

6.4 Önbellek

Bir bilgisayar işlemcisi çok hızlıdır ve sürekli olarak hafızadan veri okur. Genellikle işlemci verinin ulaşmasını beklemek zorunda kalır, çünkü hafıza erişim süreleri işlemci hızından daha yavaştır. Önbellek, muhtemelen çok yakın gelecekte ihtiyaç duyulacak türdeki verinin saklanması için işlemci tarafından kullanılan, küçük ve geçici (temporary), aynı zamanda da hızlı bir hafızadır.

Çevremizde önbelleğin bilgisayarlar haricinde de pek çok örneği bulunmaktadır. Bu örnekleri aklınızda tutmanız bilgisayarda kullanılan önbelleği daha iyi anlamanız için size yardımcı olacaktır. Evinizin garajında çok büyük bir alet kutunuz olduğunu düşünün. Bodrumda evinizin geliştirmek üzere bir proje üzerinde çalışıyorsunuz. Bu projede matkap, ayarlı anahtar, çekiç, şerit metre, çeşitli testereler ve çeşitli tornavidalar kullanmanız gerektiğini biliyorsunuz. İlk yapmanız gereken ölçmek ve bir miktar tahta kesmek. Garaja gider şerit metreyi kocaman bir alet kutusundan alırsınız, bodruma koşarsınız, tahtayı ölçersiniz, sonra yeniden garaja koşarsınız, şerit metreyi bırakırsınız ve testereyi alırsınız, bodruma testereyle dönüp tahtayı kesersiniz. Sonra birkaç tahta parçasını vidalayarak birleştirmeye karar verirsiniz. Garaja koşar testereyi alırsınız, bodruma geri dönüp vidaların koyulacağı delikleri matkapla delersiniz, garaja geri dönersiniz, testereyi bırakıp anahtarı alırsınız ve bodruma geri dönersiniz, yanlış anahtarı aldığınızı farkeder ve garajdaki alet kutunuza gidersiniz, başka bir anahtar alıp tekrar bodruma... Durun! Gerçekten bu şekilde mi çalışırdınız? Hayır! Mantıklı bir insan olarak kendi kendinize şöyle düşünürsünüz: “Bir anahtara ihtiyacım varsa, daha sonra farklı büyüklükteki bir başka anahtara daha ihtiyacım olabilir. Öyleyse tüm anahtar setini yanıma almalıyım.” Bunu bir adım ileri götüreceğiz olursak: “Aletlerden biriyle işim bittiğinde diğerine ihtiyacım olacak. Öyleyse küçük bir alet kutusu alıp bodruma inmeliyim.” Bu şekilde ihtiyaç duyduğunuz aletleri elinizin altında tutuyorsunuz, böylece onlara kolaylıkla ulaşıyorsunuz. Böylece kolay erişim ve hızlı kullanım için birkaç aleti önbelleğe almış oldunuz. Muhtemelen ihtiyaç duymayacağınız aletleri ise ulaşmak için daha fazla zaman harcayacağınız uzak bir yerde bıraktınız. Önbelleğin yaptığı şey tamamen bundan ibaret: İşlemci tarafından daha önceden erişilmiş veya erişilmesi muhtemel olan veriyi daha hızlı ve daha yakında olan bir hafızada tutar.

Bir başka önbellek örneği bakkal dükkanlarında bulunur. Nadiren bakkala tek parça ürün almaya gidersiniz. Hemen lazım olan ürünlerle birlikte kısa zaman içinde lazım olabilecek ürünleri de alırsınız. Bakkal dükkanı ana hafızaya, eviniz de önbelleğe benzer. Bir başka örnek olarak, kaç kişinin bütün bir telefon fihristi taşıdığını düşünün. Bunun yerine çoğumuz küçük bir adres defteri taşırız. En çok arayacağımız kişilerin isimlerini ve telefon numaralarını yazarız. Adres defterimizde bir numarayı bulmak bir telefon fihristine göre çok daha kolaydır, ismin yerini bulup numarayı alırız. Telefon fihristini evde bir çekmecede bulundururuz ama adres defterini el altında tutmaya çalışırız. Çünkü telefon fihristini nadiren kullanırız. Adres defteri telefon fihristine oranla çok daha küçüktür, ancak birini arayacağımız zaman o kişi büyük olasılıkla adres defterimizde bulunur.

Araştırma yapan öğrenciler bir başka basit önbellek örneği veriyorlar. Kuantum hesaplamaları üzerine bir tez yazdığınızı varsayın. Kütüphaneye gidip bir kitap alırsınız, sonra eve dönüp bu kitaptan gerekli bilgiyi alırsınız, tekrar kütüphaneye gidip bir başka kitap alırsınız ve eve dönerseniz... Bu şekilde mi çalışırdınız? Hayır. Kütüphaneye gidersiniz, ihtiyacınız olabilecek tüm kitapları alır ve sonra eve dönersiniz. Kütüphane ana hafızaya ve eviniz de yine önbelleğe benzer.

Son bir örnek olarak, yazarlarınızdan birinin ofisini nasıl kullandığını düşünelim. İhtiyaç duymayacağı (veya altı aydan daha fazla bir süredir kullanmadığı) gereçler büyük dosya dolaplarına kaldırılır. Ancak sıkça kullandığı veriler masasının üstünde, el altında bulunur ve bunlara kolaylıkla ulaşabilir. Bir dosyadan herhangi bir şeye ihtiyaç duyduğunda, klasörden iki evrağı almak yerine dosyanın tümünü alır. Sonra bu dosya masasının üstündeki diğer evrakların yanına eklenir. Dosya dolapları yazarın ana hafızası, masası da önbelleğidir.

Önbelleğin çalışma şekli de bu örneklerle aynı temel ilkelere dayanır. Verileri getirmek için sürekli olarak ana hafızaya erişilmemesi amacıyla sıkça kullanılan veri önbelleğe kopyalanır. Önbellek yazarımızın masası gibi düzenlenmemiş halde olabilir ya da adres defteriniz gibi düzenli olabilir. Her iki şekilde de veri erişilebilir olmalıdır. Bilgisayardaki önbellek gerçek hayattaki örneklerden önemli bir farklılık gösterir: Bilgisayar hangi verinin erişilmesi muhtemel olduğunu bilemez, böylece yer ilkesini kullanır ve ana hafızaya erişmesi gerektiğinde bütün bir bloğu önbellekten ana hafızaya aktarır. Eğer bu blokta kullanılma olasılığı yüksek olan bir başka parça varsa, bütün bir blok erişim zamanını azaltır. Bu yeni bloğun önbellekteki yeri iki şeye bağlıdır: Önbellek eşleme planı (cache mapping policy, bir sonraki bölümde görülecek) ve önbellek boyutu (yeni bir blok için yer problemi).

Önbelleğin boyutu pek çok farklılıklar gösterebilir. Genellikle kişisel bilgisayarların Level 2 (L2) önbelleği 256K veya 512K'tır. Level 1 (L1) önbellek daha küçüktür, genellikle 8K veya 16K boyutundadır. L1 önbelleği işlemcide yer alır, L2 önbelleği ise ana hafıza ile işlemci arasında yer alır. Bu nedenle L1 önbelleği L2'den daha hızlıdır. L1 ve L2 arasındaki ilişki bakkal dükkanı örneğimizdeki gibi düşünülebilir: Dükkan ana hafıza ise, L2 önbelleği sizin buzdolabınız, L1 önbelleği ise sofranızdır.

Önbelleğin amacı, yakın geçmişte kullanılmış olan verileri işlemciye ana hafızadan daha yakın bir yerde tutarak hafıza erişimini hızlandırmaktır. Önbellek ana hafıza kadar büyük olmasa da oldukça hızlıdır. Ana hafıza genellikle 60ns erişim süresi olan DRAM'dan oluşur, önbellek ise genellikle DRAM'dan çok daha hızlı ve kısa dönüş süreli (cycle time) olan SRAM'den oluşur (genellikle 10ns erişim süresi vardır). İyi performans vermesi için önbelleğin çok büyük olması gerekmez. Genel fikir önbelleği yeterince küçük yapmaktır. Böylelikle bit başına ortalama maliyeti ana hafızanıninkine yakın olur, yararlı olmak için de yeterince büyüktür. Bu hızlı hafıza oldukça pahalı olduğundan tüm ana hafızanın bu teknolojiyle yapılması mümkün değildir.

Önbelleği özel yapan nedir? Önbellek adresle erişilmez, içerikle erişilir. Bu nedenle önbelleğe içeriği adreslenebilen hafıza (content addressable memory - CAM) adı da verilir. Birçok önbellek eşleme planına göre, istenen verinin önbellekte olup olmadığı denetlenmelidir. İstenen verinin yerinin belirlenmesi işlemi basitleştirmek için pek çok önbellek eşleme algoritması kullanılır.

6.4.1 Önbellek Eşleme Planları

Önbelleğin işlevsel olabilmesi için yararlı veriyi saklaması gerekir. Ancak işlemci tarafından bulunamazsa bu veri yararsız hale gelir. Veriye ya da komutlara ulaşırken işlemci bir ana hafıza adresi üretir. Eğer veri önbelleğe kopyalanmışsa, verinin önbellekteki adresi ana hafızadaki adresiyle aynı olmaz. Örneğin ana hafızada 2E3 adresinde bulunan veri önbelleğin ilk adres yerinde bulunabilir. Peki işlemci önbelleğe kopyalanan verinin yerini nasıl bilebilir? İşlemci ana hafıza adresini önbellek adresine çeviren özel bir eşleme planı kullanır.

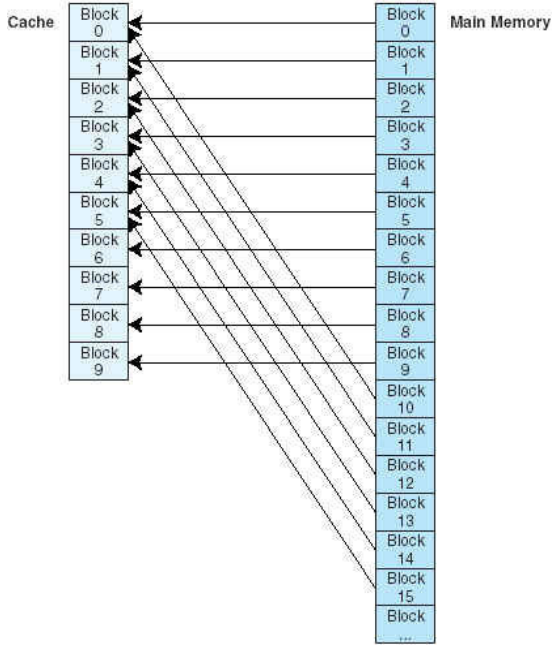
Bu adres çevrimi ana hafızadaki bitlere özel değer verilerek yapılır. Öncelikle bitleri alanlar dediğimiz belirli gruplara ayırırız. Eşleme planına bağlı olarak iki ya da üç alanımız olur. Bu alanları nasıl kullanacağımız, kullandığımız eşleme planına bağlıdır. Eşleme planı verinin nerede yer aldığını, orijinalinden önbelleğe ne zaman kopyalandığını belirler ve ayrıca işlemciye önbelleğin aranması sırasında önceden kopyalanmış olan verinin bulunması için bir yöntem sağlar.

Bu eşleme planlarını incelemeyen önce, verinin önbelleğe nasıl kopyalandığını anlamak önemlidir. Ana hafıza ve önbellek aynı boyuttaki bloklara bölünür (blokların boyutları farklılıklar gösterebilir). Bir hafıza adresi üretildiğinde, önbellek istenen kelimenin (word) burada olup olmadığını anlamak için araştırılır. İstenecek kelime önbellekte bulunursa, bu kelimenin bulunduğu tüm ana hafıza bloğu önbelleğe yüklenir. Daha önce de bahsettiğimiz gibi, bu plan yer ilkesinden dolayı başarılıdır. Eğer bir kelime şimdi referans edilmişse, büyük bir olasılıkla bu kelimenin yakınındaki kelimeler de pek yakında referans edilecektir. Bu nedenle bir kelimenin kaybedilmesi genellikle çok sayıda kelimenin bulunmasıyla sonuçlanır. Örneğin, siz bodrum katındasınız ve ilk kez aletlere ihtiyaç duydunuz, unuttuğunuz bir şey var ve garaja gitmelisiniz. Aletleri takım halinde alıp bodruma döndüğünüzde, ev geliştirme projenizde çalışırken defalarca garaja gitmek zorunda kalmayacağınızı umuyorsunuz. Önbellekteki bir kelimeye ulaşmak (zaten bodrumda olan bir alet) ana hafızadaki bir kelimeye ulaşmaktan (garaja tekrar gitmek) çok daha hızlı olduğundan, önbellek genel erişim süresini hızlandırır.

Peki ana hafıza adresindeki alanları nasıl kullanırız? Ana hafıza adresinin bir alanı, eğer veri önbellekteyse (cache hit) önbellekte bulunduğu yeri, değilse (cache miss) nereye yerleştirileceğini gösterir. (İlişkili eşlenen önbellek için durum biraz farklıdır, kısaca bahsedeceğimiz). Referans edilen önbellek bloğunun geçerliliği kontrol edilir. Bunu gerçekleştirmek için önbellek bloğuyla geçerli bir bit ilişkilendirilir. 0 geçerli biti önbellek bloğunun geçersiz olduğunu gösterir (cache miss), bu durumda ana hafızaya erişilmelidir. 1 geçerli biti ise geçerli olduğunu gösterir (cache hit olabilir, ancak emin olmak için tamamlamamız gereken bir adım daha vardır). Bundan sonra önbellek bloğundaki etiket ile adresimizin etiket alanını karşılaştırırız (Etiket önbellekte karşılık gelen blokla birlikte depolanan ana hafıza adresinden türetilmiş özel bir bit grubudur). Eğer etiketler aynıysa, istediğimiz önbellek bloğuna ulaştık demektir (cache hit). Bu noktada istediğimiz kelimenin bloktaki yerini belirlemeliyiz. Bunun için kelime alanı (word field) adı verilen ana hafıza adresinin farklı bir bölümü kullanılır. Tüm önbellek eşleme planları kelime alanına ihtiyaç duyar, ancak kalan alanlar eşleme planı tarafından belirlenir. Bu bölümde üç temel eşleme planını inceleyeceğiz.

Doğrudan Eşlenmiş Önbellek (Direct Mapped Cache)

Doğrudan eşlenen önbellek modüler bir yaklaşım kullanarak önbellek eşlemelerini atar. Çünkü önbellek bloğundan daha fazla ana hafıza bloğu vardır. Burada ana hafıza bloklarının önbellek yerleri için yarıştığı açıkça bellidir. Doğrudan eşleme, ana hafızadaki X bloğunu önbellekteki mod N 'ye göre Y bloğuna eşler. Burada N önbellekteki toplam blok sayısıdır. Örneğin, eğer önbellek 10 blok içeriyorsa, ana hafızadaki blok 0 önbellekteki blok 0'a, ana hafızadaki blok 1 önbellekteki blok 1'a, ... , ana hafızadaki blok 9 önbellekteki blok 9'a, ana hafızadaki blok 10 önbellekteki blok 0'a eşlenir (Şekil 6.2). Bundan dolayı, ana hafızadaki blok 0 ve 10 (ve 20, 30 ...) önbellekteki blok 0 için yarışır.



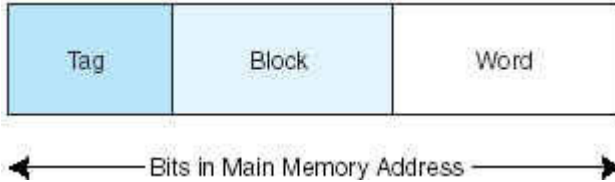
Şekil 6.2: Ana Hafıza Bloklarının Önbellek Bloklarına Doğrudan Eşlenmesi

Eğer ana hafızada hem blok 0 hem de blok 10 önbellekteki blok 0'a eşlenirse, işlemci verilen bir zamanda hangi bloğun önbellekteki blok 0'da bulunduğunu nasıl bilebilir? Cevap, her blok önbelleğe kopyalanır ve önceden tanımlı olan etiket ile tanımlanır. Önbelleğe daha yakından bakacak olursak, sadece ana hafızadan kopyalanan veriden çok daha fazlasını depolandığını görürüz. Şekil 6.3'de iki geçerli önbellek bloğu vardır. Blok 0 ana hafızadan çok sayıda kelimeyi içerir ve "00000000" etiketiyle tanımlanır. Blok 1 de bazı kelimeler içerir ve "11110101" etiketiyle tanımlanır. Diğer iki önbellek bloğu geçerli değildir.

Block	Tag	Data	Valid
0	00000000	words A, B, C,...	1
1	11110101	words L, M, N,...	1
2	-----		0
3	-----		0

Şekil 6.3: Önbelleğe Yakından Bakış

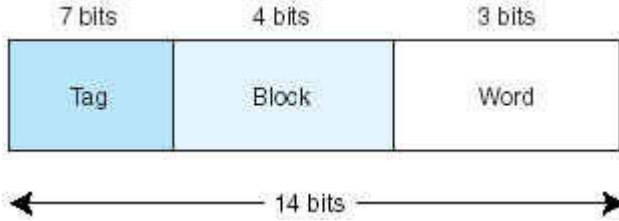
Doğrudan eşleme uygulayabilmek için ikili ana hafıza adresi Şekil 6.4'te gösterilen alanlara bölünür.



Şekil 6.4: Doğrudan Eşleme Kullanılan Ana Hafıza Adresinin Biçimi

Her alanın boyutu ana hafıza ve önbelleğin fiziksel özelliklerine bağlıdır. Kelime alanı (word field – offset field) belirli bir bloktan bir kelimeyi eşsiz bir şekilde tanımlar, bunun için de tahsis edilen sayıda biti içermelidir. Bu blok alanı (blok field) için de geçerlidir, eşsiz bir önbellek bloğu seçmelidir. Etiket alanı (tag field) ise kalan kısımdır. Ana hafızanın bir bloğu önbelleğe kopyalandığında, bu etiket blokla birlikte depolanır ve eşsiz bir şekilde bu bloğu tanımlar. Bu üç alanın toplamı bir ana hafıza adresindeki bit sayısına eklenir.

Örnek: 2^{14} kelimededen (word) oluşan bir hafıza olduğunu varsayın, önbellekte 16 blok olsun ve her blokta 8 kelime olsun. Buna göre bu hafızada $2^{14}/2^{13} = 2^{11}$ blok vardır. Bildiğimiz gibi her ana hafıza adresi 14 bite ihtiyaç duyar. Bu 14 bit adres alanının en sağdaki 3 biti kelime alanını yansıtır (bir bloktaki 8 kelimenin birini eşsiz bir şekilde tanımlamak için 3 bite ihtiyaç duyarız). Önbellekteki belirli bir bloğu seçmek için 4 bite ihtiyaç duyarız, böylece blok alanı ortadaki 4 bitten oluşur. Kalan 7 bit etiket alanını oluşturur. Bu alanlar boyutlarıyla birlikte Şekil 6.5'te gösterilmiştir.



Şekil 6.5: Örneğimiz İçin Ana Hafıza Adres Biçimi

Daha önce de bahsettiğimiz gibi her blok için gerekli etiket bu blokla birlikte önbellekte depolanır. Bu örnekte ana hafızadaki blok 0 ve 16 önbellekteki blok 0'a eşlendiğinden, etiket alanı blok 0 ve blok 16'nın birbirinden ayrılabilmesine izin verir. Blok 0'daki ikili adresler, üstte en soldaki 7 bitte bulunan blok 16'dakilerden farklıdır. Bu nedenle etiketler farklı ve eşsizdir.

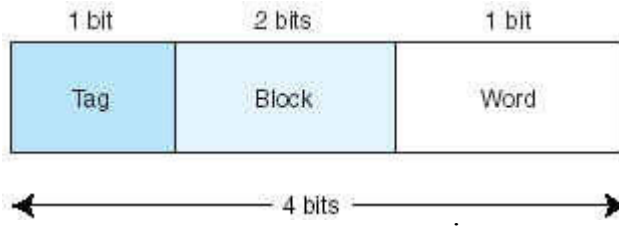
Bu adreslerin nasıl farklı olduğunu anlamak için daha küçük ve daha basit bir örneği inceleyelim. 8 bloğa bölünmüş 16 kelimelik bir ana hafıza ile doğrudan eşlemeyi kullanan bir sistemimiz olduğunu varsayalım (her blokta 2 word vardır). Önbellek 4 blok boyutunda olsun (toplam 8 kelime için). Tablo 6.1 ana hafıza bloklarının önbelleğe nasıl eşlendiğini göstermektedir.

Ana Hafıza	Eşlenir	Önbellek
Blok 0 (0, 1 adresler)	→	Blok 0
Blok 1 (2, 3 adresler)	→	Blok 1
Blok 2 (4, 5 adresler)	→	Blok 2
Blok 3 (6, 7 adresler)	→	Blok 3
Blok 4 (8, 9 adresler)	→	Blok 0
Blok 5 (10, 11 adresler)	→	Blok 1
Blok 6 (12, 13 adresler)	→	Blok 2
Blok 7 (14, 15 adresler)	→	Blok 3

Bildiğimiz gibi:

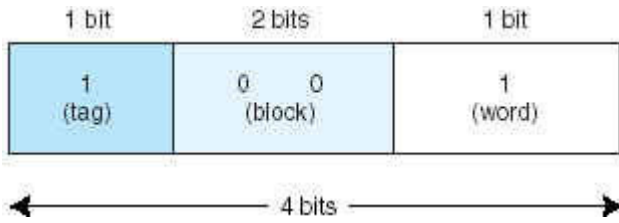
- Bir ana hafıza adresi 4 bitlidir (çünkü ana hafızada 2^4 veya 16 kelime vardır).
- Bu 4-bit ana hafıza adresi üç alana ayrılır: Kelime alanı 1 bittir (Bir bloktaki iki kelimeyi birbirinden ayırabilmek için sadece 1 bite ihtiyaç duyarız); blok alanı 2 bittir (ana hafızada 4 blok vardır ve her bloğu eşsiz bir şekilde tanımlayabilmek için 2 bite ihtiyaç vardır); son olarak etiket alanı 1 bittir (kalan kısım).

Ana hafıza adresi Şekil 6.6’da gösterilen alanlara ayrılır.



Şekil 6.6: 16 Kelimelik Bir Hafıza İçin Ana Hafıza Adresi Biçimi

Ana hafıza adresi 9’u oluşturmaya çalıştığımızı varsayalım. Yukarıdaki eşleme tablosunda gördüğümüz gibi, adres 9 ana hafızada blok 4’te bulunur ve önbellekte blok 0’a eşlenmelidir (yani ana hafızada blok 4’ün içeriği önbellekte blok 0’a kopyalanmalıdır). Bilgisayar önbellek eşleme bloğunu belirlemek için güncel ana hafıza adresini kullanır. Bu adres ikili olarak Şekil 6.7’de gösterilmiştir.

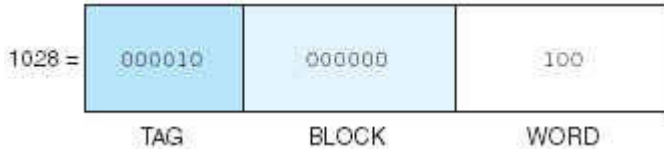


Şekil 6.7: Ana Hafıza Adresi $9 = 1001_2$ ’nin Alanları

İşlemci bu adresi oluştururken, 00 blok alanı bitlerini alır ve önbellekteki belirli bloğa yönlendirmek için kullanır. 00 önbellekte blok 0’ın kontrol edilmesi gerektiğini ifade eder. Eğer önbellek bloğu geçerliyse, önbellekte blok 0 ile ortak olan etiketle ana hafıza adresindeki etiket alanının 1 değeri karşılaştırılır. Eğer önbellek etiketi 1 ise, blok 4 halen önbellekte blok 0’a bağlıdır. (Bunu anlayabilmek için, blok 4’teki ana hafıza adresi $9 = 1001_2$ ile blok 0’daki ana hafıza adresi $1 = 0001_2$ karşılaştırılır. Bu iki adres sadece en sağdaki bitleri ile birbirinden ayrılır. Bu bit önbellek tarafından etiket olarak kullanılır.) Etiketlerin aynı olduğunu varsayarsak, bu durumda ana hafızadaki blok 4 önbellekte blok 0’da bulunur, kelime alanının 1 değeri blokta bulunan iki kelimenin birini seçmek için kullanılır. Bit 1 olduğundan, görelşi konumu (offset) 1 olan kelime seçilir. Böylece ana hafıza adresi 9’dan kopyalanan veri okunmuş olur.

Bu bağlamda bir örnek daha yapalım. İşlemci bu defa adres $4 = 0100_2$ ’ı oluştursun. Ortadaki iki bit (10) aramayı önbellekte blok 2’ye yönlendirir. Eğer blok geçerliyse, en soldaki etiket biti (0) önbellek bloğuyla birlikte depolanmış olan etiket biti ile karşılaştırılır. Eğer eşleşirlerse, bu bloktaki ilk kelime (offset 0) işlemciye döndürülür. Bu işlemi daha iyi anlamak için ana hafıza adresi $12 = 1100_2$ ile benzer bir alıştırma yapın.

Şimdi daha büyük bir örneğe bakalım. 14 bitlik ana hafıza adresi ve 64 blokluk önbellek kullanan bir sistemimiz olduğunu varsayın. Eğer her blok 8 kelime içerirse, 15 bitlik ana hafıza adresi 3 bit kelime alanı, 6 bit blok alanı ve 6 bit etiket alanı şeklinde bölünmüştür. Eğer işlemci ana hafıza adresini oluşturursa, önbellekteki blok 0’a bakacak, 000010 etiketini bulursa bu bloкта offset 4’teki kelimeyi işlemciye döndürür.



Tamamen İlişkili Önbellek

Doğrudan eşlenen önbellek diğer önbellekler kadar pahalı değildir, çünkü eşleme işlemi arama yapmaya gerek yoktur. Her ana hafıza bloğunun önbellekte eşlendiği özel bir konum vardır. Bir ana hafıza adresi önbellek adresine çevrildiğinde, işlemci bu hafıza bloğu için önbellekte nereye bakacağını bilir: Blok alanındaki bitleri inceler. Bu adres defterinize benzemektedir: Sayfaların genellikle bir alfabetik dizini vardır, böylece “Joe Smith”i aradığımızda “s” bölümüne bakarsınız.

Her ana hafıza bloğu için eşsiz bir konum belirlemek yerine tam tersini yaparız: Ana hafıza bloklarının önbellekte herhangi bir konuma yerleştirilmesine izin veririz. Bu şekilde eşlenen bir bloğu bulmanın tek yolu tüm önbelleği aramaktır. Bunun için de tüm önbelleğin ilişkili hafızadan yapılmış olması gerekir, bu sayede paralel olarak arama yapılabilir. Diğer bir deyişle, sadece bir arama ile istenen veri bloğunun önbellekte olup olmadığını anlamak için istenen etiket ile önbellekteki tüm etiketleri karşılaştırmalıdır. İlişkili hafızanın ilişkili aramaya izin verebilmesi için özel donanıma ihtiyacı vardır, bu da oldukça pahalıdır.

İlişkili eşleme kullanılarak ana hafıza adresi iki parçaya bölünür: Etiket ve kelime. Örneğin, bir önceki hafıza konfigürasyonumuzu (2^{14} kelime, 16 blokluk önbellek, 8 kelimelik bloklar) kullandığımızda, Şekil 6.8’de gördüğümüz gibi, kelime alanı hala 3 bittir, etiket alanı ise 11 bittir. Bu etiket önbellekteki her blokla birlikte depolanmalıdır. Önbellek belirli bir ana hafıza bloğu için arandığında, ana hafıza adresinin etiket alanı önbellekteki tüm geçerli etiketlerle karşılaştırılır. Eğer eş bulunursa blok bulunmuş demektir. (Etiket eşsiz olarak bir ana hafıza bloğunu ifade etmektedir.) Eğer eş bulunamazsa, kaybımız var demektir (cache miss) ve blok ana hafızadan aktarılmalıdır.



Doğrudan eşleme ile, eğer yeni bir bloğun yerleştirilmesi gereken önbellek konumunda başka bir blok varsa, bu blok önbellekten silinir (modifiye edildiyse ana hafızaya geri yazılır, edilmediyse overwrite edilir). Tamamen ilişkili eşleme ile, önbellek dolu olduğunda hangi bloğu atmamız gerektiğine karar vermek için bir yerdeğişim algoritmasına ihtiyacımız vardır (buna kurban blok – victim block adını veriyoruz). Bunun için basit bir *ilk giren ilk çıkar* (first-in, first-out) algoritması ya da *en az kullanılan* (least recently used) algoritması kullanılabilir. Kullanılabilecek olan çok sayıda yerdeğişim algoritması vardır, bunlardan kısaca bahsedilmiştir.

Küme İlişkili Önbellek

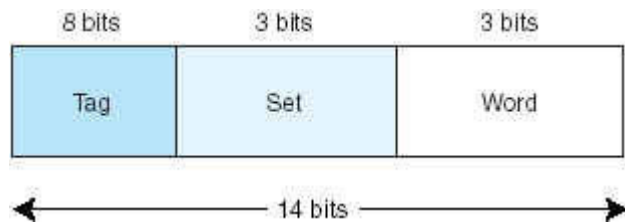
Hızlı ve karmaşık olduğundan dolayı ilişkili önbellek oldukça pahalıdır. Doğrudan eşleme ise pahalı olmamasına rağmen son derece kısıtlayıcı özelliklere sahiptir. Doğrudan eşlemenin önbellek kullanımını nasıl kısıtladığını görebilmek için, önceki örneğimizde açıkladığımız mimaride bir program çalıştırdığımızı düşünelim. 0 ve 16 blokları aynı konuma eşlenmiştir. Yani program tekrarlı bir şekilde 16'yı almak için 0'ı atar, sonraki aşamada da 0'ı almak için 16'yı atar ve önbellekte henüz kullanılmamış bloklar olmasına rağmen bu şekilde devam eder. Tamamen ilişkili önbellek ana hafızadaki bir bloğun herhangi bir konuma yerleştirilmesine izin verir. Ancak blokla birlikte depolamak için özel donanıma ek olarak daha büyük bir etikete ihtiyaç duyar (yani daha büyük ve daha pahalı bir önbellek). Bizim ise bunların arasında olan optimum bir düzene ihtiyacımız vardır.

Üçüncü eşleme planımız bu iki planının bir bileşimi olan *N yollu küme ilişkili önbellek eşleme*. Bu planda bloğu belirli bir önbellek konumuna eşlemek için adresi kullanırız, bu açıdan doğrudan eşlenen önbeleğe benzemektedir. Aralarındaki önemli fark ise şöyledir: Adres tek bir önbellek bloğuna eşlemek yerine birçok önbellek bloğunu içeren bir kümeye eşlenir. Önbellekteki tüm kümeler aynı boyutta olmalıdır. Bu boyut önbellekten önbeleğe farklı olabilir. Örneğin, Şekil 6.9'da görüldüğü gibi, bir 2 yollu küme ilişkili önbellekte küme başına iki önbellek olsun. Bu şekilde 0'ın iki blok içerdiğini görüyoruz, biri geçerlidir ve A, B, C, ... verilerini tutar, diğeri de geçersizdir. Aynısı Set 1 için de geçerlidir. Set 2 ve Set 3 aynı zamanda iki blok tutabilir, ancak şimdilik her kümede sadece ikinci bloklar geçerlidir. Bir 8 yollu küme ilişkili önbellekte, küme başına 8 önbellek bloğu vardır. Doğrudan eşlenen önbellek, küme boyutunun 1 olan N yollu küme ilişkili önbelleğin bir özel durumudur.

Set	Tag	Block 0 of set	Valid	Tag	Block 1 of set	Valid
0	00000000	Words A, B, C, ...	1	-----		0
1	11110101	Words L, M, N, ...	1	-----		0
2	-----		0	10111011	P, Q, R, ...	1
3	-----		0	11111100	T, U, V, ...	1

Şekil 6.9: İki Yollu Küme İlişkili Önbellek

Küme ilişkili önbellek eşlemede, ana hafıza adresi üç parçaya bölünür: Etiket alanı, küme alanı ve kelime alanı. Etiket ve kelime alanları daha önceki görevlerinin aynılarını üstlenir, küme alanı ise ana hafızanın eşlendiği önbellek kümelerini gösterir. 2^{14} kelimelik ana hafıza ve her bloğu 8 kelime içeren 16 blokluk önbellekle birlikte 2 yollu küme ilişkili eşleme kullandığımızı varsayın. Eğer önbellek toplam 16bloktan oluşursa ve her kümede 2 blok varsa, önbellekte 8 kelime var demektir. Bundan dolayı, küme alanı 3 bit, kelime alanı 3 bit ve etiket alanı 8 bittir. Bu konu Şekil 6.10'da gösterilmiştir.



Şekil 6.10: Küme İlişkili Eşleme İçin Biçim

6.4.2 Yerdeğişim Planları (Replacement Policies)

Doğrudan eşlenen önbellekte, eğer bir önbellek bloğu için çekişme varsa yapılabilecek tek şey vardır: Varolan blok yeni bloğa yer açmak için önbellekten atılır. Bu işleme yerdeğişim denir. Doğrudan eşlemede bir yerdeğişim planına ihtiyaç yoktur, çünkü her yeni blok için gerekli konumlar önceden belirlenir. Ancak tamamen ilişkili önbellek ile küme ilişkili önbellekte, atılacak olan kurban bloğu belirlemek için yerdeğişim algoritmasına ihtiyacımız vardır. Tamamen ilişkili önbelleği kullanırken, verilen ana hafıza bloğunun eşlenebileceği K tane muhtemel önbellek konumu vardır (K =önbellekteki blok sayısı). N yollu küme ilişkili eşlemede, bir blok verilen bir küme içindeki N farklı bloktan herhangi birine (birilerine) eşlenebilir. Önbellekte hangi blokla yer değişimi yapacağımızı nasıl bileceğiz? Yer değişimini belirleyen algoritmaya *yerdeğişim planı* denir.

Birçok popüler yerdeğişim planı vardır. *Optimal* algoritması, pratik olmamasına rağmen diğer tüm algoritmaları ölçmek için benchmark olarak kullanılabilir. Yakında tekrar ihtiyaç duyacağımız verileri önbellekte tutmak isteriz ve tekrar ihtiyaç duymayacaklarımızı da önbellekten atarız. Bu iki kritere göre çalışan ve geleceğe bakacak hangi blokların tutulacağını, hangilerinin ise atılacağını kesin olarak belirleyen bir algoritma en iyisi olurdu. *Optimal* algoritmasının yaptığı da tam olarak budur. Gelecekte en uzun süre kullanılmayacak olan bloğu yer değiştirmek isteriz. Örneğin, kurban blok olarak seçilecek blok, blok 0 ile blok 1 arasından seçileceğini, blok 0'ın 5 saniye içinde tekrar kullanılacağını, blok 1'in ise 10 saniye içinde kullanılmayacağını varsayalım. Blok 1'i atarız. Pratik bir bakış açısına göre, geleceği göremeyiz, ancak bir program çalıştırabilir sonra durdurup tekrar çalıştırabiliriz. Böylece efektif bir şekilde geleceği görürüz. İkinci defada *optimal* algoritmasını çalıştırırız. *Optimal* algoritması mümkün olan en düşük kayıp oranını garanti eder. Çalıştırdığımız her programın geleceğini göremediğimizden, *optimal* algoritması diğer algoritmaların ne kadar iyi ya da kötü olduğunu belirlemek için sadece bir ölçüdür. *Optimal* algoritmasına en yakın sonuçları veren algoritma diğerlerinden daha iyi sayılır.

Optimal algoritmasına en çok yaklaşan algoritmalara ihtiyaç duyarız. Bu aşamada birçok seçeneğimiz vardır. Örneğin, geçici konumu ele alabiliriz (temporal locality). Yakın geçmişte kullanılmamış olan bir değer, yakında tekrar kullanılıp kullanılmayacağını tahmin edebiliriz. Her bloğun en son erişildiği zamanları kaydedebilir ve en az kullanılan bloğu kurban blok olarak ilan edebiliriz. Bu *en az kullanılan* (LRU - *least recently used*) algoritmasıdır. Ne yazık ki LRU her önbellek bloğu için sistemde bir geçmiş tutulmasını gerektirir. Bu da önemli bir alan demektir ve önbelleğin çalışmasını yavaşlatır.

İlk giren ilk çıkar (*First in, first out - FIFO*) diğer bir popüler yaklaşımdır. Bu algoritma ile önbellekte en uzun süre bulunan blok kurban blok olarak seçilir ve önbellekten atılır.

Diğer bir yaklaşım ise kurban bloğu rastgele seçmektir. LRU ve FIFO'nun sorunu yanlış gösterilen referanstan dolayı sürekli olarak bir bloğun atılması, yeniden alınması, sonra tekrar atılması ve yeniden alınmasıdır. Bazı insanlar rastgele yerdeğişimin bazen yakında kullanılabilir veriyi atmasına rağmen bu soruna asla yol açmayacağını düşünür. Ancak gerçekten rastgele yer değişimi yapmak zordur ve genel performansı düşürür.

Seçilen algoritma genellikle sistemin nasıl kullanılacağına bağlıdır. Tüm senaryolar için en iyi olan bir algoritma yoktur. Bu nedenle tasarımcılar çok çeşitli durumlarda iyi sonuçlar veren algoritmaları kullanır.