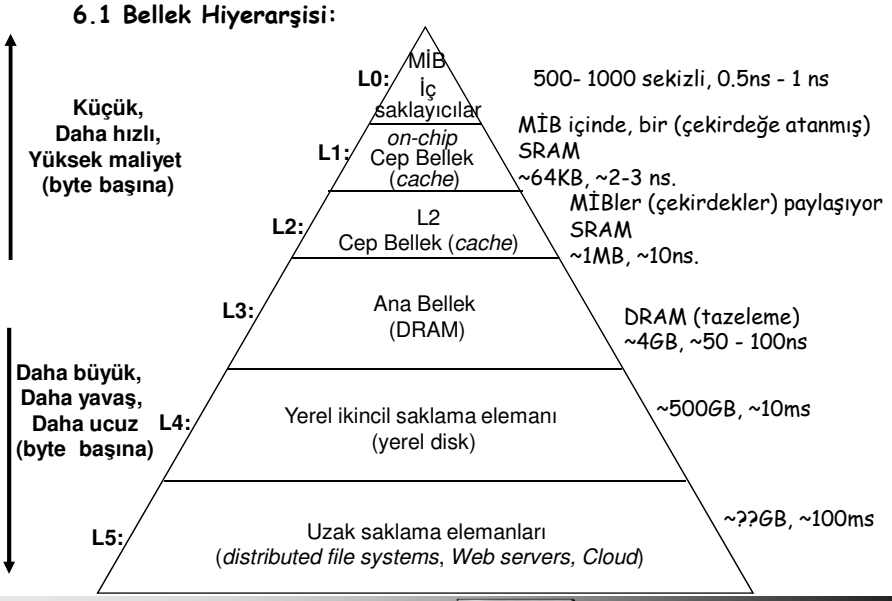
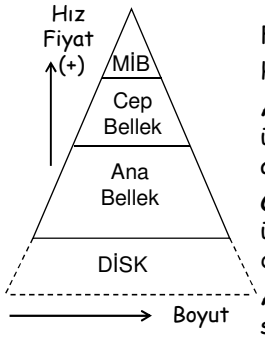
6 Bellek Organizasyonu (İç / Dış)

6.1 Bellek Hiyerarşisi:





6.1 Bellek Hiyerarşisi (devamı**):**

Farklı hız, maliyet ve boyutlarda bellekler üretilmektedir.

Hızlı belleklerin maliyetleri daha yüksektir.

Ana bellek (*Main memory*): DRAM (*Dynamic Memory*) olarak üretilir. Kapasitesi daha büyük, birim maliyeti daha düşük, ancak daha yavaştır.

Cep bellek (*Cache memory*): SRAM (*Static Memory*) olarak üretilir. İşlemci ile aynı tümleşik devrede olabilir. Kapasitesi daha küçük, birim maliyeti daha yüksek, ancak daha hızlıdır.

Amaç: Farklı belleklerden uygun miktarlarda kullanarak ve sık erişilen verileri daha hızlı olan belleğe yerleştirerek maliyeti düşük ve ortalama erişim süresi kısa hiyerarşik bir bellek sistemi oluşturulması amaçlanır.

Bu yaklaşım programlardaki **başvuru yöreselliği (*Locality of Reference*)** özelliğinden yararlanır.

Cep bellek ve ana bellek MİB'in program ve veriler için doğrudan erişebildiği sistem içi belleklerdir (*internal*).

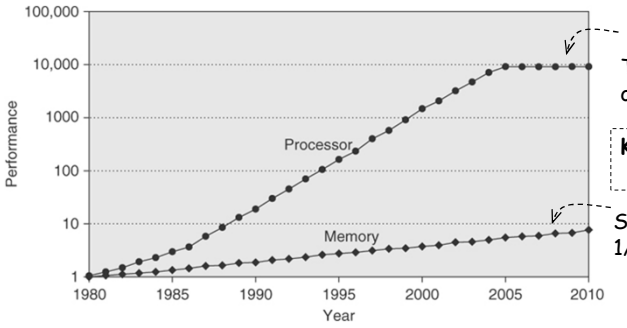
Diskteki verilere ise doğrudan erişilmez önce ana (veya cep) belleğe getirilir.

6.2 Ana bellek ile MİB arasındaki performans farkı:

Ana bellekler, dinamik belleklerden (*DRAM - dynamic RAM*) oluşurlar.

İşlemcilerin hızları ana belleklerden çok daha yüksektir.

Ana bellekler işlemcilerin bant genişliği gereksinimlerini karşılayamazlar.



Tek işlemcinin saniyedeki ortalama bellek erişimi isteği

Kaynak: Hennesy, Patterson, *Computer Architecture, 5/e*

Saniyedeki erişim sayısı, 1/gecikme

Çok işlemcili (çekirdekli) sistemlerde bant genişliği gereksinimi daha da fazladır.

3.2 GHz saat işareti frekansına sahip 4 çekirdekli Intel i7 işlemcisinin toplam bant genişliği gereksiniminin uç değeri 409.6 GB/s'dir.

DRAM tabanlı ana belleğin sağladığı bant genişliği ise bunun ancak %6'sıdır (25 GB/s)

6.3 Başvuru yöreselliği (*Locality of reference*):

Programlar son eriştikleri komut ve verilere tekrar erişme eğilimi gösterirler.

Gözlem: Programların çoğu çalışma zamanlarının %90'ını kodun sadece %10'luk kısmında geçirmektedir.

Programların kısa süre önce eriştikleri komut ve veriler gözlemlenerek yakın gelecekte erişecekleri adresler öngörülebilir.

İki tür başvuru yöreselliği vardır:

Zamanda Yöresellik (*Temporal*): Bir adrese başvurulduktan sonra büyük olasılıkla bir süre sonra aynı adrese tekrar başvurulur.

Coğrafi (Uzayda) Yöresellik (*Spatial*): Belleğe bir başvuru yapıldıktan sonra büyük olasılıkla bir sonraki başvuru yakın bir adrese olacaktır.

Yöreselliğin nedenleri:

- Programların yapısı: Komutlar birbirini izleyen adreslere yerleşir, ilgili veriler yakın adreslere yerleştirilir.
- Döngüler
- Diziler
- Tablolar

6.4 Bellek Teknolojileri

Rastgele Erişim Bellekler RAM (*Random Access Memory*):

"Rastgele erişim" terimi yanlış kullanılmaktadır, çünkü günümüzde bilgisayarlarda kullanılan tüm yarı iletken (elektronik) bellekler rastgele erişimlidir (sırasal değil).

RAM özelikleri:

- İşlemciler bellek gözlerine hem okumak hem de yazmak için doğrudan ve hızla erişebilirler (ek birimlere gerek duyulmaz).

  Bu işlemler elektrik işaretleri ile gerçekleştirilir.

  Bu tip bellekleri "Yaz Oku Bellek" olarak adlandırmak daha doğru olur.
- RAM uçucu (*volatile*) bellektir. Güç kesilirse veriler kaybolur.

  Bu nedenle RAM sadece geçici (*temporary*) saklama birimi olarak kullanılır.

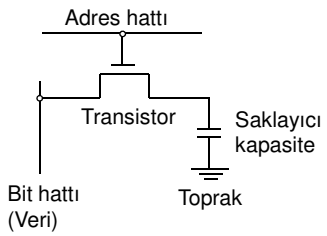
Belleklerde gecikme ölçüleri:

- **Erişim Süresi (*Access time*)**: Erişim isteğinin başlaması ile adreslenen sözcüğün okunması (yazılması) arasında geçen süre.
- **Çevrim Süresi (*Cycle time*):** İki bellek erişimi arasındaki olası en kısa süre

İki tür RAM vardır; Dinamik RAM DRAM (*Dynamic* RAM) ve

Statik RAM SRAM (*Static* RAM).

Dinamik RAM DRAM (*Dynamic* RAM):



- Ana belleklerde kullanılır.
- DRAM hücreleri (bit) verileri kapasite yükleri şeklide saklarlar.
- Kapasiteler doğal yapıları nedeniyle yüklerini kaybederler (boşalırlar). Bu nedenle dinamik bellekleri periyodik olarak (~ 8ms) **tazelemek** gerekir.
- Dinamik olarak adlandırılır, çünkü güç kesilmese bile depoladığı yük zamanla boşalır.
- Tazeleme sırasında bellek kullanılamaz (gecikme). (-)
- Her okumadan sonra veri tekrar yazılır çünkü okuma yükü boşaltır (gecikme) (-).
- Erişim süresi ile çevrim süresi farklıdır. Çevrim süresi > erişim süresi (-)
- Ucuz ve yoğundur: bir transistor/bit. Tek yongada çok sayıda bir yer alır. (+)

DRAM teknolojisinde ilerlemeler:

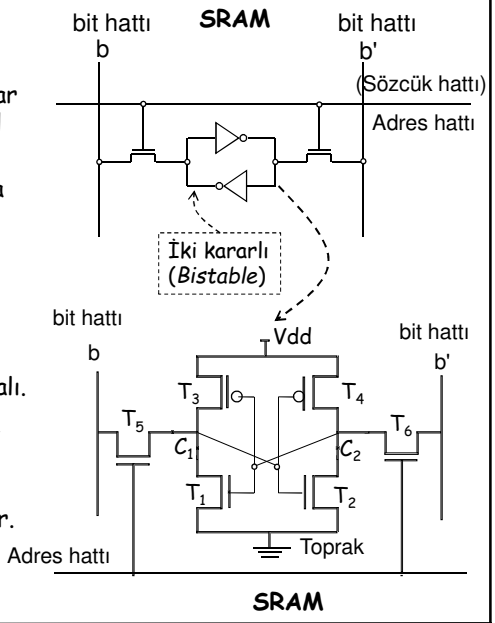
- SDRAM (*Synchronous* DRAM): Saat işareti eklenmiştir. Toplu veri aktarımı.
- DDRAM (*Double data* rate DRAM): Saat işaretinin hem çıkan hem de inen kenarında erişim yapılır.

Statik RAM SRAM (*Static* RAM):

- Cep belleklerde kullanılır.
- Verileri (bit) saklamak için "flip-flop"lar (veya tutucular) kullanılır (Bkz. Sayısal Devreler ders notları).
- Bir bit için 6 transistor kullanılır (daha pahalı, bir yongada daha az bit). (-)
- Tazeleme gerekli değil. (+)
- Erişim süresi ≈ çevrim süresi. (+)

DRAM – SRAM Karşılaştırması:

- İkisi de uçucudur; güç sürekli sağlanmalı.
- DRAM daha yoğundur (küçük hücreler, birim alanda daha çok hücre) ve aynı boyuttaki SRAM'dan daha ucuzdur.
- DRAM tazeleme devresine gerek duyar.
- DRAM gecikmesi daha fazladır.
- DRAM ucuz, SRAM hızlı.



Çağrışımlı Bellek (ÇB), İçerikle Adreslenen Bellek
(*Associative Memory, Content Addressable Memory* – CAM)

- Hızlı **arama** gerektiren uygulamalarda kullanılır.
  Cep belleklerde de hızlı arama için kullanılırlar.
- Veri, bellekten adresine göre değil, içeriğinin belli bir kısmına bağlı olarak okunur.
- Kullanıcı aranan veri sözcüğünü verir (*Argument* A) (adresi değil), ÇB (*CAM*) tüm bellek satırlarında eş zamanlı (sırasal değil) arama yapar; eğer varsa sözcüğün bulunduğu satırı belirler, yoksa verinin bellekte olmadığını belirler.
- Kullanıcı ayrıca verinin hangi kısmının aranacağını belirten anahtar (*Key* K) değerini de verir.
- Eğer verinin belirtilen kısmı bellekte bulunursa ilgili bellek satırına ait uyuşma biti "1" yapılır ve çıkışa o satırdaki veri aktarılır.
- Verileri saklamak için SRAM, eş zamanlı (paralel) arama için sayısal devre içerir.

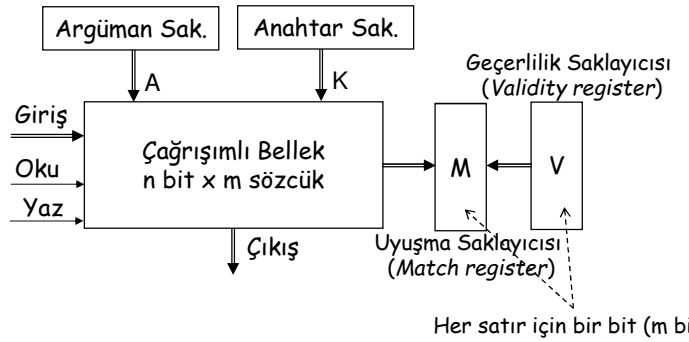
Çağrışımlı Bellek (ÇB), İçerikle Adreslenen Bellek (devamı)

İlk elektrik verildiğinde bellekte rasgele değerler olur.

Her satıra, o satırda geçerli veri olup olmadığını gösteren ek bir geçerlilik (*valid*) biti karşı düşürülür.

İlk elektrik verildiğinde denetim donanımı tüm bitleri "geçersiz" (V=0) yapar.

Daha sonra i. satıra veri yazılınca o satıra ait bir "geçerli" (V_i=1) yapılır.



Örnek:

A: 1011 1111

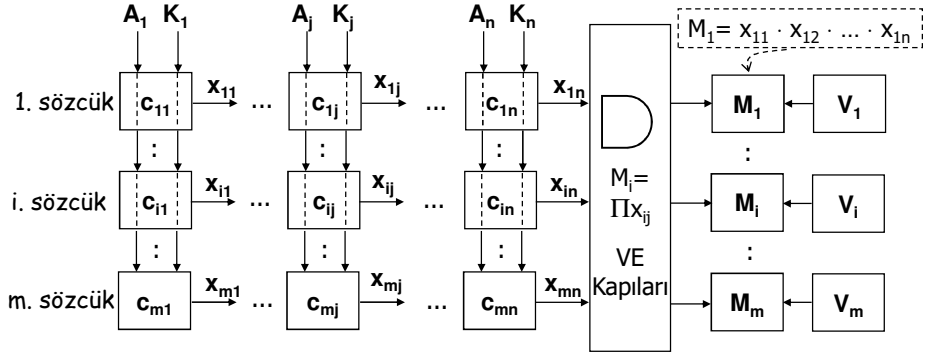
K: 1111 0000

Bellek: 1100 1111 M=0

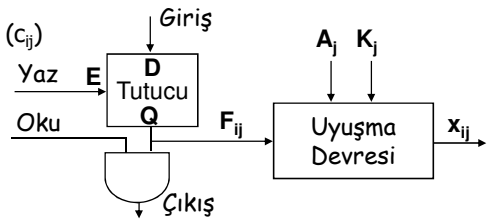
1011 1010 M=1

Çıkış: 1011 1010

İç Yapı: A_i ve K_i değerleri tüm hücrelere paralel olarak (aynı anda) gider.



Bir hücrenin (c_{ij}) yapısı:



$X_{ij} = \overline{K_j} + \overline{A_j \oplus F_{ij}}$

$X_{ij} = \overline{K_j} + A_j \odot F_{ij}$

$X_{ij} = \overline{K_j} + A_j \cdot F_{ij} + \overline{A_j} \cdot \overline{F_{ij}}$

$M_i = V_i \cdot \prod_{j=1}^{n} x_{ij}$

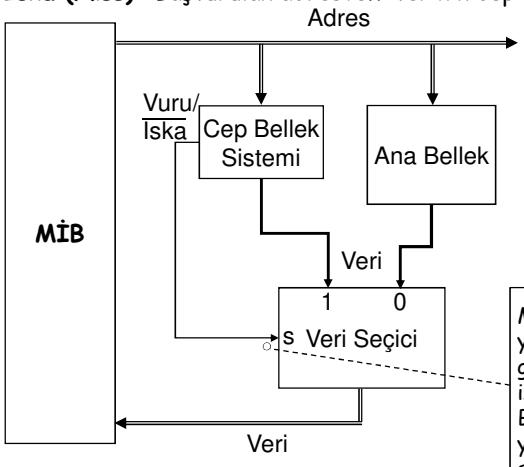
6.5 Cep Bellek (*Cache Memory*)

6.5.1 Cep Bellek Sisteminin Çalışma Mantığı

Sık başvurulan adreslerdeki verilerin **kopyalarının** cep bellekte tutulması amaçlanır.

Vuru (*Hit*): Başvurulan adresteki verinin cep bellekte bulunması.

Iska (*Miss*): Başvurulan adresteki verinin cep bellekte bulunmaması.



Örnek:
Sadece okumalar için:
Cep bellek erişim süresi: 20 ns
Ana bellek erişim süresi: 100 ns
Vuru oranı: H=0.9 (%90 vuru var.)
Ortalama bellek erişim süresi:
t_a= 0.9*20 + 0.1*100 = 28 ns.

MC68000 gibi asenkron bellek erişimi yapan sistemlerde, vuru olup olmamasına göre MİB'e farklı gecikmelerle DTACK işareti gönderilir.
Böylece veri cep bellekten alındığında yol çevriminin daha çabuk tamamlanması sağlanır.

6.5.1 Cep Bellek Sisteminin Çalışma Mantığı (devamı)

Blok aktarımı:

Iska (aranan verinin cep bellekte olmaması) durumunda erişilmek istenen verinin de içinde bulunduğu bir **blok** veri, ana bellekten cep belleğe getirilir (kopyalanır).

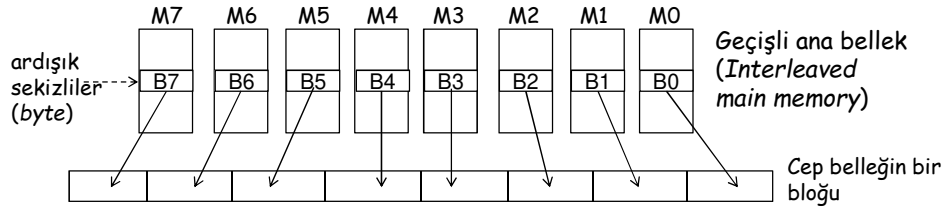
Blok aktarımının neden : coğrafi yöresellik (*spatial locality*).

Çünkü bir sonra erişilecek olan veri büyük olasılıkla şimdi erişilen verinin yakın adreslerinde olacaktır.

Tek veri yerine blok aktarımı yapılmasının olumsuz yanı uzun sürmesi olabilir.

Ana bellek ile cep bellek arasındaki blok aktarımının süresini kısaltmak için geçişli bellek (*interleaving technique*) kullanılmaktadır.

Ardışık adreslere gelen veriler farklı bellek modüllerinde saklanırlar; böylece bu verilere aynı anda erişmek mümkün olur (RAID'e benzer şekilde, bkz. Bölüm 7.2).



6.5.1 Cep Bellek Sisteminin Çalışma Mantığı (devamı)

Yer değiştirme (*Replacement*) teknikleri:

Cep belleğin boyutu ana bellekten daha küçüktür.
Belli bir anda ana bellekteki verilerin sadece bir kısmı cep bellekte bulunabilir.

Cep bellek doluyken ana bellekten yeni bir veri getirmek gerektiğinde cep bellekteki hangi bloğun kaldırılacağını belirlemek için bir yer değiştirme algoritmasına (*replacement algorithm*) gerek duyulur.

En yaygın kullanılan yer değiştirme teknikleri:

- **FIFO** (*First In First Out*): Cep bellekte en uzun süredir yer alan blok çıkartılır.
- **LRU** (*Least Recently Used*): Son zamanlarda en az kullanılan blok cep bellekten çıkartılır.

  Blokların kullanım tarihçesi dikkate alınır.

  Her bloğa atanan yaşlanma sayaçları (*aging counters*) ilgili bloğa yapılan erişimlerin kaydını tutarlar.

Cep bellek işlemleri, cep bellek yönetim birimi (*Cache Memory Controller / Cache Memory Management Unit*) adı verilen bir **donanım birimi** tarafından yapılır.

6.5.2 Cep Bellek Erişim Yöntemleri (*Mapping*)

- Ana bellekteki bir veri o anda cep bellekte var mı?
- Varsa cep belleğin neresinde yer alıyor?

1. Çağrışımlı (*Associative*) Erişim Yöntemi

a) Blok Yapısı Olmadan:

Gerçekte tüm yöntemler blok yapısı ile birlikte kullanılır.

Konuya giriş yapmak için ilk önce yöntem blok yapısı olmadan anlatılacaktır.

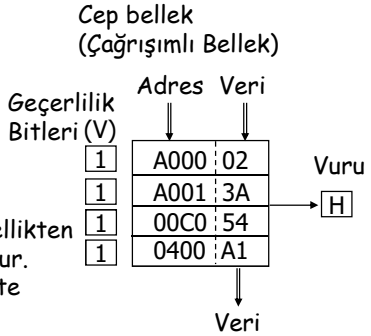
Yöntem: En çok başvurulan adresler ve içerikleri bir çağrışımlı bellekte (*Associative memory*) tutulur.

MİB tarafından üretilen adres cep bellekte (çağrışımlı bellek) aranır.

Vuru durumunda veri cep bellekten okunur.

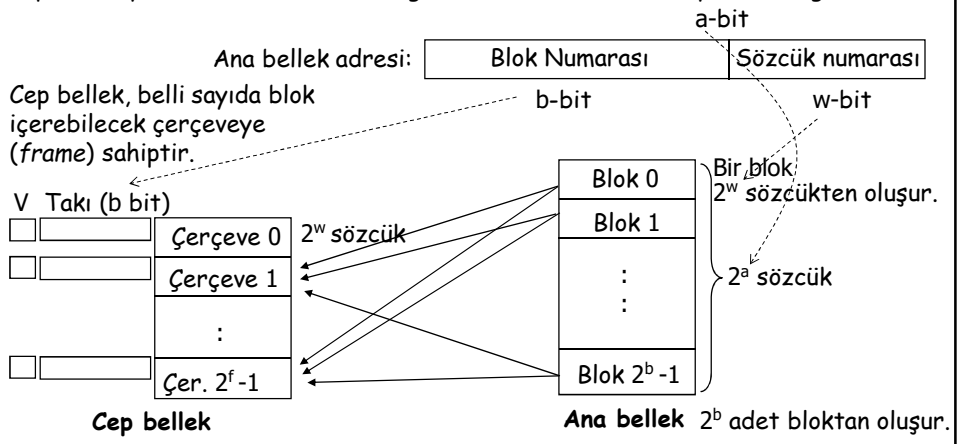
Iska durumunda veri ana bellekten okunur ve aynı zamanda cep belleğe yerleştirilir.

Blok yapısı kullanılmadığında sadece zamanda yöresellikten yararlanılır, coğrafi yöresellikten yararlanılmamış olur. Bu nedenle bu yöntem gerçekte blok yapısı ile birlikte kullanılır.



b) Blok Yapılı Tam Çağrışımlı Erişim Yöntemi (*Full Associative*)

Cep bellek yönetim sistemi MİB'ten gelen adresi iki alt alana ayırarak değerlendirir:



Ana belleğin bir bloğu cep belleğin herhangi bir çerçevesinde yer alabilir.

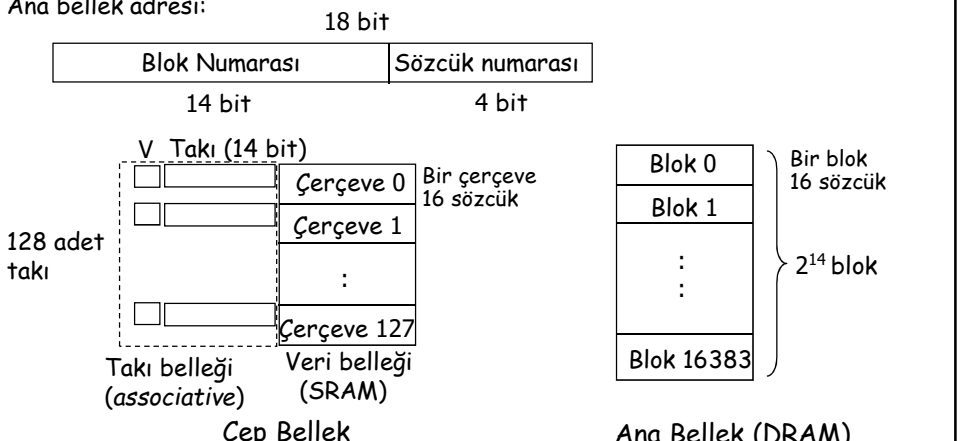
Bir çerçevede hangi bloğun olduğu takı (*tag*) bilgisinden anlaşılır.

Bu yöntemde takı bilgisi olarak ana bellek adresinin blok numarası kullanılır.

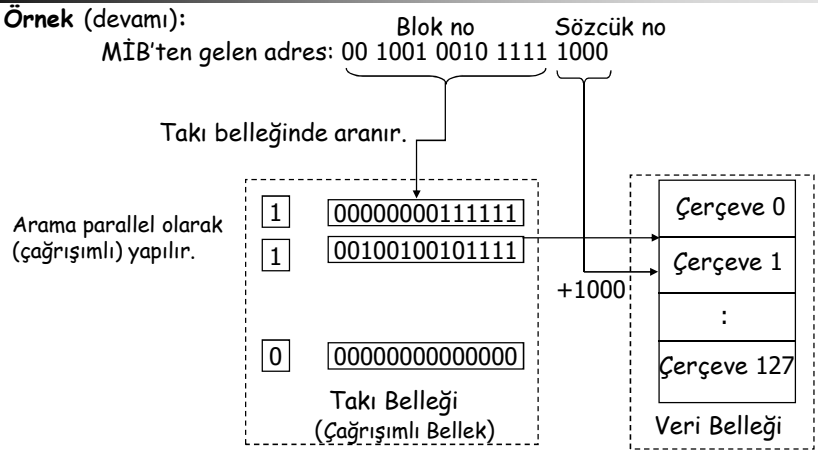
Örnek: Tam Çağrışımlı Yöntem (*Full Associative*)

Ana Bellek: 256K x sözcük — Adres: a = 18 bit
Blok boyu: 16 sözcük — w = 4 bit — Ana bellekte 2^{14} adet blok vardır. b = 14
Cep Bellek: 2K x sözcük veri taşıyabiliyor. — Cep bellekte 2^7 (128) adet çerçeve vardır. f = 7

Ana bellek adresi:



Örnek (devamı):



Eğer blok değiştirme (*replacement*) yöntemi olarak **LRU** (*Least Recently Used*) kullanılıyorsa takı belleğinde her çerçeveye ait yaşlanma sayaçları da (*aging counter*) bulunur.

Bir çerçeveye başvurulmadığı zaman o çerçevenin sayacı arttırılır.

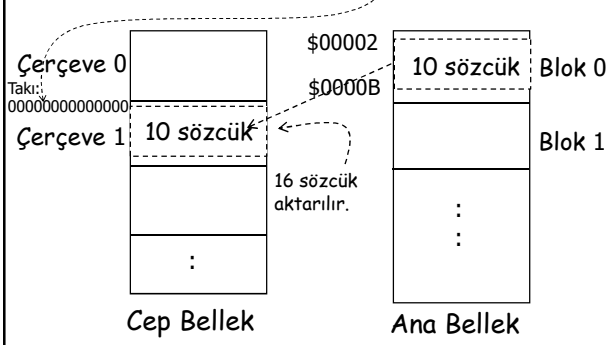
Blok değiştirme gerektiğinde "en yaşlı" çerçevedeki blok kaldırılır.

Örnek: Cep bellekte dizi erişimi

Ana Bellek: 256K x sözcük Cep Bellek: 2K x sözcük Blok boyu: 16 sözcük

Durum A) Sistemdeki bir program başlangıç adresi $00002 olan ve 10 sözcükten oluşan bir diziye erişiyor.
MİB diziye erişmeye başladığında cep bellekteki "en yaşlı" çerçeve Çerçeve 1'dir.

Dizinin başlangıç adresi: 00 0000 0000 0000 0010 ($00002)
Dizinin son adresi : 00 0000 0000 0000 1011 ($0000B)



MİB dizinin ilk sözcüğüne ($00002) eriştiğinde ıska olur.

Dizide sadece 10 sözcük olmasına rağmen cep bellek yönetim birimi Blok 0'ın tamamını (16 sözcük) Çerçeve 1'e aktarır.

MİB dizinin kalan 9 elemanına eriştiğinde vuru olur.

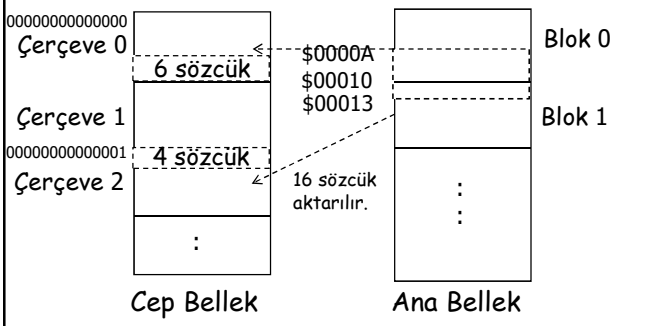
Toplam: 1 ıska; 9 vuru

Örnek: Cep bellekte dizi erişimi (devamı)

Ana Bellek: 256K x sözcük Cep Bellek: 2K x sözcük Blok boyu: 16 sözcük

Durum B) Sistemdeki bir program başlangıç adresi $ 0000A olan ve 10 sözcükten oluşan bir diziye erişiyor.
Cep bellekteki "en yaşlı" çerçeveler Çerçeve 0 ve 2'dir.

Dizinin başlangıç adresi: 00 0000 0000 0000 1010 ($0000A)
Dizinin son adresi: 00 0000 0000 0001 0011 ($00013)



Her blok ayrı (bağımsız) işlenir.

Farklı blokların takı değerleri farklıdır.

Bu örnekte iki blok aktarımına gerek olur.

Dizinin boyu bir blok (çerçeve) boyundan küçük olmasına rağmen yerleştirildiği adresten dolayı cep bellekte iki çerçeve kaplar.

Örnek: Cep bellekte dizi erişimi (devamı)

- MİB dizinin ilk sözcüğüne ($0000A) eriştiğinde ıska olur.
- Cep bellek yönetim birimi Blok 0'ın tamamını (16 sözcük) Çerçeve 0'a taşır.
- Sonraki 5 erişimde vuru olur.
- MİB dizinin 7. sözcüğüne ($00010) eriştiğinde ıska olur, çünkü bu sözcük farklı bloktadır ve adresinin takı değer farklıdır.
- Cep bellek yönetim birimi Blok 1'in tamamını (16 sözcük) LRU yöntemine göre Çerçeve 2'ye aktarır.
- Sonraki 3 erişimde vuru olur.
- Toplamda 2 ıska, 8 vuru olur.

Eğer dizinin başlangıç adresi uygun seçilseydi (örneğin $00000) dizi ana bellekte bir blok, cep bellekte de sadece bir çerçeve kaplayacaktı (durum A'da olduğu gibi).

Görüldüğü gibi dizilerin ana belleğe yerleştirilme şekilleri cep bellek sistemlerinin performansını etkilemektedir.

2. Doğrudan Dönüşüm (*Direct Mapping*)

Ana bellekteki bir bloğun cep bellekteki hangi çerçeveye yerleşebileceği belli ve sabittir.

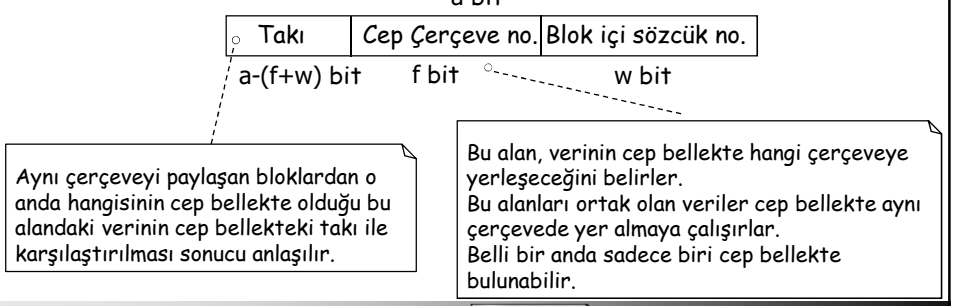
Bu nedenle bir bloğun cep bellekteki yerini aramaya gerek yoktur (önceden bellidir).

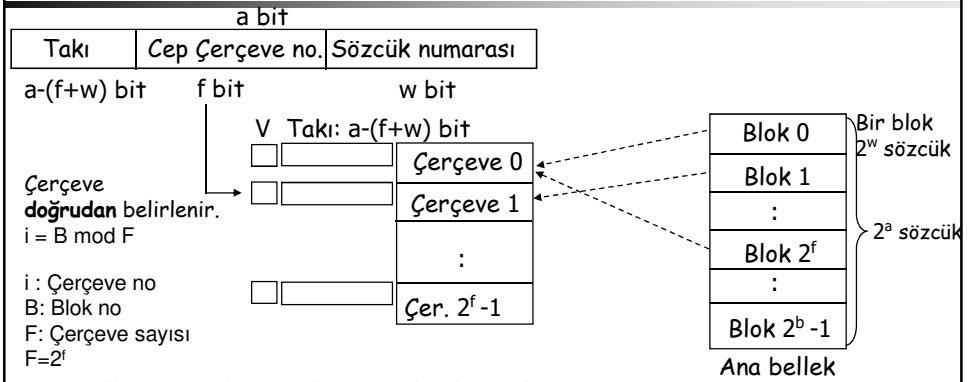
Bloğun yeri cep içinde aranmadığından çağrışımlı (*associative*) bellek kullanılmaz.

Ana belleğin boyu cep bellekten daha büyük olduğundan ana bellekteki birden fazla blok aynı çerçeveyle eşleşir.

Belli bir anda hangi bloğun ilgili çerçevede olduğunu belirlemek gerekir.

Cep bellek yönetim sistemi MİB'ten gelen adresi üç alt alana ayırarak değerlendirir:





Ana belleğin bir bloğu, sadece kendi adresindeki "cep çerçeve no" alanının belirlediği çerçevede yer alabilir.

Doğrudan dönüşüm yönteminde her bloğun cepteki yeri (çerçeve no) belli olduğundan,

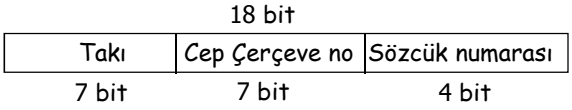
- Cep bellekte boş çerçeveler olsa bile aynı çerçeveyi kullanan iki blok aynı anda cep bellekte yer alamaz (örneğin Blok 0 ve Blok 2^f).
- Blok değiştirme aşamasında karar verme yöntemlerine gerek yoktur.
- Çağrışımlı bellek kullanımına gerek yoktur.

Örnek (*Direct Mapping*):

Ana Bellek: 256K x sözcük
Blok boyu: 16 sözcük
Cep Bellek: 2K x sözcük
veri taşıyabiliyor.

Adres: a = 18 bit
w = 4 bit Ana bellekte 2^{14} adet blok vardır. b = 14
Cep bellekte 2^7 (128) adet çerçeve vardır. f = 7

Ana bellek adresi:



Örnek sistemde aşağıdaki iki adresteki veri cep bellekte aynı çerçeveye yerleşemeye çalışacaktır.

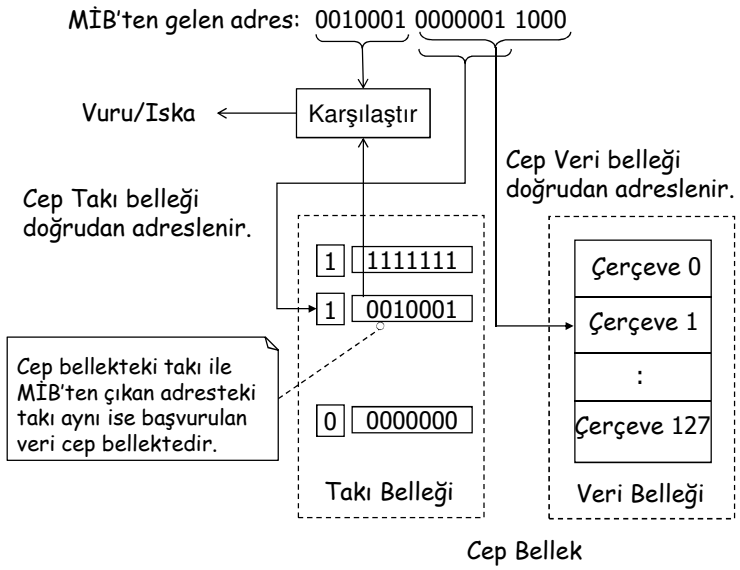
Takı	Çerçeve no	Sözcük no
0000000	0000000	XXXX
0000001	0000000	XXXX

Her ikisinin de çerçeve numarası alanları 0000000 olduğundan Çerçeve 0'da aranacaklardır.

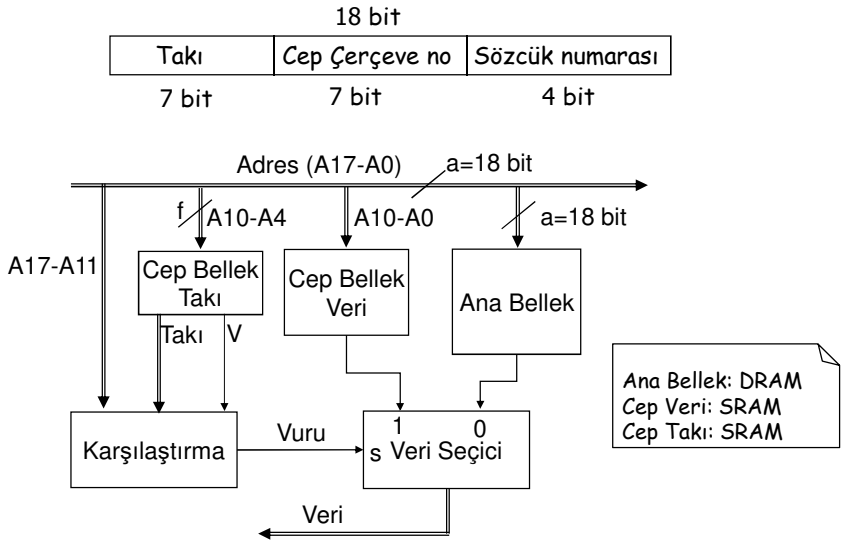
Belli bir anda bu adreslerdeki verilerden sadece bir tanesi cep bellekte bulunabilir.

O anda hangisinin gerçekte cep bellekte (Çerçeve 0) olduğu adresteki takı alanı ile cep bellekteki takı alanının karşılaştırılmasıyla belirlenir.

Örnek (*Direct Mapping*):



Örnek sistemin blok diyagramı:



3. Kümeli ve Çağrışımlı (*Set Associative*) Dönüşüm

Doğrudan erişim ile tam çağrışımlı erişim arasında bir yöntemdir.

Cep bellek **küme**lere (***set***) bölünür ve her kümede belli sayıda çerçeve bulunur.

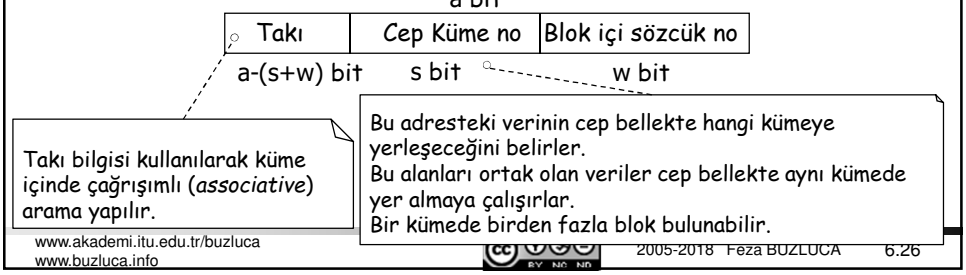
Ana bellekteki bir bloğun cep bellekteki kaç numaralı kümeye (Snum) yerleşebileceği belli ve sabittir.

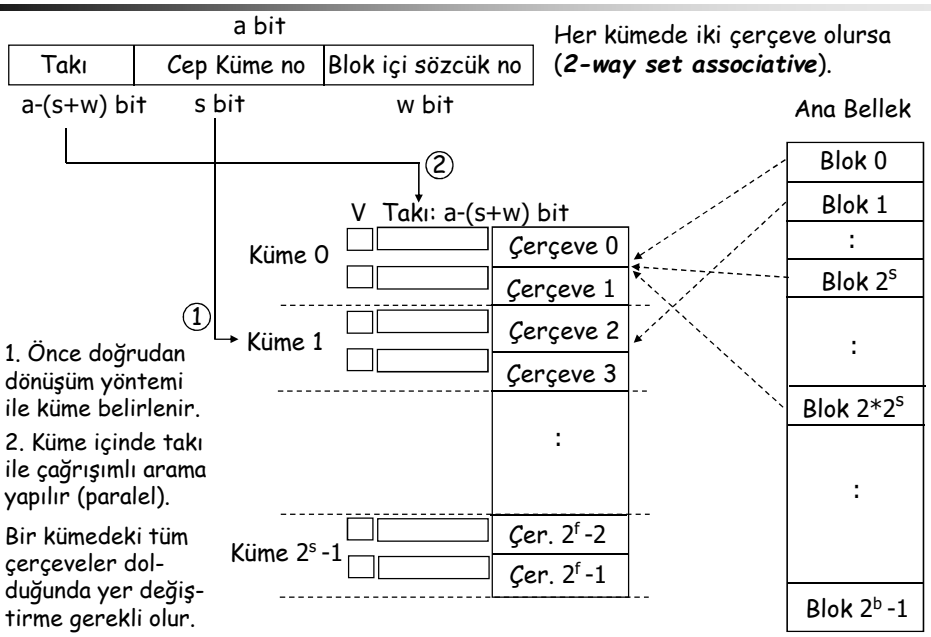
Snum = B mod S Snum: Bloğun yerleşeceği küme numarası
B: Ana bellek blok numarası, S: Cep bellekteki küme sayısı

Küme numarası sabit olmakla birlikte bir blok ilgili kümenin içindeki herhangi bir çerçeveye yerleşebilir.

- Doğrudan dönüşüm ile küme numarası belirlenir (sabit).
- Çağrışımlı yöntemle küme içindeki çerçeve belirlenir (esnek).

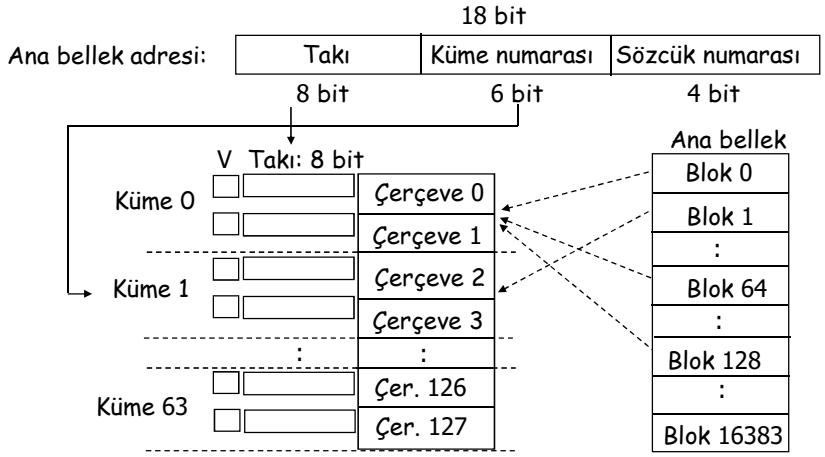
Cep bellek yönetim sistemi MİB'ten gelen adresi üç alt alana ayırarak değerlendirir:





Örnek (*Set Assosciative*) :

Ana Bellek: 256K x sözcük — Adres: a = 18 bit
Blok boyu: 16 sözcük — w = 4 bit — Ana bellekte 2^{14} adet blok vardır. b = 14
Cep Bellek: 2K x sözcük — Cep bellekte 2^{7} adet (128) çerçeve vardır. f = 7
veri taşıyabiliyor.
Her kümede 2 blok olsun. Bu durumda cep bellekte 64 küme vardır.



Dönüşüm (*mapping*) yöntemlerinin özeti:

- **Tam çağrışımlı (*Associative*) dönüşüm** en esnek ve verimli yöntemidir, çünkü ana bellekteki bir blok cep bellekteki herhangi bir uygun çerçeveye yerleşebilir.
  Olumsuz yönü ise büyük boyutlu çağrışımlı belleğin maliyetidir.
- **Doğrudan dönüşümde** her bellek bloğunun yer alabileceği cep çerçevesi sabittir.
  Temel olumsuzluk cep belleğin verimsiz kullanımıdır.
  Cep bellekte boş yerler olsa bile bir ana bellek bloğu zaten dolu olan bir cep bellek çerçevesine yerleşmeye çalışabilir.
  Bu olumsuzluk vuru oranını düşürür
  Olumlu yönü ise basitliğidir. Cep bellekte bloğun yerini aramaya gerek yoktur.
  Bloğun yeri sabit olduğu için yer değiştirme kararı vermeye de gerek yoktur.
- **Kümeli ve çağrışımlı yöntemin** verimliliği ise tam çağrışımlı yöntem ile doğrudan dönüşüm arasındadır.
  Bu yöntem küme seçiminde doğrudan dönüşümün basitliğinden yararlanır.
  Bir kümedeki çerçeve sayısını değiştirerek bu yöntemin verimliliği diğer iki yöntemden birine yaklaştırılır.
  Örneğin bir kümedeki çerçeve sayısı arttırılırsa tam çağrışımlı yönteme yaklaşılır.

6.5.3 Cep Bellek – Ana Bellek Etkileşimi

→ **Okuma (Vuru):** Veri cep bellekten okunur.

→ **Okuma (Iska):**

a) Doğrudan Okuma (*Read Through – RT*):

Veri ana bellekten MİB'e okunurken aynı anda cebe de getirilir. Cep belleğe ve ana belleğe paralel erişilir.

b) Dolaylı Okuma (*No Read Through - NRT*):

Veri ana bellekten önce cep belleğe getirilir, sonra MİB cep belleği okur.

→ **Yazma (Vuru):**

a) Doğrudan Yazma (*Write Through – WT*):

Her yazma çevriminde veri hem cep belleğe hem de ana belleğe yazılır.

(-) Bu yöntem bellek erişim süresini arttırır.

(+) Cep bellekteki verilerin her zaman ana bellekteki veriler ile uyumlu olmasını sağlar (*coherence*).

→ **Yazma (Vuru)** (devamı)**:**

b) Sonradan Yazma (*Write Back –WB*): Veriler sadece cep belleğe yazılır.

Değişikliğe uğrayan blok ancak cep bellekten kaldırılırken ana belleğe yazılır.

İki tür sonradan yazma yöntemi vardır: Basit sonradan yazma (*Simple Write Back – SWB*) ve Bayraklı sonradan yazma (*Flagged Write Back – FWB*).

- **Basit sonradan yazma** (*Simple Write Back – SWB*):

  Cep bellekten çıkartılan her blok ana belleğe yazılır.

  Verilerin cep bellekteyken değişip değişmediği kontrol edilmez.

- **Bayraklı sonradan yazma** (*Flagged Write Back – FWB*):

  Sadece değişikliğe uğramış olan bloklar cep bellekten çıkartılırken ana belleğe yazılır.

  Cep bellekteyken değişen blokları belirleyebilmek için cep bellekteki takı belleğinde her çerçeve için bir "**kirlenme**" (***dirty***) biti bulunur.

  Cep bellekteki bir blok kaldırılırken bulunduğu çerçevenin kirlenme biti kontrol edilir.

  Eğer bu blok cep bellekteyken değiştiyse ana belleğe geri yazılır.

  Eğer blok cep bellekteyken değişmemişse ana bellekten yeni gelen blok doğrudan bu çerçeveye yerleştirilir.

→ **Yazma (Iska):**

a) Cebe Yükleyerek Yazma (*Write Allocate - WA*): Ana bellekte değiştirilen blok aynı zamanda cep belleğe de getirilir.

b) Cebe Yüklemeden Yazma (*No Write Allocate - NWA*): Veri sadece ana belleğe yazılır.

Daha sonra eğer bu veri okunmak istenirse ıska olacağından ilgili blok cep belleğe getirilir.

Doğrudan yazma (WT) yöntemi cebe yükleyerek (WA) ya da yüklemeden yazma yöntemleri (NWA) ile birlikte kullanılabilir. WTWA, WTNWA

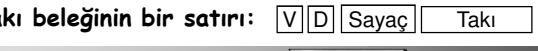
Sonradan yazma (WB) yönteminde cep belleği sürekli güncel tutmak için cebe yükleyerek (WA) yazma kullanılır. WBWA

Takı beleğinde bulunabilen bilgiler:

Geçerlilik (V) ve takı bilgisine ek olarak, kullanılan yöntemlere bağlı olarak takı belleğinde aşağıdaki bilgiler de bulunabilir:

LRU kullanılıyorsa yaşlanma sayaçları, bayraklı sonradan yazma (*Flagged Write Back – FWB*) yöntemi kullanılıyorsa "**kirlenme**" (D) biti.

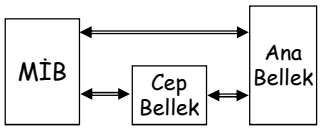
Takı beleğinin bir satırı:



6.5.4 MİB - Cep Bellek – Ana Bellek Bağlantısı

MİB - cep bellek – ana bellek bağlantıları iki farklı şekilde yapılabilir:

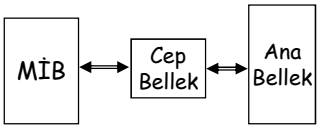
a) Paralel bağlantı



- ***Read Through* (RT):** Blok ana bellekten cep belleğe aktarılırken gerekli veri aynı zamanda MİB tarafından paralel olarak okunabilir.
- ***Load Through* (LT):** MİB cep belleğe veri yazarken veri aynı zamanda paralel olarak ana belleğe de yazılır.

Paralel yapı özellikle Doğrudan Yazma (*Write Through – WT*) yöntemi için uygundur. Sonradan Yazma (*Write Back –WB*) yöntemi ile de kullanılabilir.

b) Seri bağlantı



- *No* ***Read Through* (NRT):** Blok önce ana bellekten cep belleğe aktarılır ardından gerekli veri MİB tarafından cep bellekten okunur.
- ***No Load Through* (NLT):** MİB veriyi cep belleğe yazar, ardından veri cep bellekten ana belleğe aktarılır.

Seri yapı Sonradan Yazma (*Write Back –WB*) yöntemi için uygundur.

6.5.5 Erişim Süreleri:

t_a : Ortalama Bellek Erişim süresi (*Average memory access time*)
w : Yazma oranı (*Write ratio*) (yazma erişimleri sayısı / toplam erişim sayısı)
h : Vuru oranı (*Hit ratio*)
t_{cache} : Cep bellek erişim süresi (*Cache memory access time*)
t_{main} : Ana bellek erişim süresi (*Main memory access time*)
t_{trans}: Cep belleğe blok aktarım süresi (*Time to transfer block to cache*)
w_d: Blok değişme (kirlenme) olasılığı

	WT, RT/LT (Doğrudan Yazma , Paralel okuma/yazma)		**WB,WA, NRT/NLT** (Sonradan Yazma, Seri okuma/yazma)	
	NWA	**WA**	**SWB**	**FWB**
Okuma Vuru (1-w)h	t_{cache}	t_{cache}	t_{cache}	t_{cache}
Okuma Iska (1-w)(1-h)	t_{trans}	t_{trans}	$2t_{trans}+t_{cache}$	$w_d\,(2t_{trans}+t_{cache})$ + $(1-w_d)(t_{trans}+t_{cache})$
Yazma Vuru wh	t_{main}	t_{main}	t_{cache}	t_{cache}
Yazma Iska w(1-h)	t_{main}	$t_{main}+t_{trans}$	$2t_{trans}+t_{cache}$	$w_d\,(2t_{trans}+t_{cache})$ + $(1-w_d\,)(t_{trans}+t_{cache})$

Erişim Süreleri Hesabı:

- **Write Through with Write Allocate, Read/Load Through (WTWA, RT/LT):**

Okuma Vuru + Okuma Iska + Yazma Vuru + Yazma Iska

$t_a = (1-w)h\, t_{cache} + (1 - w)(1-h)t_{trans} + w{\cdot}h{\cdot}t_{main} + w(1-h)(t_{main} + t_{trans})$

$t_a = (1 - w)h\, t_{cache} + (1 - h)t_{trans} + w{\cdot}t_{main}$

- **Write Through with No Write Allocate, Read/Load Through (WTNWA,RT/LT)**

$t_a = (1 - w)h\, t_{cache} + (1-w)(1-h)t_{trans} + w{\cdot}h{\cdot}t_{main} + w(1-h)t_{main}$

$t_a = (1 - w)h\, t_{cache} + (1 - w)(1 - h)t_{trans} + w{\cdot}t_{main}$

- **Simple Write Back with Write Allocate,No Read Through (SWBWA,NRT/NLT)**

Okuma Vuru + Okuma Iska +Yazma Vuru + Yazma Iska

$t_a = (1 - w)h\, t_{cache} + (1 - w)(1-h)(2t_{trans} + t_{cache}) + w{\cdot}h{\cdot}t_{cache} + w(1 - h)(2t_{trans} + t_{cache})$

$t_a = t_{cache} + (1 - h){\cdot}2{\cdot}t_{trans}$

t_{trans} terimlerinden biri cep bellekteki bloğu ana belleğe geri koymak için, diğeri de yeni bloğu ana bellekten cep belleğe getirmek içindir.

- **Flagged Write Back,Write Allocate, No Read Through (FWBWA,NRT/NLT):**

$t_a = t_{cache} + (1 - h)t_{trans} + w_d{\cdot}(1 - h)t_{trans}$

$t_a = t_{cache} + (1 - h)(1+w_d)t_{trans}$

Cep bellek içeren örnek işlemciler:

- **Intel386™ ve öncesi:** Cep bellek işlemci tümdevresinin dışında SRAM bellek.
- **Intel486™ (1989)**
  8-KByte on-chip (L1)
- **Intel® Pentium® (1993)**
  L1 on-chip: 8 KB komut, 8 KB veri cebi (Harvard mimarisi)
- **Intel P6 Ailesi: (1995-1999)**
  - **Intel Pentium Pro:**
    L1 on-chip: 8 KB komut, 8 KB veri cebi (Harvard mimarisi)
    İlk defa bu yapıda L2 cep bellek işlemci ile aynı yapının içine tümleştirildi.
    L2 on-chip: 256 KB. L1 ve L2 cep belleklerin işlemci ile bağlantıları farklıdır.
  - **Intel Pentium II:**
    L1 on-chip: 16 KB komut, 16 KB veri cebi (Harvard mimarisi)
    L2 on-chip: 256 KB, 512 KB, 1 MB
- **Intel® Pentium® M (2003)**
  L1 on-chip: 32 KB komut, 32 KB veri cebi
  L2 on-chip: 2 MByte'a kadar

Intel® Core™ i7-980X Processor Extreme Edition (2010)
Tek tümdevrede çok işlemci var: 6 Çekirdek (core)
12 MB smartcache: Tüm çekirdekler kullanır.