilmenin nedenini saptayarak gereksenen sayfanın ana belleğe taşınmasına çalışacaktır. Sayfa ana belleğe getirildiðinde işletimi kesilen komut yeniden başlatılır ve bu süreç iş sonuna deðin sürdürülür.



Şekil:49. Demond-Paged bellek yönetimi

Şekil-49 'da belirtildiği gibi İş\_4'ün adresleme alanı tümüyle fiziksel alana yansıtılmamıştır. Ancak İş\_4 sadece 0 ve 1. page'lerdeki bilgilere erişiyorsa çalışmasında herhangi bir aksaklık olmayacağı olacaktır.

Bu durumda iki sorun söz konusudur:

- 1) Bir program (iş) fiziksel alana yansıtılmamış bir page'deki bilgilere erişmek isterse ne olacaktır ?.

**2) Hangi page'lerin bellekte tutulması gerekiğinde nasıl karar verilecektir.**

(Şekil-49) da belirtildiği gibi İŞ\_4 'ün 104 adresindeki 1,2410 gibi bir komutla 2410 adresindeki bir bilgiye erişmek isteniyor ancak bu bilginin bulunduğu page bellekte değildir. Bu gibi durumlarda hangi bilgilerin bellekte olduğunu belirlemek için "page map table"ında " 1 bitlik bir statü bilgisi tutulmaktadır.

Statütü bit'i;

Hayır-0 (Page bellekte değil)  
Evet-1 (Page bellekte)

Bellekte bulunmayan bir page'e erişmek istendiğinde page mapping donanımı, statüsü (0) olan bu page için bir "page- interrupt"ı oluşturur. Bu interrupt'un servisini yapan işletim sistemi istenilen page'i belleğe yerleştirir ve page map table 'da gereken düzenlemeleri yapar.

İşte page'lerin gerekiğinde belleğe yüklenmesi nedeniyle bu tekniğe "Demand-Paged" bellek yönetim tekniği adı verilmektedir.

Sisteme girilmesi istenilen bir iş (Program) geldiğinde önce bu işin page'i yüklenir. Diğerleri ise istek(Demand) üzerine gerekiğinde belleğe getirilir. Böylece o anda kullanılmayacak olan page'lerin belleği işgal etmeleri engellenir.

Bir "Page Interrupt" oluştugunda istenilen page'in nereden yükleneceği sorusu akla gelmektedir. Aslında sisteme girilen bir işin adresleme alanı tümüyle ikincil-bellek adını verdigimiz manyetik disk veya durum gibi saklama organlarında tutulmaktadır. Söz konusu iş için bir page istenildiğinde ikincil bellekten alınıp ana belleğe yerleştirilmektedir.

Demand-Paged teknigi, bellek yönetimi açısından etkin bir teknik olmasına karşın bazı sorunları da vardır. Örneğin (Şekil-49) incelenecək olursa; A 1,2410 komutunu işleyebilmek için iş-4 'ün 2.page'ini yüklemek gereklidir. Ancak bellekteki bütün bloklar kullanıldığından yeni gelen page nereye yerleştirilecektir?. Ana bellekteki bütün bloklar kullanılıyorsa, yeni bir page'i belleğe yerleştirmek için bellekteki bloklardan birisinin boşaltılması gereklidir. Bunun için boşaltılacak bloktaki bilgilerin ikincil belleğe kopya edilmesi gerekmektedir. Bu işleme page swapping, page removal, page replacement veya page turning adı verilmektedir.

Bir diğer soru ise, hangi bloğun boşaltılabilcegi hakkında nasıl karar verilmelidir? Bu konu ile ilgili birçok çalışma yapılmış ve çeşitli page-replacement algoritmaları geliştirilmiştir. Bunlardan en önemli olanlarından ileride söz edilecektir.

Bu teknikte diğer bir sorun da, ana bellekten ikincil belleğe kopya edilen page'in tekrar belleğe getirilmesi olayıdır. Bu durum genellikle çok küçük ana belleği bulunan sistemlerde görülen ve Thrashing adı verilen bir olaydır. Thrashing konusu da

günümüze kadar gelen bir araştırma konusudur ve bununla ilgili halen devam etmekte olan bazı çalışmalar bulunmaktadır. (Denning,Ramdell,Coffman).

Demand-Paging bellek yönetim teknigini uygulayan sistemlere örnek olarak VS-1, VS-2, VM-370, Honeywell 6180'de Multics ve Univac 70/46'da VMOS işletim sistemleri verilebilir.

### **Donanım Desteği.**

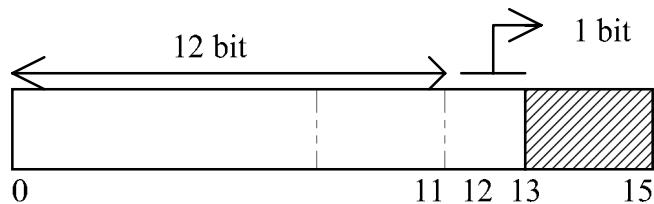
Sayfala yöntemine ek olarak, adres çözümleme sürecinde donanımın, erişimi istenen sayfanın bellekte bulunup bulunmadığının denetlemesi gereklidir. Donanımda adresleme sürecini durdurmak üzere ek bir kesilmenin öngörülmesi gereklidir.

İşletim sistemi birçok durumda öğelerin gerçek adresleriyle çalışmak zorundadır. Örneğin G/Ç kanallarına iletilecek veri alanı adreslerinin, ana bellekteki fiziksel bir konumu doğrudan göstermesi istenir. Bu gibi sorunları karşılamak üzere donanımda bir ögenin gerçek adresini saptayan ek bir komut (load real address) ve daha özel uygulamalar için de (örneğin yükleyici yordamında) sistemin gerçek adreslerle çalışabilecek ek olanaklar gereklidir.

Donanım açısından "demand-paged" bellek yönetimi "paged" bellek yönetiminde sözünü ettigimiz donanıma (PMT,PMTAR,PMR) ek olarak 3 donanım elemanı daha gerektirir. Bunlar ;

- 1)** Page'lerin ana bellekte veya ikincil bellekte bulunduğu göstermek amacıyla kullanılan ve "Page Map Table" in içinde tutulan 1 bitlik statü bilgisi,
- 2)** Bellekte olmayan bir bilgi istenildiğinde "interrupt" veri kontrolü işletim sisteme aktaracak bir mekanizma,
- 3)** Her page'in kullanımıyla ilgili bilgileri tutup "Page-removal" işleminde işletim sisteminin hangi page'in (bloğun) boşaltılması gerektiği üzerine karar vermesinde yardımcı olacak bir mekanizmadır.

Bu donanım elemanlarının nasıl sağlandığını açıklamak için IBM 370 sisteminin "Exteded Control Mode" unda yani DAT'in çalıştırıldığı ortam incelenebilir. DAT mekanizmasının çalışmasını anlatırken belirttiğimiz gibi blok numaraları page map table içinde 2 baytlık yani 16 bitlik bir sahayı kaplayacak bir şekilde tutulmaktadır. Ancak bu 16 bitlik bu sahanın sadece 12 bitlik bir kısmı en fazla 4096 bloğu belirtmek için yeterli olduğundan 12.bit pozisyonu page statüsünü belirlemek için kullanılmaktadır. (Şekil-50).



Şekil:50. Blok numarasý

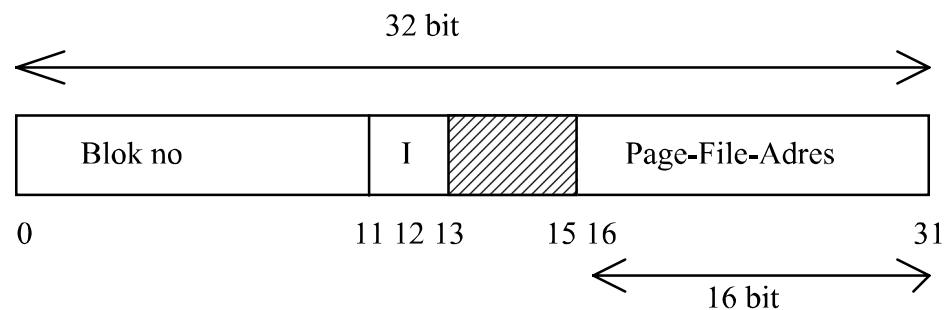
Eğer söz konusu page bellekte değilse  $I=1$  dir ve bu page 2 ye erişilmek istenirse işletim sistemi "page interrupt"ının olmasını sağlar. Page exception veya page fault adı verilen bu interrupt oluştugunda (Interrupt code=17) programın (İşin) erişmek istediği yerin adresi ana bellekte 144 adresinde saklanır. İşletim sistemi 144 adresindeki bilgiyi kullanarak gereken page'i belleğe yükler.

Diğer yandan 370 sistemlerde her blok için 2 bitlik bilgi daha tutulmaktadır. Bunlardan biri reference bit olup, o bloktaki bilgilere erişilmek istenildiğinde (LOAD,STORE gibi) 1 değerini almaktadır. Diğer ise change bit olup, o bloktaki herhangi bir bilginin üzerinde değişiklik yapıldığında (STORE gibi) 1 de/erini almaktadır. Reference bit ve change bit bazı sistemlerde (Honeywell 6180'de olduğu gibi) Page Map Table içinde tutulmaktadır.

IBM 370 sistemlerde ise bu 2 bitlik bilgi belleğin her 2 K'sı için kullanılan özel amaçlı LOCK registerlerinde tutulmaktadır. İşletim sistemi set storage key privileged komutunu kullanarak bu bitlerin değerlerini değiştirebilmektedir.

### **Yazılım Desteği.**

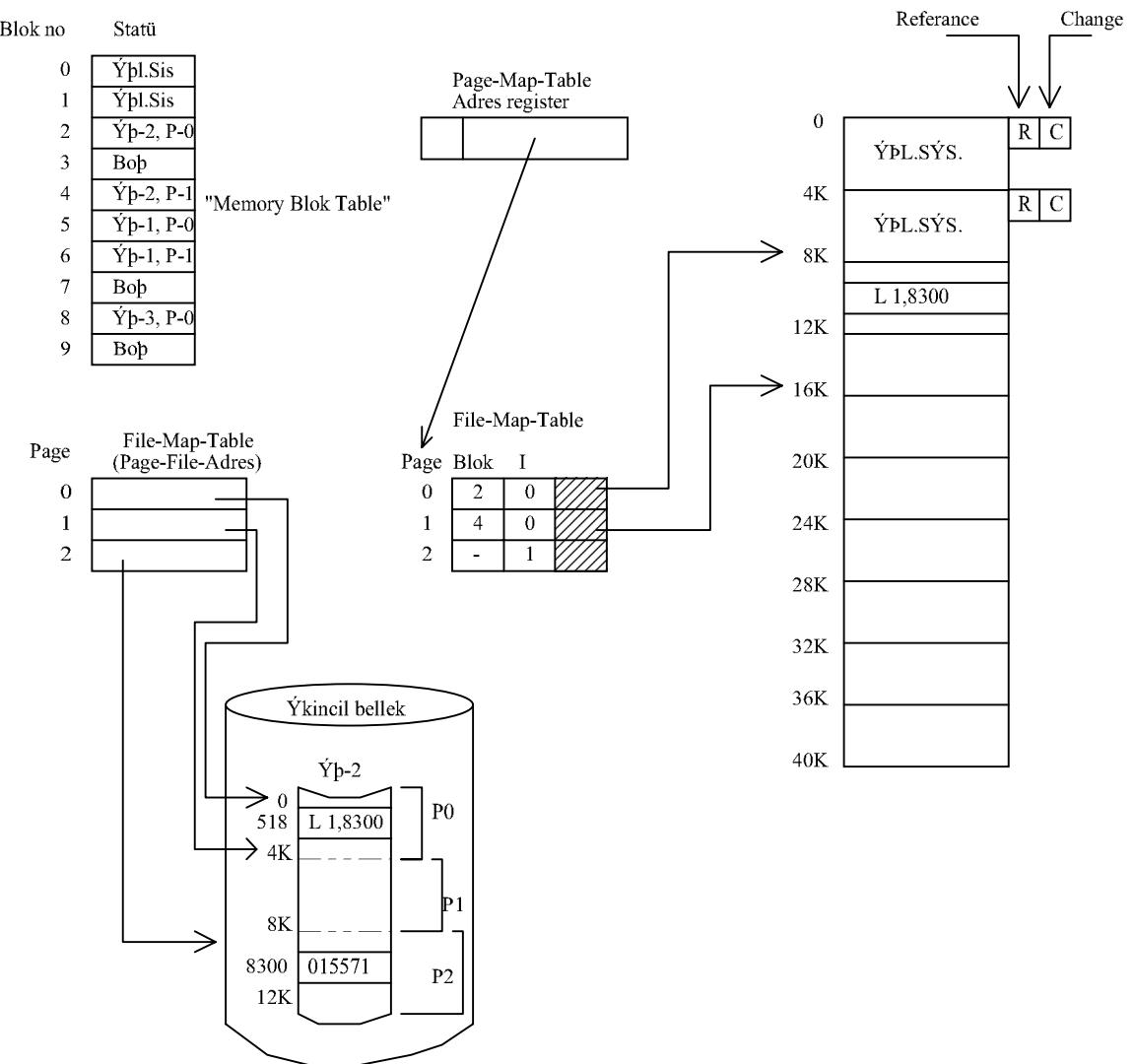
Demand Paged Bellek Yönetiminde adresleme alanının (page'lerin) ana bellek ile ikincil bellek arasında transfer edilmeleri söz konusu olduğundan pagelerin ikincil bellekte nerede bulunduklarını belirtecek bilgilerin tutulması gerekmektedir. Aslında bu konu işletim sistemlerinin "bilgi yönetimi" ile ilgili bölümüne girmektedir, şimdilik her page'in ikincil bellekte bir adresi olduğu varsayılabılır ve bu adrese page file adresi adını verebiliriz. Mantıksal olarak page file adresi de page map table içine alınabilir. (Şekil-51).



Şekil:51. Blok-No, Page-adres ilişkisi

Page Map table'da bulunan bilgiler direkt olarak donanımla değildir (DAT), page-file-adres ise yazılım yoluyla işlenebilir bilgilerdir. Bu nedenle page-file-adres bilgileri genellikle file-map-table (FMT) adı verilen ayrı bir yerde tutulmaktadır.

File-map-table, page-map-table ve memory-blok-table arasındaki ilişkiler (Şekil-52)' de gösterilmiştir.



Şekil:52. File-Map-Table, Page-map-table, Memory blok table ilişkisi.

## Adresleme Sürecinin İncelenmesi

Bu yöntemde adresleme süreci donanım ve yazılımın tümlediği bir yapıyı kullanır. Erişilmek istenen sayfanın bellekte bulunmadığı durumlarda adresleme sürecinin kesilmesi belirsiz bir süre sonunda çözümlenecek bir aramaya girilmesinden kaynaklanır. Bu aramayı adım adım incelediğimizde çözümleyebileceğimiz sorunları şöyle sıralayabiliriz;

- Bellekte atanacak boş bir sayfa bulma,
- Boş yer yoksa, yer açmak üzere bir seçim yapmak,
- Ana bellekten atılacak sayfa ikincil bellekteki kopyasından farklı ise, yani günlenmiş ise (içindeki veriler değiştirilmiş ise) bu sayfayı ikincil bellekteki yerine yazmak,

- d) Erişilmek istenen sayfayı ana belleğe yüklemek,
- e) İşletimi kesilen komutu yeniden başlatmak.

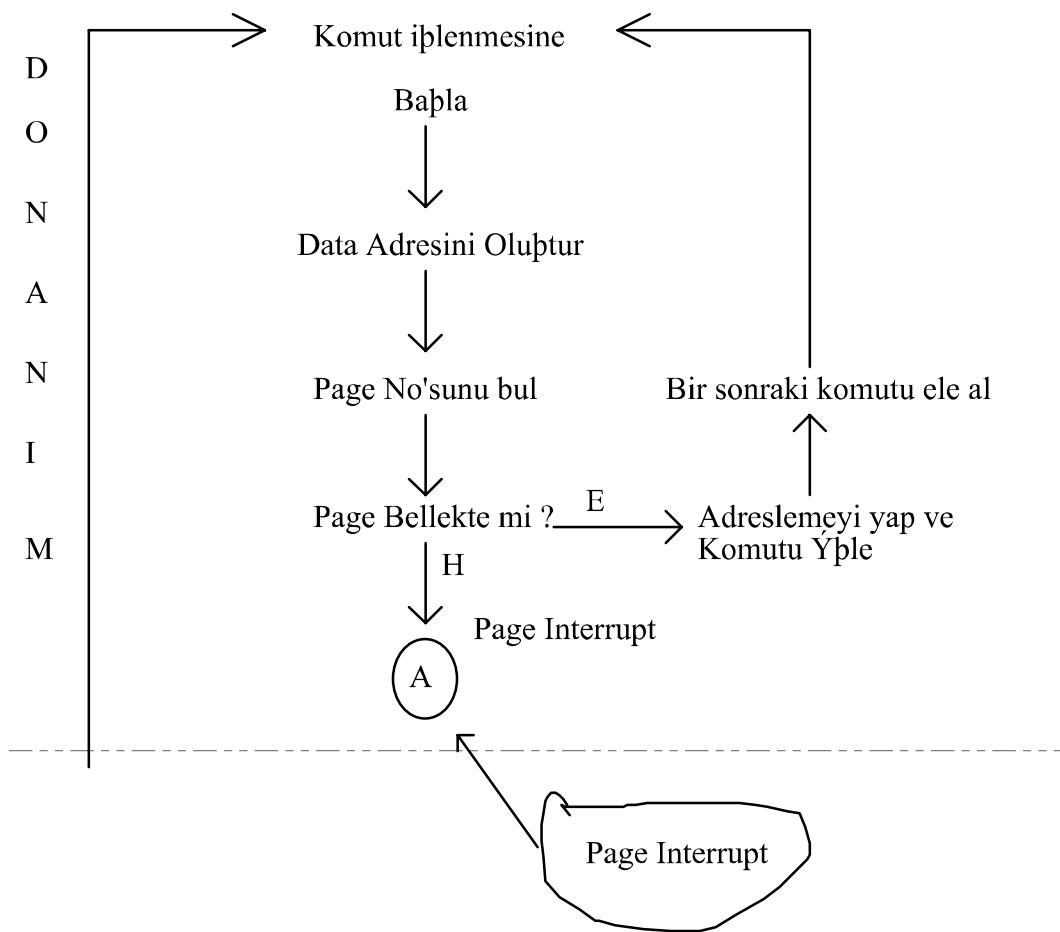
Bu işlemleri gerçekleştirdirken ek bellek sayfalarına gereksinim duyabileceğimiz düşünülsürs, yöntemin yalnız görünümüne karşın işletim esnasında karmaşık yapılar oluşturabileceği gözlenebilir.

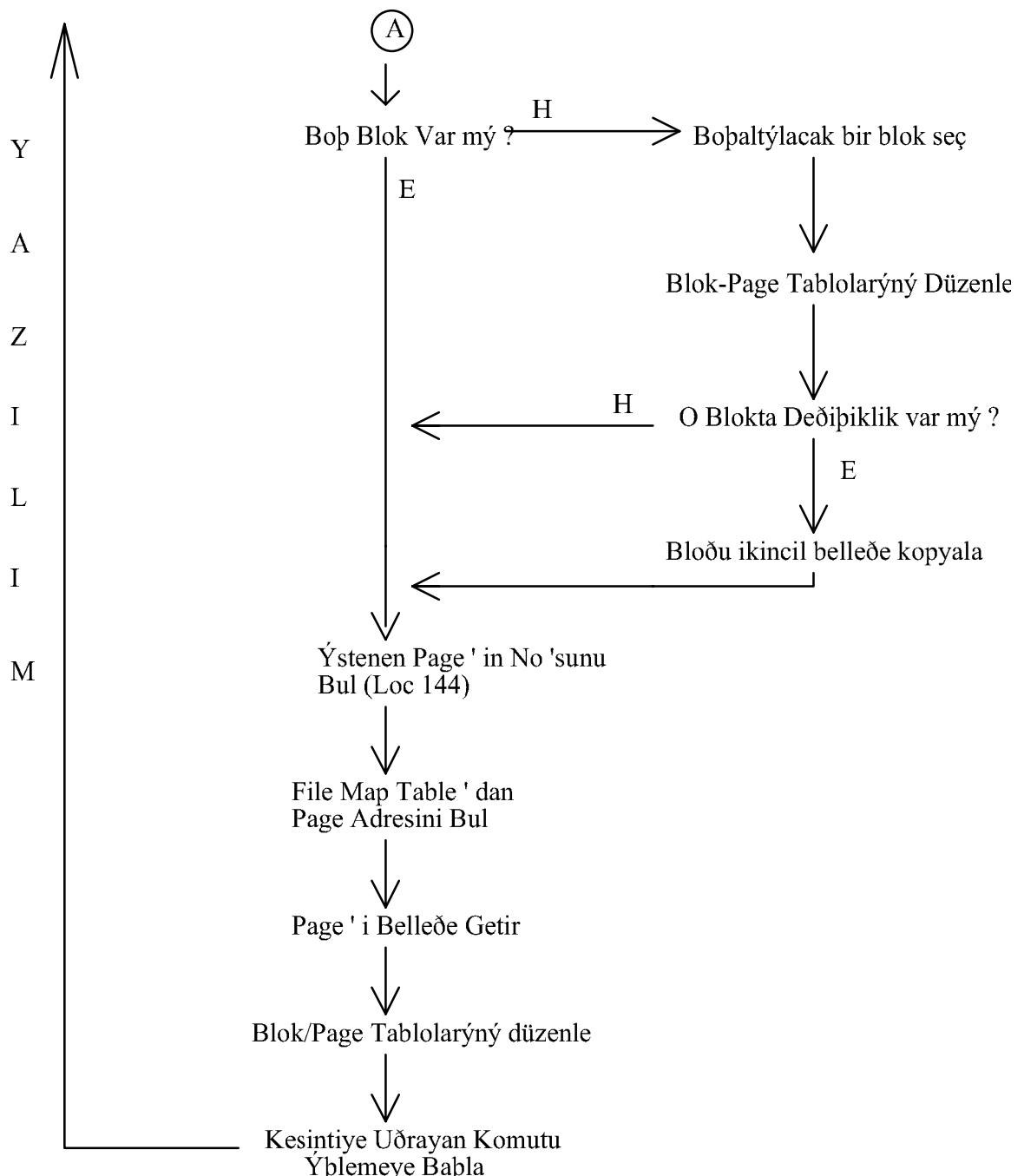
Yukarıda saydığımız adımların gerçekleştirmek için kullanılan veri yapılarının da ek bilgileri içermesi gereklidir. Örneğin hangi sayfanın bellekten çıkarılacağına karar vermek için sayfanın kullanım durumunu yansıtan bir göstergeye gereksinim vardır.

Kullanılan algoritmalarla göre (FIFO, LRU,...) bu göstergeler karmaşık bir yapıya doğru gidebilirler. Ana bellekteki sayfaların kullanmadada yada serbest olduğunu gösteren belirteçlerin yanı sıra, sayfanın durumunu saptayan ek belirteçlere gereksinim duyulur.

Ana bellekle ikincil bellek arasında aktarılmakta olan bir sayfayı yeniden seçmemek için "aktarılmakta" olduğunu tutmak gereklidir. Ana bellekte kullanılan bazı sayfalar da boşaltılamaz durumdadır. Özellikle G/C'ların yapıldığı alanların yerini değiştirmek çok sakıncalı durumlar yaratır, bunların da "saltık(transit)" konum olarak gösterilmesi gereklidir. Ana belleği boşaltmak üzere seçilen bir sayfanın her zaman ikincil belleğe yazılması gereksiz bir yük olabilir. Bu sayfanın ikincil bellekteki kopyasıyla uyumlu olması bu taşıma işlemini gereksiz kılar. Durumu saptamak üzere erişim, donanımca günlenen "değiştirilmiş sayfa" göstergesi kullanılır.

Demand-Paged bellek yönetiminin en önemli özelliklerinden birisi bellek atama işleminin dinamik olarak yani "execution" sırasında yapılmasıdır. Bu nedenle bu bellek yönetiminin uygulandığı sistemlerde bellek yönetim donanımı (DAT gibi) ile bellek yönetimi yazılımı arasında çok sıkı bir bağ bulunmaktadır. Bellek yönetim yazılımını page interrupt'ları açısından ele alacak olursak, bu bağın nasıl olduğu (Şekil 53)'de gösterilmiştir.





Þekil: 53. Demond Paged Bellek Yönetiminde Yazýlmý-Donaným Ýlipkisi

### **Yöntemin Tartışılması.**

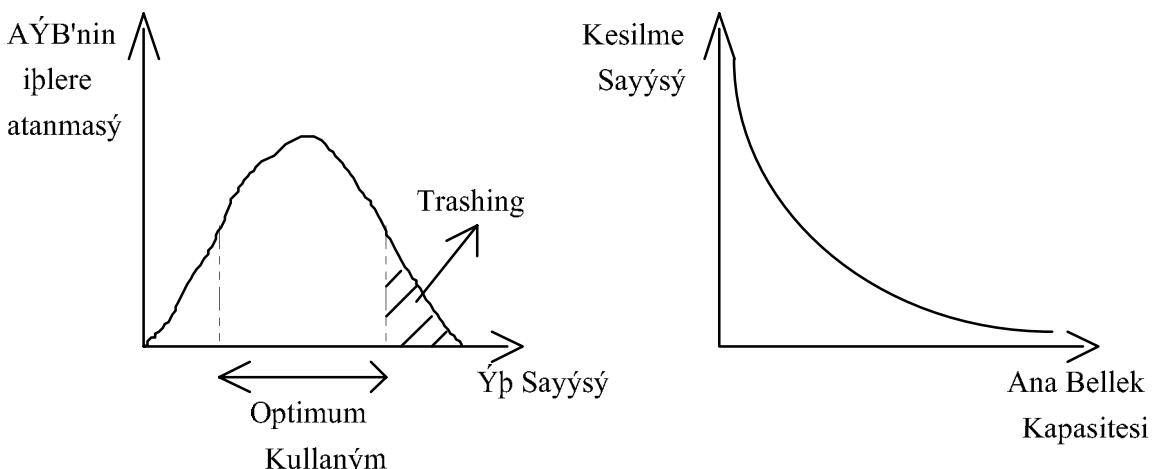
Demand-Paged yönetimi ana bellekte işletimin her anında yalnızca kullanılan program sayfasını ve gerekirse bir veri sayfasının bulunmasını yeterli gördüğünden, ana belleğin yönetimini çok etkinleştirilen bir yaklaşım olarak yaygın bir biçimde uygulanmaktadır. Ancak her yöntemde olduğu gibi getirdiği katkılar bir takım yan sakıncalarla dengelenmektedir.

## Avantajlar

- 1) Görüntü bellek uygulaması nedeniyle işlere sınırsız ana bellek olanağı sağlamaşı,
- 2) Çok iş düzeninde iş sayısını sınırsız olarak artttırmak,
- 3) Sistem kaynakk kullanımını artttırmak.

## Dezavantajlar

- 1) Sistemde izlenip tutulacak veri sayısını arttırr,
- 2) Bulunmayan sayfa kesilmesini işleyecek ek yazılım gerektirir,
- 3) Komut işletim süresi artar,
- 4) Adresleme sürecinin yazılım kesiminde de "bulunmayan sayfa" kesilmeleri oluşabilecegi gözönünde tutulursa, komut işletim süresi belirsizleşir yada bir sayfayı boşaltmak için ek sayfa gerektiğinde sistemde kilitlenmeler oluşabilir.
- 5) Boşaltılacak sayfayı seçecek yordam çok karmaşık bir algoritma olabilmesine karşın başarısı sınırlı olabilir.
- 6) Ana bellekte yer bulunmadığında, eksik sayfa uyarlarının artması sistemin başarısını düşürmeye başlar. Bu durumun son aşaması, yeni bir sayfayı yüklemek için yükleme istemini yapan program kesimini ana bellekten atmaktadır.
- 7) İşletim sisteminin eksik sayfa kesilmeleriyle yoğun bir şekilde uğraşmaya başlamasına "trashing" adı verilir. Trashing'de çalışan bir sistemde ana bellekte yeterli yer açıp tikanan işleri sonuçlandırmak bellek yöneticisinin en karmaşık görevidir. ( Şekil-54 )



Şekil : 54. Trashing

## Sayfa Yükleme Politikası

- 1) Sayfaları kullanımından önce yükleme ( Prepaging ).

- 2)** İstem üzerine yükleme.
- 3- a)** Programların işletimdeki davranışlarını öngörme zorluğu.
- b)** Yanlış sayfayı yüklemenin getirdiği ek kayıp.

## Sayfa Atama Yöntemleri

Gerekli sayfaların bellekten atılmaması gereklidir.

- 1)** Rasgele
- 2)** FIFO ( First in first out )
  - Bellekte en çok kalmış sayfayı çıkarma
  - Kalıntılar temizlenir
  - En çok kalan gereksiz.
- 3)** LRU ( Last Recently Unused = En uzun süredir kullanılmayan )
  - En uygun yöntem
  - Donanım da devreye girerek pahalı çözüme yol açması.
- 4)** Önceden tanımlanmış öncelikler
  - Öncelikli işlerin diğerlerini tıkamaması
- 5)** Optimal
  - En uzun zaman kullanılmayacak olan
- 6)** Karışık yöntemler

## Working Set

Working set, bir programın  $\Delta t$  zaman aralığında çalışmak için gereksediği sayfa sayısızır (Herhangi bir görev için kabul edilir bir zaman aralığı içinde bulunmayan ya da eksik sayfa uyarısı üremeyen sayfa sayısıdır). Zaman aralığını programın işletim süresi olarak tanımladığımızda "working set" program boyuna doğru yaklaşır, tam eşitlik programda kullanılmayan kesimler nedeniyle sağlanmaz.

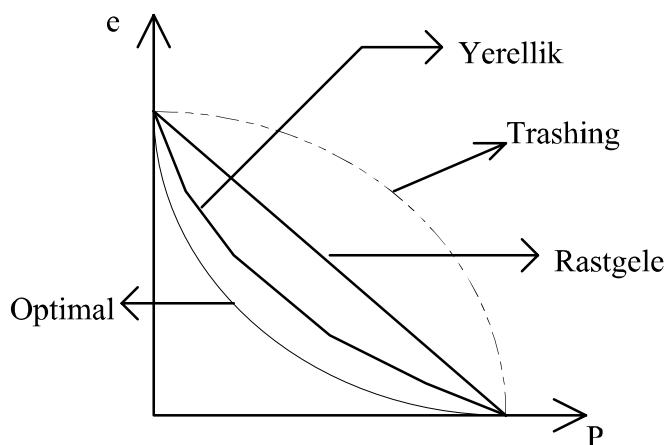
İstem üzerine sayfalama yapan sistemlerde ana bellek yöneticisinin en uygun yöntemi uygulamak üzere dayanabileceğinin en sağlıklı parametre, programların working set'leridir. Ancak programların dinamik davranışları işletimden önce kesinlikle kestirilemez ve bu davranış değişik uygulamalarda değişebilir.

İşletim sistemlerinde başarı ölçüyü yapanlar, genelde programların davranışlarında belirgin bir yerellik olduğunu gözlemişlerdir. Bu yerel davranış iki sınıfta toplanabilir;

- 1)** Zaman içinde yerellik: Son erişilen bir verinin yakın bir zamanda tekrar erişilme özelliğidir. Programdaki döngüler ve sık kullanılan değişkenler alt yordamlardan kaynaklanır.
- 2)** Yerleşimde Yerellik : Programın bir konumundaki bir verinin yakınında bulunan diğerlerinin kısa bir süre sonra erişilme özelliğidir. Komutların ardışık işletimi, diziler, zincirler gibi doğrusal veri yapılarının kullanımı, ilk kullanılan değişkenlerin program içinde bir araya konmasından kaynaklanır.

İştem üzerine sayfalamanın uygulandığı bir sistemde işletilen programlar ne kadar yerel davranışırlarsa sistemin başarı şansını o derece arttırlar. Başka bir deyişle bir görevin working set'i kabul edilir bir zaman aralığı içinde bellekte bulunmayan sayfa uyarısını üretmeye kesimidir.

$e$  değişkeninin bir program içinde bellekte bulunmayan sayfa uyarısına yol açan komut yüzdesi olduğunu varsayıyalım,  $p$  de ana bellekte bulunan sayfaların yüzdesini göstersin. Bir işletimde bellek ulaşımının dağılımının rastgele olduğu varsayılırsa  $e=1-p$  dir. Başka bir deyişle herhangi bir bellek erişimini herhangi bir sayafaya yönlendirme olasılıktadır.



Ancak yerellik kavramı gerçek eğrinin ilkinin altında olması gerektiğini göstermektedir. Ancak bu çizim ana bellekte working set'i oluşturan "en iyi p" sayfanın bulunduğu varsayılmaktadır. Ayrıca en sık adreslenen sayfaların da hemen hemen her zaman bellekte tutulduğu varsayılmıştır. Bu varsayımlar sayfa atama yönetemeleri ile doğrudan ilişkilidir. Optimal atama algoritması bu varsayımları karşılarken LRU tam sağlanamaz.

### Sayfalama Yöntemlerinde Davranış Anormallikleri

Trashing , işletimde "eksik sayfa" kesilmelerinin yoğunlaşması ve işletimi aksatacak düzeye çıkmasıdır. Sorun, her işe ayrılan fizikal bellek kapasitesinin working set'i yeterince içermediginden kaynaklanır.

### Bleedy Anormalliği

Görüntü bellek düzenlemede sayfalama yönteminin en iyi uygulanması için içgüdüsel ve sezgisel yöntemlerin kullanımı çoğu kez beklenen sonucu vermez. Örneğin bir işe ayrılan bellek alanı büyükçe bu işin ürettiği eksik sayfa uyarısını her zaman

azalabileceği ileri sürülebilir. 4 sayfadan oluşan bir programı ele alalım ve sayfa isteme sırası,

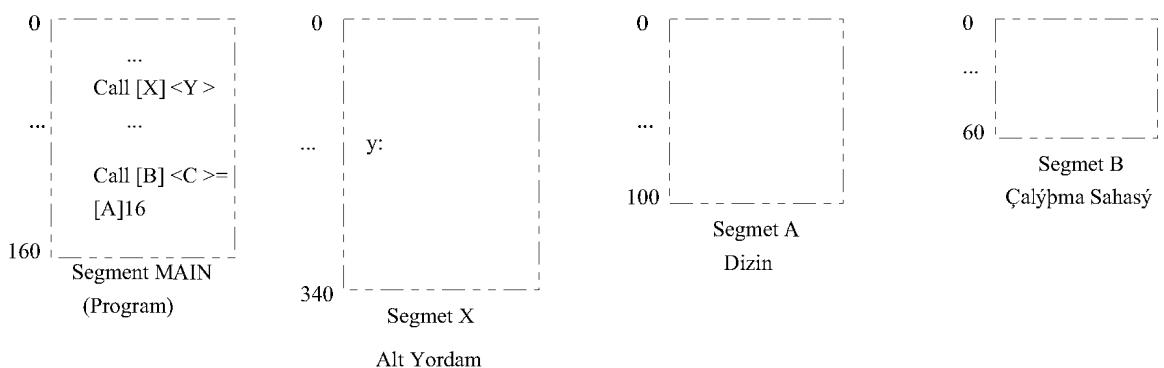
1-2-3-4-1-2-5-1-2-3-4-5

olsun

Bu program FIFO atama yöntemiyle kendisine 4 sayfa ayrıldığında 10 kesilme, 3 sayfa ayrıldığında 3 kesilme üretmektedir.

### 10.2.2.2. Segmented (Kesimleme) Bellek Yönetimi

Kesimleme, programları mantıksal bütünlük içeren birimlere ayırarak ana belleğe yükleyen yaklaşımı verilen addır. Alt yordamlar, yapısal programlama dillerindeki blok yapıları, işlevlerden herbiri mantıksal bir birim olarak varsayılabılır. Hepsinin ortak özelliği kendi içinde işlevsel bir bütün oluşturmasıdır. (Şekil-55).



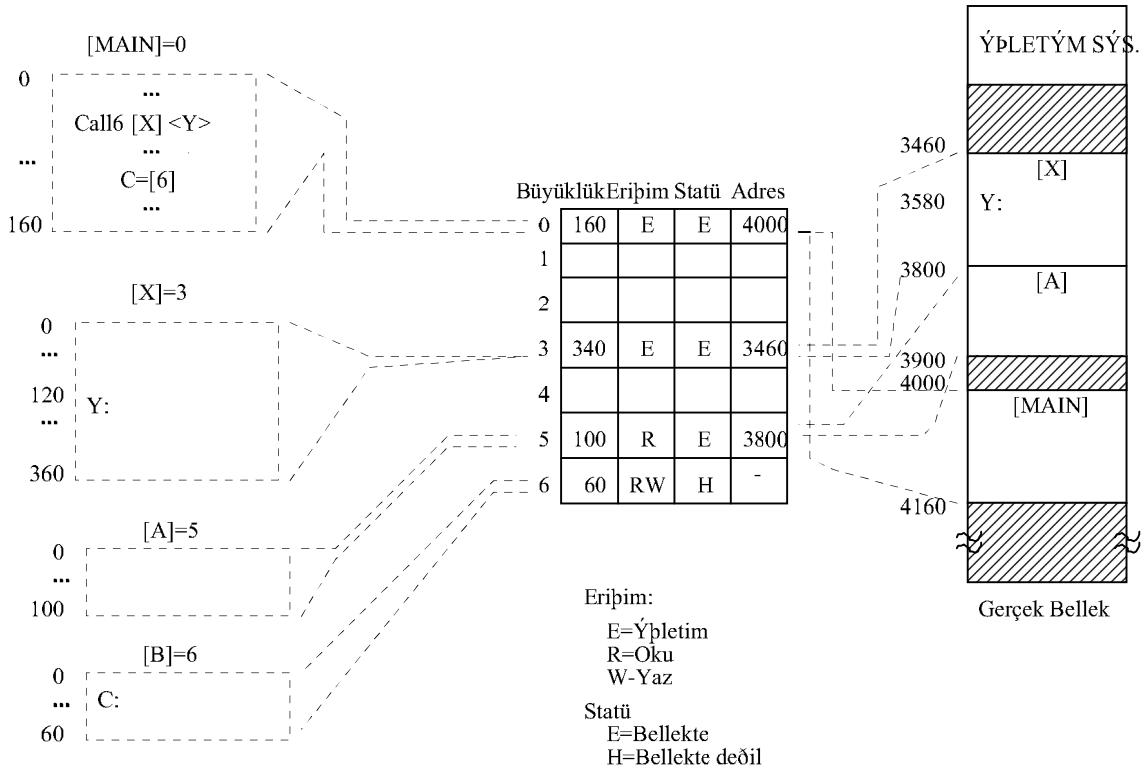
Şekil:55. Kesilme Adres Sahası

İstem üzerinde sayfalama yönteminde sözkonusu ettiğimiz yerel davranışın, mantıksal açıdan bir bütün olan kesimde çok daha yüksek düzeyde gözlenmesi doğaldır.

Ceviriciler ve derleyiciler kesimleme uygulanan sistemlerde program içindeki öğelerin adreslerini, kesim numarası ve kesim başına göreli konum olarak çözümlerler. İşletim sayfalama sisteminde olduğu gibi, gerçek adreslerin hesaplanması için program kesimlerinin bellek içindeki (ana, ikinci bellek) konumlarını belleyen dinamik adres tablosuna gereksinim vardır. Bu benzerlikten dolayı kesimleme ve istem üzerine sayfalama yöntemleri sık sık karıştırılır. İstem üzerine sayfalamada programların eşit uzunlukta fiziksel bölgelere, kesimlemede ise değişir uzunlukta mantıksal bölgelere ayrıldığını unutmamak gereklidir.

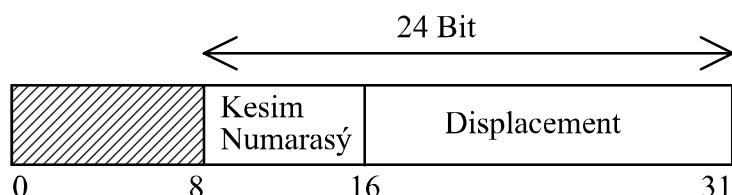
## Donanım Desteği.

Kesimleme yönteminde adresleme sürecinin görelî kesim tablosundan başlayarak gerçek adresi saptaması ve eriştiği kesimin ana bellekte bulunduğuun sınanması gereklidir. Kesim boyları değişir uzunlukta olduğu için erişilen ögenin kesim içinde kaldığının denetlenmesi gereklidir (Şekil-56)



Şekil: 56. Kesimleme Bellek Yönetimi

Program yüklenirken işletim sistemi onu segment adı verilen kesimlere ayırır. İşletim sistemi gerçek belleği eniyi şekilde düzenlemek amacıyla programları kesimlere ayırır. IBM-370 sistemlerinde kesim lokasyon adresi 24 bit uzunluğunda olup iki bölüme ayrılmıştır (Şekil-57)

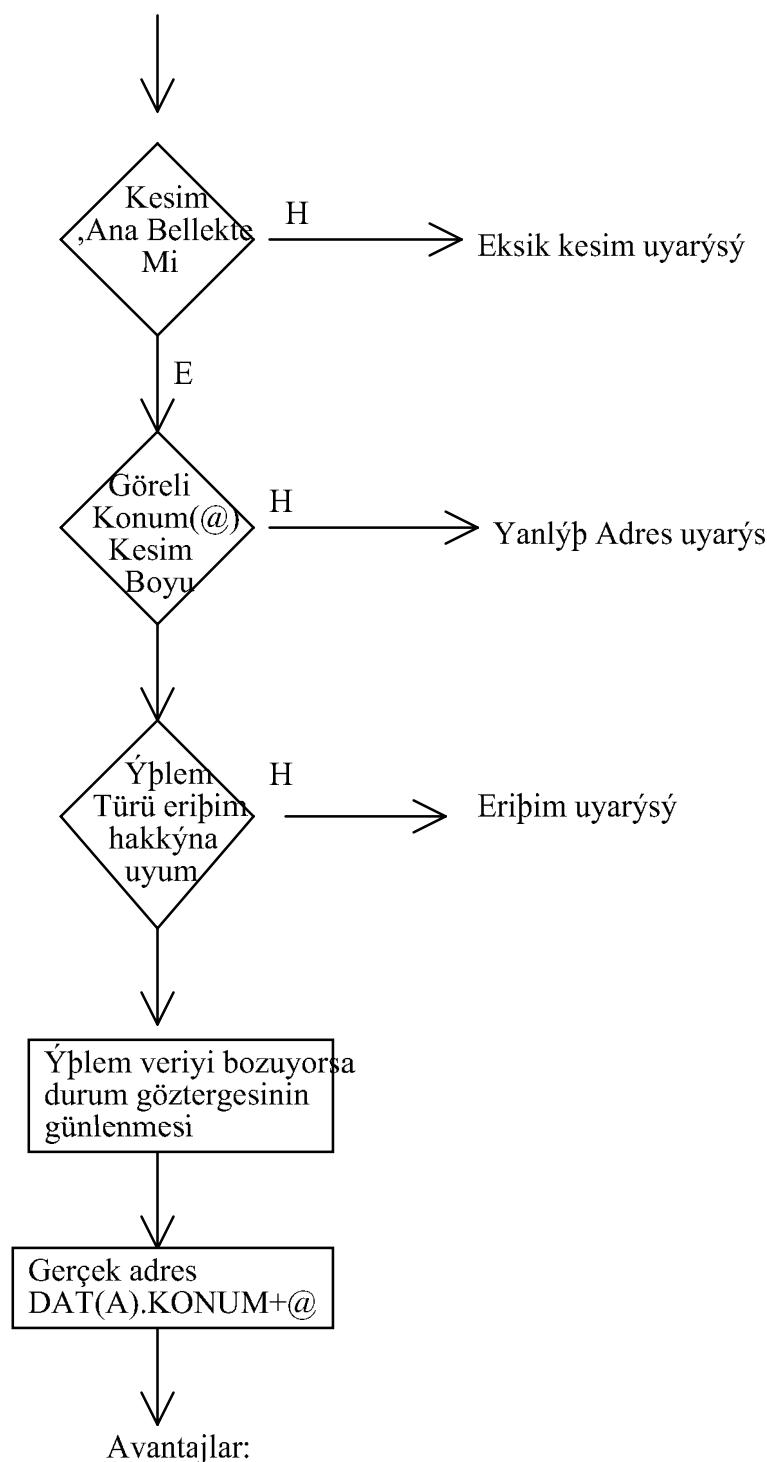


Şekil : 57. Kesim Efektif Adresi

(Şekil-57) 'den anlaşıldığı gibi 8 bit ( $2^8$ )=256 olduğundan programlar 256 kesime bölünebilir ve her kesim 64K (65536) genişliğinde olabilir.

### **Yazılım Desteği**

Adresleme sürecinin işlenmesi:Erişilen adres (A,@)



- 1) Görüntü bellek kavramının getirdiği tüm olanaklar.
- 2) İşletim sırasında değişimle uzunlukta kesim kullanma olanağının sağlanması
- 3) Program ve yordam bağlama (adlandırma=program içi görelî konum) sürecinin işletme kaydırılabilmesi (sadece işletme giren kesimler bağlanıyor).
- 4) Kesimleri paylaşabilme olanağı (erişim hakları tanımlayarak). Örneğin iki ayrı program COS fonksiyonu kullanırsa bu fonksiyonu içeren kesim bellekte yaratılır ve bu kesim iki program tarafından paylaşımı olarak kullanılır.

#### **Dezavantajları:**

- 1) Dinamik bellek atama yöntemlerinin tümünde olduğu gibi gerek yazılım gerekse donanım pahasının artması.
- 2) Kesimlerin değişik boyda olması nedeniyle ana bellek yönetiminin güçlenmesi.
- 3) En büyük kesim boyunun ana belleğin fiziksel kapasitesiyle sınırlanması (sayfalama ile kesimlemenin farkı).
- 4) İstem üzerine sayfalama olduğu gibi thrashing'i önleyici yaklaşımın getirdiği yük.

#### **10.2.2.3. Segmented And Demand-Paged(Kesimleme Ve İstenebilir Sayfalı Bellek Yönetimi**

Kesimlemenin sağladığı esneklik ve olanakları koruyarak sakıncalarını bir ölçüde kaldırmanın bir yolu sayfalama yöntemini kesimler düzeyinde uygulamaktır.

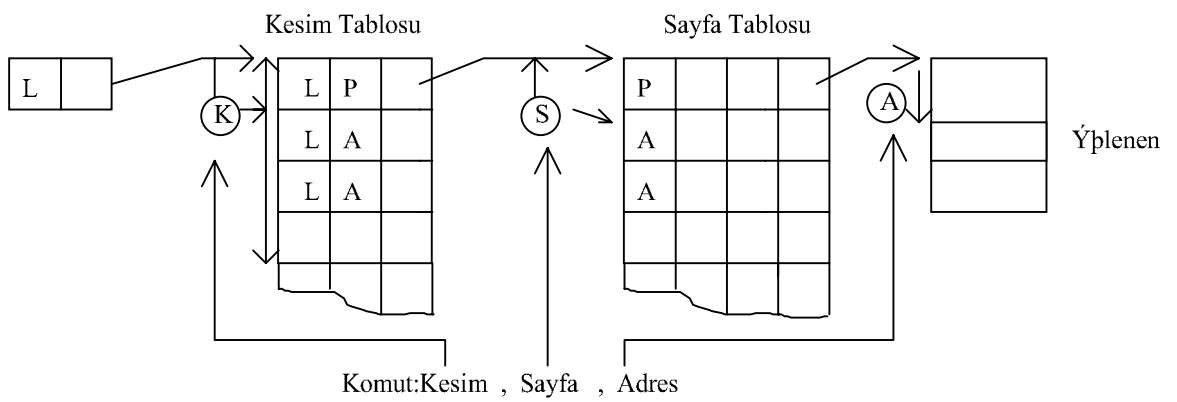
Paylaşım,dinamik bağlama,işletim aşamasında boy değiştirme gibi kesimlemenin verdiği olanaklar, her bir kesimin eşit uzunluktaki fiziksel birimlere ayrılmasıyla geçerliliklerini korurlar. Ve dolayısıyla ana bellek yönetiminde tüm bir kesim yerine uzunluğu belli sayfalar kullanılacaktır.

#### **Donanım Desteği.**

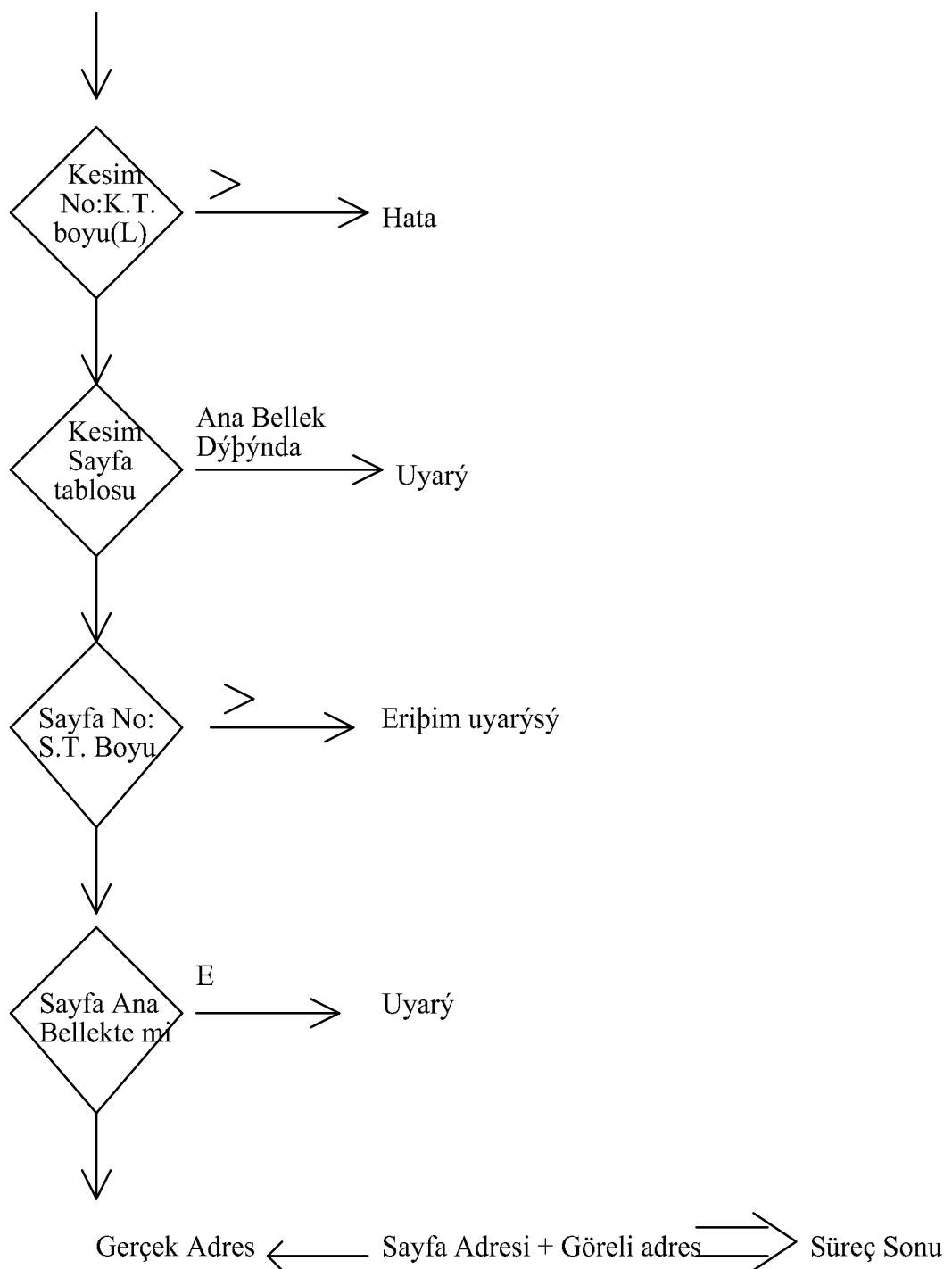
Komutlardaki adresler;

Kesim no + sayfa no + sayfa içi konum

biçimine dönüştüğü için her programa bir kesimler tablosu,her kesime de bir sayfa tablosu gereklidir. (Şekil-58)



Şekil: 58. Kesimleme ve Yüstenebilir-Sayfalama Bellek yönetimi



### **Avantajları:**

Yöntem şimdide deðin sayılan tüm avantajları içerir.

### **Dezavantajları:**

- 1) Donanımın pahası
- 2) Yazılım ve tutulacak veri yapılarının karmaþıklığı.

Kesimleme ve istenebilir-sayfalı bellek yönetimi, büyük sistemlerde başarıyla uygulanan ileri düzeyli bir yöntemdir.

#### **10.2.3. Diğer Bellek Yönetim Teknikleri.**

**1) Swapping:** Ana bellekteki program kesimlerinin ikincil belleğe taşınması. tek ve bitişken yeri değiştir bellek yönetimlerinde tüm bellek için söz konusu işe veya kullanıcıya ilişkin tüm kesimi çıkarılıyor.

**2) Overlay:** Bazı kesimlerin ana bellek ile ikincil bellek arasında gidip gelmesi.

**3) Cash Memory:** Ana belleğin önüne konmuş bellek.

**4) Spool:** İkincil belleklerde giriş buffer'larından okuma yapıp yine ikincil bellek bufferlarında oluşturma.