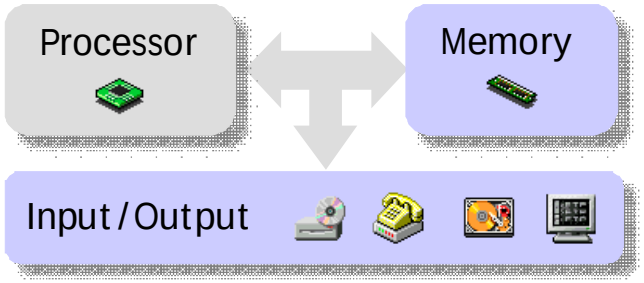
فصل هفتم

مقدمه‌ای بر حافظه نهان

در فصل پایپلاین یاد گرفتیم که چطور یک پردازنده سریعتر طراحی کنیم. حال سؤال این است که چگونه می-توان با فراهم کردن داده کافی همیشه پردازنده را مشغول نگه داشت.
در این فصل بر روی حافظه تمرکز خواهیم نمود که معمولاً به عنوان یک گلوگاه، کارآیی سیستم را محدود می-نماید.
نحوه اتصال پردازنده، حافظه و سیستم‌های ورودی و خروجی در شکل۱ نشان داده شده است.



شکل۱: نحوه اتصال اجزای یک کامپیوتر

ما به سؤالات زیر پاسخ خواهیم داد:

- چالش‌های موجود در ساخت یک سیستم حافظه بزرگ و سریع کدامند؟
- حافظه نهان یا حافظه کش (حافظه سریع) چیست؟
- چرا حافظه نهان مشکل را حل می‌کند؟
- حافظه نهان چگونه سازماندهی می‌گردد؟ به عبارتی ما چگونه داده‌ها را جاگذاری نموده و چگونه آنها را پیدا می‌کنیم؟

کامپیوترهای امروزی به سیستم‌های حافظه بزرگ و سریع نیاز دارند

- ظرفیت حافظه‌های بزرگ برای اکثر برنامه‌های پایگاه داده، محاسبات علمی با مجموعه داده‌های بزرگ، ویدیو، موسیقی و غیره مورد نیاز می‌باشد.
- برای اینکه پایپلاین پردازنده با کارآیی بالایی بتواند به فعالیت خود ادامه دهد، در این صورت بهتر است که دسترسی به هر دو حافظه داده‌ها و دستورات در یک کلاک انجام گیرد. هنگامی که پردازنده superscalar باشد مشکلات سرعت دسترسی به حافظه بیشتر هم خواهد شد.

متأسفانه ما باید مابین سرعت، هزینه و ظرفیت موازنه برقرار کنیم. موازنه‌های صورت گرفته برای سه نوع حافظه در جدول ۱ نشان داده شده است.

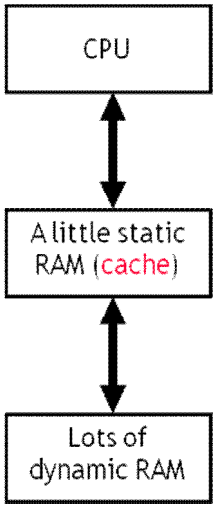
جدول ۱: موازنه بین سرعت، هزینه و ظرفیت برای سه نوع حافظه

حافظه	سرعت	هزینه	ظرفیت
Static RAM	سریعترین	گران	کوچکترین
Dynamic RAM	کند	ارزان	بزرگ
Hard disks	کندترین	ارزانترین	بزرگترین

- خرید زیاد حافظه سریع برای خیلی‌ها بسیار گران است.
- اما حافظه دینامیک نسبت به بقیه واحدهای موجود در مسیر داده، تأخیر بسیار بیشتری دارد. اگر هر دستور lw یا sw به حافظه دینامیک دستیابی داشته باشد، در این صورت یا باید پریود کلاک را افزایش دهیم و یا اینکه پایپلاین خود را یک یا چند کلاک متوقف کنیم (stall دادن) تا داده حافظه آماده گردد.

معرفی حافظه نهان

به نظر می‌رسد که اگر ما بتوانیم یک بالانس بین حافظه سریع و ارزان پیدا کنیم، در این صورت به یک راهکار مناسب دست پیدا خواهیم کرد. ما این کار را با استفاده از حافظه نهان که یک حافظه کوچک، سریع و گران می‌باشد انجام می‌دهیم. جایگاه حافظه نهان در یک سیستم کامپیوتری در شکل ۲ نشان داده شده است.



شکل ۲: جایگاه حافظه نهان

- حافظه نهان مابین پروسسور و حافظه اصلی سرعت پایین که از نوع دینامیک است قرار می‌گیرد.

- حافظه نهان یک کپی از داده‌های پر استفاده حافظه اصلی را در داخل خود نگهداری می‌کند.

با استفاده از حافظه نهان سرعت دسترسی به حافظه در مدت زمان کلی اجرای برنامه افزایش پیدا می‌کند و دلیل این امر هم این است که ما موارد پر استفاده را سریعتر کرده‌ایم (قانون امدال).

- خواندن‌ها و نوشتن‌ها به آدرس‌های پراستفاده حافظه اصلی، با استفاده از حافظه نهان سرویس داده می‌شود.
- ما فقط برای داده‌های کمتر استفاده شده به حافظه اصلی مراجعه می‌کنیم.

مفهوم محلیت

معمولاً پیدا کردن داده‌های پر استفاده قبل از اجرای واقعی یک برنامه مشکل و حتی غیر ممکن می‌باشد. این مسأله فهمیدن اینکه چه داده‌هایی را باید در داخل حافظه نهان کوچک و سریع قرار داد مشکل می‌نماید. اما در عمل برنامه‌ها دارای رفتار محلیت می‌باشند که حافظه نهان می‌تواند از آن استفاده نماید.

- اصل محلیت زمانی (temporal locality) می‌گوید که وقتی یک برنامه به یک آدرس حافظه مراجعه می‌کند، به احتمال زیاد آن برنامه به همان آدرس مجدداً مراجعه خواهد نمود.
- اصل محلیت مکانی (spatial locality) می‌گوید که اگر یک برنامه به یک آدرس حافظه مراجعه می‌کند، به احتمال زیاد آن برنامه به آدرس‌های نزدیک آن آدرس نیز مراجعه خواهد نمود.

محلیت زمانی در فضای دستورات

حلقه‌ها مثال‌های خوبی برای محلیت زمانی می‌باشند. بدنه یک حلقه ممکن است چندین بار اجرا گردد که در این صورت آن چند آدرس از حافظه دستورات چندین بار مرتباً مورد دسترسی قرار خواهد گرفت. مثال زیر را در نظر بگیرید.

```
Loop:  lw     $t0, 0($s1)
       add    $t0, $t0, $s2
       sw     $t0, 0($s1)
       addi   $s1, $s1, -4
       bne    $s1, $0, Loop
```

در این مثال، هر کدام از دستورات به طور مثال lw چندین بار اجرا خواهد شد یعنی پردازنده به آدرس دستور lw چندین بار مراجعه خواهد نمود. بر طبق محلیت زمانی، اگر دستور lw در حال اجرا باشد، این دستور به احتمال زیاد در آینده‌ای نزدیک هم اجرا خواهد شد.

محلیت زمانی در فضای داده

برنامه‌ها اغلب به داده‌های یکسانی بارها و بارها مراجعه می‌کنند (بویژه داخل حلقه‌ها). در مثال زیر متغیرهای i و sum مرتباً خوانده و نوشته می‌شوند.

```
sum = 0;
for (i = 0; i < MAX; i++)
```

```
    sum = sum + f(i);
```

متغیرهای پر استفاده می‌توانند در داخل رجیسترها قرار داده شوند ولی این کار به دلایل زیر همیشه امکان پذیر نیست:

- تعداد رجیسترها محدود است.
- نمی‌توان آدرس یک رجیستر را نگهداری کرد (یعنی یک اشاره‌گر به آن ایجاد نمود).
- شرایطی وجود دارد که مجبور هستیم متغیرها را در حافظه نگهداری کنیم که به اشتراک گذاری حافظه و اختصاص حافظه به صورت دینامیکی مثال‌هایی از این دست می‌باشند.

محلیت مکانی در برنامه‌ها

مفهوم محلیت مکانی می‌گوید که اگر یک برنامه به یک آدرس دسترسی پیدا کند، در این صورت به احتمال زیاد در آینده به آدرس‌های نزدیک آن آدرس نیز دسترسی خواهد داشت. به مثال زیر توجه کنید.

```
sub     $sp, $sp, 16
sw      $ra, 0($sp)
sw      $s0, 4($sp)
sw      $a0, 8($sp)
sw      $a1, 12($sp)
```

در این مثال بعد از اجرای دستور sub نزدیکترین دستور به آن که اولین sw بعد از sub می‌باشد اجرا خواهد شد. پس از اولین sw دستور بعد از آن اجرا خواهد شد و بقیه هم به همین ترتیب. تقریباً همه برنامه‌ها رفتار محلیت مکانی را از خود بروز می‌دهند، دلیل این کار این است که دستورات یک برنامه معمولاً به صورت ترتیبی اجرا می‌شوند. اگر ما دستوری را که در آدرس i حافظه قرار دارد اجرا کنیم، در این صورت احتمالاً در آینده دستور بعدی را که در آدرس i+1 قرار دارد را نیز اجرا خواهیم نمود.

توجه: قطعه کدهایی نظیر حلقه‌ها، هم رفتار محلیت مکانی و هم رفتار محلیت زمانی را از خود نشان می‌دهند.

محلیت مکانی در فضای داده‌ها

برنامه‌ها اغلب به داده‌هایی مراجعه می‌کنند که به صورت پشت سر هم در حافظه ذخیره شده‌اند.

- عناصر آرایه نظیر آرایه a در کد زیر به صورت پشت سر هم در حافظه ذخیره می‌گردند

```
sum = 0;
for (i = 0; i < MAX; i++)
    sum = sum + a[i];
```

- همچنین هر کدام از فیلدهای یک رکورد یا یک شیء نظیر employee در کد زیر، به صورت پشت سرهم در حافظه قرار می‌گیرند.

```
employee.name = “Homer Simpson”;
employee.boss = “Mr. Burns”;
employee.age = 45;
```

توجه: داده‌های یک برنامه همانند دستورات آن، هم می‌توانند محلیت زمانی داشته باشند و هم محلیت مکانی.

حافظه نهان چگونه از مفهوم محلیت زمانی استفاده می‌کند؟

اولین مرتبه که پردازنده از یک آدرس در حافظه اصلی عملیات خواندن را انجام می‌دهد، یک کپی از آن داده در داخل حافظه نهان نیز گذاشته می‌شود. در زمانهای آینده که آن آدرس خوانده می‌شود، ما می‌توانیم به جای اینکه مجدداً به حافظه اصلی (که از جنس حافظه دینامیک بوده و سرعت آن پایین است) مراجعه کنیم از نسخه کپی شده داده در حافظه نهان استفاده نمائیم. بنابراین، خواندن اول به دلیل اینکه هم به حافظه اصلی مراجعه می‌کند و هم به حافظه نهان، مقداری کندتر از قبل انجام می‌گیرد، اما خواندن‌های بعدی به دلیل اینکه فقط به حافظه نهان مراجعه می‌کنند، بسیار سریعتر انجام می‌گیرد.

فرآیند فوق جهت خواندن از حافظه، از مفهوم محلیت زمانی استفاده می‌نماید به دلیل اینکه داده‌هایی که زیاد مورد دستیابی قرار می‌گیرند، در یک حافظه نهان سریع قرار داده می‌شوند.

حافظه نهان چگونه از مفهوم محلیت مکانی استفاده می‌کند؟

موقعیکه پردازنده آدرس i از حافظه اصلی را می‌خواند، یک کپی از آن داده در حافظه نهان قرار داده می‌شود. اما به جای اینکه فقط داده مربوط به آدرس i را داخل حافظه نهان کپی کنیم، ما می‌توانیم چندین مقدار را در یک لحظه به داخل حافظه نهان کپی کنیم. به طور مثال ما می‌توانیم داده‌های مربوط به آدرس‌های i تا i+3 را داخل حافظه نهان کپی کنیم. در این صورت، اگر پردازنده در آینده بخواهد از آدرس‌های i+1 ، i+2 و i+3 عملیات خواندن را انجام دهد، می‌تواند به جای مراجعه به حافظه اصلی سرعت پایین، به حافظه نهان مراجعه نموده و داده‌های خود را از داخل آن بخواند. به طور مثال، به جای خواندن فقط یک عنصر آرایه در یک زمان و کپی آن به داخل حافظه نهان، می‌توان چهار عنصر پشت سر هم آرایه را به داخل حافظه نهان کپی نمود.

در واقع در فرآیند فوق، ما برای خواندن‌های اولیه پنالتی کارآیی می‌دهیم، اما چون پردازنده در آینده به داده-های بیشتری نیاز خواهد داشت و برنامه‌ها دارای رفتار محلیت مکانی می‌باشند، امیدواریم که این پنالتی‌ها نه تنها جبران خواهند شد، بلکه به کارآیی‌های خیلی بالاتری نسبت به حالت عدم استفاده از حافظه نهان دست پیدا خواهیم کرد.

موارد دیگر استفاده از کش

ایده حافظه نهان در موارد متعددی مورد استفاده قرار گرفته است که از آن جمله می‌توان به موارد زیر اشاره نمود:

- شبکه‌های کامپیوتری احتمالاً بهترین مثال می‌باشند. شبکه‌ها معمولاً دارای تأخیر بالا و پهنای باند پایین می‌باشند، به همین دلیل فرستادن داده‌های تکراری خیلی جالب به نظر نمی‌رسد. مرورگرهای وب نظیر Firefox و IE صفحات اخیراً مشاهده شده شما را بر روی هارد دیسک ذخیره می‌نمایند تا در صورتی که مجدداً به آنها نیاز پیدا کردید، دیگر آنها را از روی شبکه بارگذاری نکند و به این ترتیب سرعت افزایش پیدا کند.

- پردازنده‌هایی که از تکنولوژی حافظه مجازی استفاده می‌کنند، نیاز به ترجمه آدرس دارند. از آنجا که فرآیند تبدیل کردن آدرس زمانبر است، تبدیل‌هایی که قبلاً انجام گرفته در داخل یک حافظه نهان ذخیره می‌گردد تا در صورت نیاز به تبدیل کردن همان آدرس در آینده، مقدار تبدیل شده از داخل حافظه نهان خوانده شود و دیگر تبدیل انجام نگیرد و به این ترتیب سرعت ترجمه را افزایش دهیم. این حافظه نهان که به منظور تبدیل کردن آدرس استفاده می‌شود، TLB[1] نام دارد.
- سیستم‌های عامل ممکن است بلاک‌های پر استفاده هارد دیسک را در داخل حافظه اصلی نگهداری کنند که در این حالت حافظه اصلی به عنوان نوعی حافظه نهان برای هارد دیسک عمل می‌کند. این بلاک‌های پر استفاده ممکن است برای دسترسی‌های سریعتر، داخل حافظه نهان پردازنده نیز ذخیره گردند.

تعاریف حافظه نهان

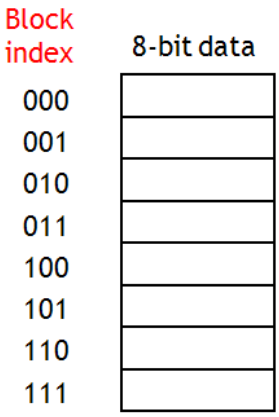
- **برخورد حافظه نهان:** اگر حافظه نهان حاوی داده‌ای باشد که ما به دنبال آن می‌گردیم، اصطلاحاً گفته می‌شود که یک برخورد[2] اتفاق افتاده است. اگر تعداد برخوردها زیاد باشد، خیلی خوب خواهد بود. به دلیل اینکه حافظه نهان می‌تواند اطلاعات را خیلی سریعتر از حافظه اصلی در اختیار قرار دهد.
- **عدم برخورد حافظه نهان:** اگر حافظه نهان داده مورد تقاضا را نداشته باشد، در این صورت گفته می‌شود که یک عدم برخورد[3] رخ داده است. عدم برخورد جالب نیست، به دلیل اینکه پردازنده باید منتظر حافظه اصلی بماند که با سرعت پایین‌تری اطلاعات را در اختیار قرار می‌دهد.
- برای اندازه‌گیری کارآیی حافظه نهان، دو نوع روش اصلی مورد استفاده قرار می‌گیرد:
  - نرخ برخورد[4]: به درصد مراجعات موفق به حافظه نهان نسبت به کل مراجعات نرخ برخورد گفته می‌شود. نرخ برخورد را با h نشان داده و به صورت $h = \frac{\text{تعداد hit}}{\text{تعداد hit} + \text{تعداد miss}}$ هم تعریف می‌کنند.
  - نرخ عدم برخورد[5]: به درصد مراجعات ناموفق به کل مراجعات نرخ عدم برخورد گفته می‌شود و مقدار آن "1-h" می‌باشد.

معمولاً پردازنده‌ها دارای نرخ برخورد ۹۵ درصد یا بالاتر می‌باشند و به همین دلیل اکثر دسترسی‌های حافظه از طریق حافظه نهان صورت می‌گیرد و بنابراین به طور باور نکردنی سرعت مراجعات به حافظه بالاست.

[1] - Translation Lookaside Buffer
[2] - Hit
[3] - Miss
[4] - Hit rate
[5] - Miss rate

یک طراحی ساده از حافظه نهان

حافظه نهان معمولاً به بلوک‌هایی تقسیم بندی می‌گردد. تعداد بلوک‌ها اغلب توانی از ۲ می‌باشد. در حال حاضر فرض ما بر این است که هر بلوک حافظه نهان دارای یک بایت بوده و به همین دلیل از خاصیت محلیت مکانی استفاده نمی‌کند. در آینده این خاصیت را هم به آن اضافه خواهیم نمود. در شکل ۳ یک حافظه نهان با ۸ بلوک که هر بلوک شامل یک بایت است نشان داده شده است.



شکل ۳: یک حافظه نهان با هشت بلوک که هر بلوک آن شامل یک بایت است

چهار سؤال مهم:

۱- ما می‌توانیم یک بلوک از اطلاعات را از حافظه اصلی به داخل حافظه نهان کپی کنیم، دقیقاً باید آن بلوک را به کجا کپی کنیم؟

۲- چگونه ما می‌توانیم بگوییم که آیا یک کلمه هم اکنون داخل حافظه نهان قرار دارد و یا اینکه مرتبه اول خواندن آن کلمه است و حتماً باید آن را از حافظه اصلی بخوانیم؟ به اولین بارگذاری داده‌ها در حافظه نهان، جایگذاری[1] گفته می‌شود.

۳- بعد از مدتی حافظه نهان ما که اندازه کوچکی دارد پر می‌گردد. برای بارگذاری یک بلوک جدید از حافظه اصلی به داخل حافظه نهان، ما باید یکی از بلوک‌های موجود در حافظه نهان را با بلوک جدید جایگزین[2] کنیم. سؤال این است که کدام بلوک حافظه نهان باید جایگزین شود؟

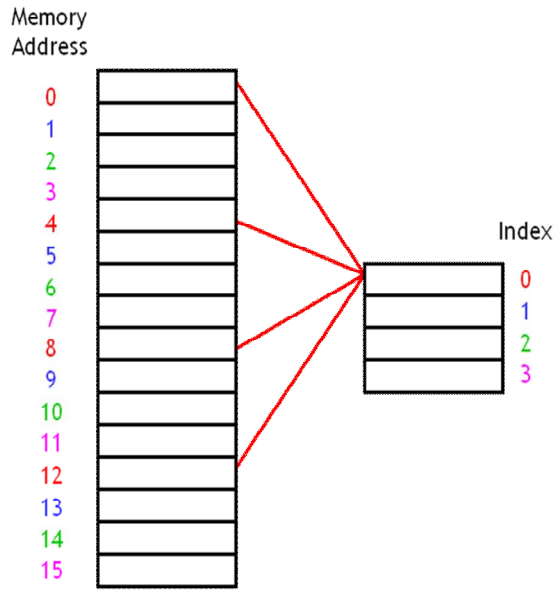
۴- عملیات نوشتن توسط سیستم حافظه چگونه انجام می‌گیرد؟

سؤالات ۱ و ۲ به هم وابسته‌اند، ما برای اینکه در آینده بتوانیم اطلاعات را در داخل حافظه نهان پیدا کنیم، باید بدانیم که آنها را کجا باید قرار دهیم!

بهتر است اطلاعات را کجای حافظه نهان قرار دهیم؟

[1] - Placement
[2] - Replacement

حافظه نهان نگاشت-مستقیم یک راهکار ساده است. در این روش هر بلوک از حافظه اصلی به یک بلوک از حافظه نهان نگاشته می‌شود. نگاشت مستقیم در شکل ۴ نشان داده شده است.



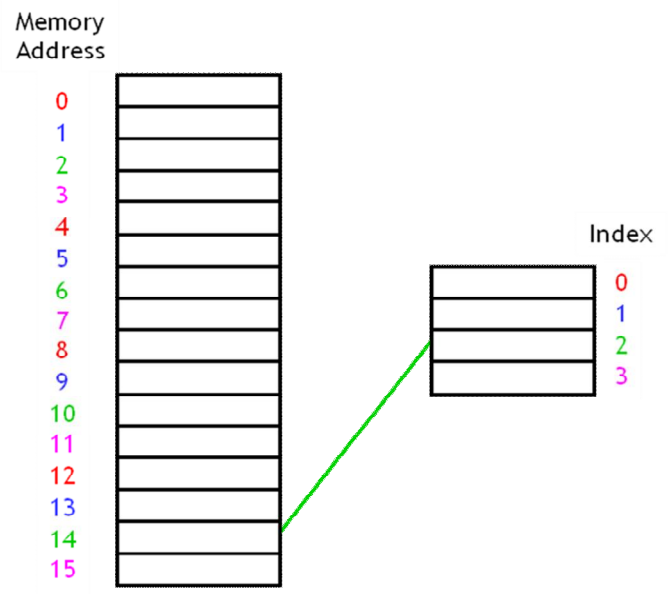
شکل ۴: نگاشت مستقیم آدرس‌های حافظه اصلی به خانه‌های حافظه نهان

در این شکل یک حافظه اصلی ۱۶ بایتی و یک حافظه نهان ۴ بایتی مشاهده می‌شود. در اینجا فرض شده که هر بلوک شامل یک بایت می‌باشد. در این شکل آدرس‌های 0، 4، 8 و 12 به آدرس 0 حافظه نهان و آدرس‌های 1، 5، 9 و 13 به آدرس 1 حافظه نهان نگاشته شده‌اند و به همین ترتیب بقیه نگاشت‌ها هم انجام شده است.

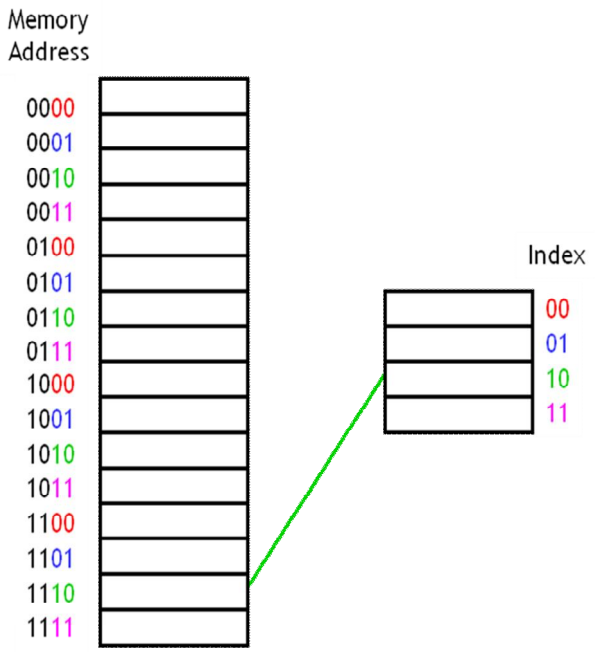
یکی از روش‌هایی که می‌توان مشخص کرد که یک آدرس از حافظه اصلی به چه آدرسی از حافظه نهان نگاشته می‌شود، استفاده از عملگر mod یا باقیمانده است. اگر حافظه نهان شامل 2^k خانه باشد، در این صورت در نگاشت مستقیم، داده موجود در آدرس "i" از حافظه اصلی به آدرس "i mod k" از حافظه نهان نگاشته می‌شود. به طور مثال، آدرس 14 در اینجا به آدرس 2 نگاشته می‌شود چونکه "14 mod 4=2". نحوه کار در شکل ۵ نشان داده شده است.

روش دیگری که جهت پیدا کردن خانه‌ای از حافظه نهان برای قرار دادن اطلاعات یک آدرس از حافظه اصلی مورد استفاده قرار می‌گیرد و معادل همان روش قبلی است، استفاده از k بیت کم ارزش آدرس در نمایش باینری می‌باشد. در این روش k بیت کم ارزش از آدرس هر خانه حافظه اصلی هر عددی را نشان دهند، آن آدرس به خانه‌ای از حافظه نهان نگاشته خواهد شد که آدرس آن، عدد مذکور باشد. به طور مثال شکل ۶ را در نظر بگیرید. همان طور که در این شکل مشاهده می‌شود، دو بیت کم ارزش آدرس 14 مساوی 10 می‌باشد ($(1110)_2$=14)، بنابراین آدرس 14 از حافظه اصلی به آدرس 2 از حافظه نهان نگاشته می‌شود ($(10)_2$=2).

توجه: k بیت کم ارزش از نمایش باینری یک عدد (i) همان باقیمانده تقسیم آن عدد به 2^k می‌باشد ($i \bmod 2^k$).



شکل ۵: نحوه نگاشت آدرس 14 حافظه اصلی در روش نگاشت مستقیم



شکل ۶: استفاده از بیت‌های کم ارزش آدرس جهت پیدا کردن خانه حافظه نهان برای نگاشت آدرس 14.

نحوه پیدا کردن اطلاعات در داخل حافظه نهان

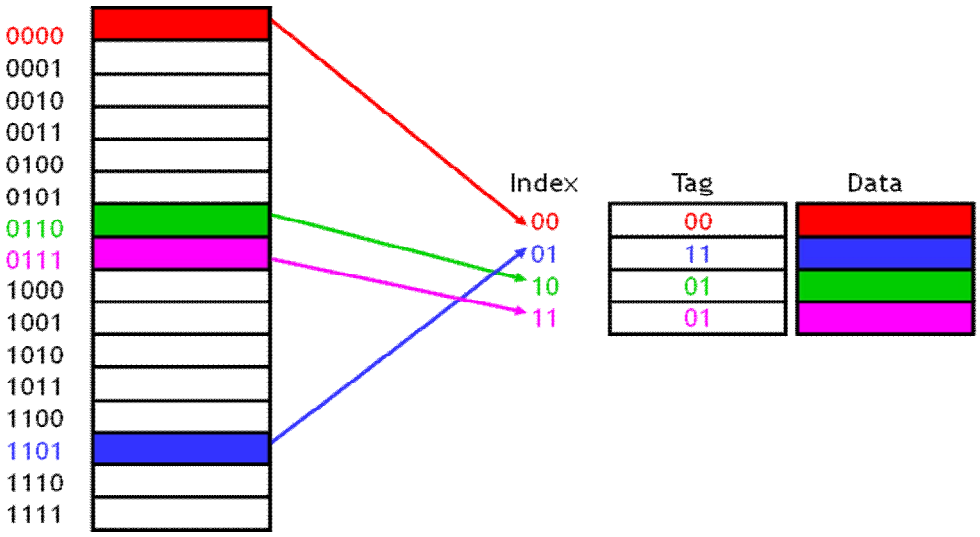
سؤال دوم این بود که چگونه مشخص کنیم داده‌ای که دنبال آن هستیم، هم اکنون داخل حافظه نهان قرار دارد یا نه. اگر ما بخواهیم آدرس i از حافظه اصلی را بخوانیم، می‌توانیم از عملگر mod جهت پیدا کردن خانه‌ای از حافظه نهان که احتمالاً داده آدرس i داخل آن ذخیره شده است، استفاده کنیم. مشکلی که وجود دارد این است

که ممکن است آدرس‌های دیگری نیز به آن خانه حافظه نهان نگاشته شوند. به طور مثال در خانه 2 حافظه نهان، هر یک از آدرس‌های 2، 6، 10 و 14 می‌توانند ذخیره گردند. حال سؤال این است که از کجا تشخیص دهیم که داده ذخیره شده در خانه 2 حافظه نهان متعلق به کدام یک از آدرس‌های ممکن می‌باشد؟

راهکار مشکل فوق این است که در داخل هر خانه حافظه نهان غیر از داده، یک مقدار دیگر با نام تگ[1] نگهداری کنیم. تگ بیت‌های باقیمانده آدرس را نگهداری می‌کند و این امکان را فراهم می‌آورد که بتوانیم بین آدرس‌های مختلفی که می‌توانند به یک خانه از حافظه نهان نگاشت شوند تمایز قائل شویم. در این صورت با بررسی بیت-های تگ خواهیم فهمید که از بین همه آدرس‌هایی که می‌توانند به یک خانه از خافظه نهان نگاشت شوند، هم اکنون کدامیک از آنها نگاشت شده است. در واقع، هر خانه از حافظه نهان دارای دو بخش است: بخش داده و بخش تگ. بخش داده، داده مربوط به آدرس نگاشته شده به آن خانه را نگهداری می‌کند و بخش تگ، قسمتی از آدرس را که برای آدرس دهی حافظه نهان استفاده نشده است.

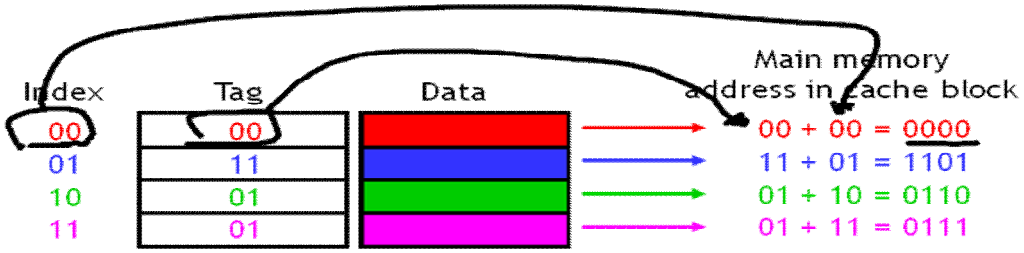
شکل ۷ را در نظر بگیرید. در این شکل به طور مثال در آدرس 3 حافظه نهان هم اکنون داده آدرس 7 حافظه اصلی ذخیره شده است. با کنار گذاشتن دو بیت کم ارزش از آدرس چهار بیتی حافظه اصلی برای آدرس 7 (0111) دو بیت 01 باقی می‌ماند که آن هم در بخش تگ حافظه نهان ذخیره شده است.

با استفاده از میدان تگ می‌توان فهمید که هم اکنون در داخل یک خانه از حافظه نهان، دقیقاً اطلاعات کدام آدرس حافظه اصلی نگهداری می‌گردد. این آدرس از حافظه اصلی با کنار هم قرار دادن یا اتصال[2] بخش تگ با مقدار اندیس[3] آن خانه از حافظه نهان مطابق شکل ۸ بدست می‌آید.



شکل ۷: استفاده از تگ در داخل حافظه نهان جهت تشخیص آدرس نگاشت شده

[1] - Tag
[2] - Concatenation
[3] - Index

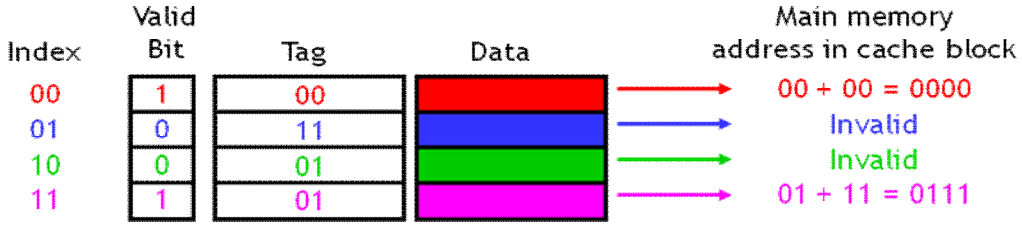


شکل ۸: نحوه بدست آوردن آدرسی که اطلاعات آن داخل یک خانه از حافظه نهان نگهداری می‌شود.

بیت اعتبار

در ابتدای کار پردازنده (بعد از روشن شدن[1] یا بازنشانی[2] سیستم)، حافظه نهان خالی است و داده معتبری داخل آن نگهداری نمی‌شود. ما باید بتوانیم این وضعیت را تشخیص دهیم. این کار با اضافه کردن یک بیت با نام بیت اعتبار[3] به هر خانه حافظه نهان قابل انجام است. در ابتدای کار که سیستم مقداردهی اولیه[4] می‌گردد، همه بیت-های اعتبار با 0 مقداردهی می‌شوند. در ادامه کار، وقتی که یک داده در خانه مشخصی از حافظه نهان بارگذاری می‌شود، بیت اعتبار مربوط به آن خانه مساوی 1 می‌گردد. نحوه استفاده از بیت اعتبار جهت تشخیص معتبر بودن داده موجود در یک خانه از حافظه نهان در شکل ۹ نشان داده شده است.

نکته: حافظه نهان علاوه بر اینکه محتوی یک کپی از داده‌های داخل حافظه اصلی می‌باشد، دارای بیت‌هایی جهت پیدا کردن داده در داخل حافظه نهان و همچنین تعیین اعتبار داده داخل هر خانه حافظه نهان می‌باشد.



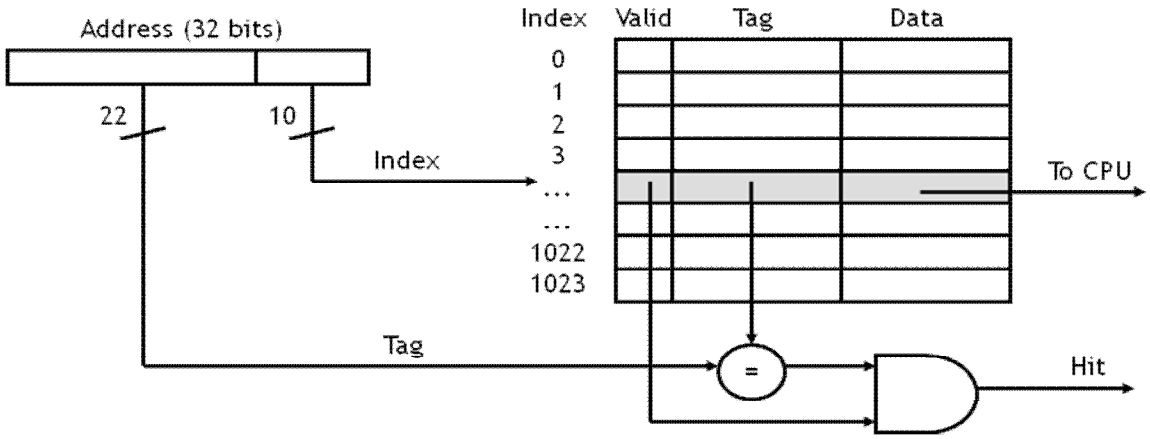
شکل ۹: نحوه استفاده از بیت اعتبار جهت تشخیص معتبر بودن داده داخل یک خانه از حافظه نهان

به هنگام مراجعه به حافظه چه اتفاقی می‌افتد؟

وقتی که CPU می‌خواهد از حافظه بخواند، آدرس ابتدا به مدار کنترل کننده حافظه نهان[5] تحویل داده می‌شود. k بیت کم ارزش آدرس به عنوان اندیس جهت آدرس دهی حافظه نهان مورد استفاده قرار می‌گیرد (k تعداد بیت اندیس حافظه نهان است). اگر خانه آدرس دهی شده حافظه نهان معتبر باشد و تگ موجود در آن خانه

[1] - Power on
[2] - Restart
[3] - Valid bit
[4] - Initialization
[5] - Cache controller

مساوی "m-k" بیت باقیمانده آدرس باشد (m تعداد بیت‌های آدرس تولید شده توسط پردازنده است و یا به عبارت بهتر تعداد بیت آدرس حافظه اصلی است)، در این صورت یک برخورد رخ داده و داده موجود در داخل حافظه نهان به پردازنده ارسال می‌گردد. فرآیند آدرس دهی حافظه نهان، مقایسه تگ و ارسال داده به پردازنده (در صورت مساوی بودن تگ تولید شده توسط پردازنده با تگ داخل حافظه نهان) در نشان شکل ۱۰ داده شده است. در این شکل تعداد بیت آدرس حافظه مساوی 32 و تعداد خانه‌های حافظه نهان مساوی 2^{10} می‌باشد، بنابراین m=32 و k=10 بوده و تعداد بیت تگ حافظه نهان مساوی m-k=22 می‌باشد.



شکل ۱۰: وقتی که یک برخورد رخ می‌دهد، داده داخل حافظه نهان به پردازنده ارسال می‌گردد

پردازنده در هر مراجعه به حافظه، ابتدا به حافظه نهان مراجعه می‌کند، اگر برخورد رخ دهد، داده مورد نیاز از داخل حافظه نهان به پردازنده ارسال می‌شود. در این حالت تأخیر دسترسی به حافظه تقریباً معادل یک کلاک خواهد بود. اگر فرض کنیم که پریود کلاک مساوی 2ns باشد، در این صورت مراجعه به حافظه در حدود 2ns طول می‌کشد.

اما اگر عدم برخورد رخ دهد و یا به عبارتی سیگنال Hit تولید شده در شکل ۱۰ مساوی 0 گردد، در این صورت باید به حافظه اصلی مراجعه نمود و داده مورد نظر را از آنجا خواند. در این صورت دسترسی به حافظه چند کلاک طول خواهد کشید. در یک عدم برخورد، ساده‌ترین کاری که یک پردازنده می‌تواند انجام دهد این است که پایپلاین خود را متوقف نماید (stall بدهد) تا اینکه داده مورد نظر از حافظه اصلی خوانده شود (همچنین به داخل حافظه نهان کپی گردد).

زمان متوسط دسترسی به حافظه نهان از فرمول زیر قابل محاسبه است:

$$t_a = t_c + (1-h)t_m$$

که در آن t_a زمان دسترسی به حافظه، t_c زمان دسترسی به حافظه نهان، h نرخ برخورد، "1-h" نرخ عدم برخورد و t_m زمان دسترسی به حافظه اصلی می‌باشد.

فرمول فوق با این فرض بدست آمده است که اولین مراجعه به حافظه نهان انجام می‌گیرد و اگر داده در حافظ نهان موجود نبود که احتمال آن "1-h" است، به سراغ حافظه اصلی می‌رویم و با زمان t_m داده را از حافظه اصلی می‌خوانیم. به طور مثال، اگر نرخ برخورد مساوی ۹۵٪، زمان دسترسی به حافظه نهان 2ns و زمان دسترسی به حافظه اصلی مساوی 200ns باشد، در این صورت متوسط زمان دسترسی به حافظه به صورت زیر محاسبه می-گردد:

$$t_a = 2ns + (1-0.95)200ns = 12ns$$

مثال: اگر در شکل ۱۰ طول داده هر خانه از حافظه اصلی مساوی ۸ بیت باشد، در این صورت اندازه حافظه نهان را به بیت مشخص نمایید.

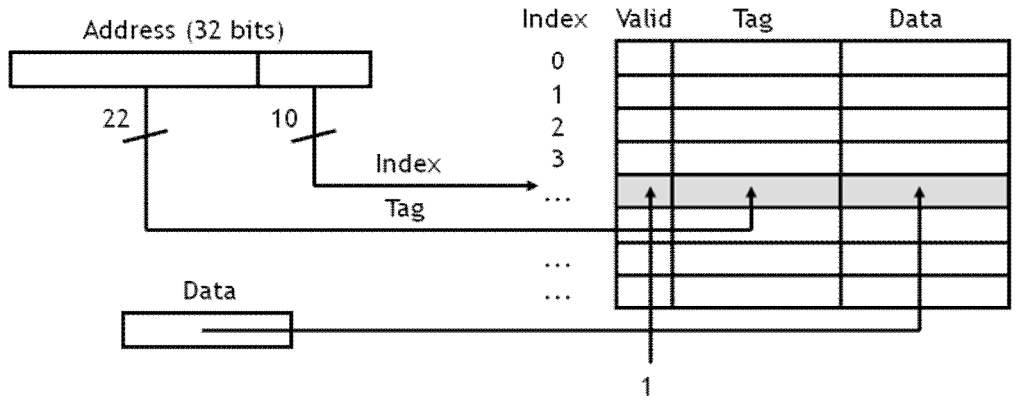
جواب: چون حافظه نهان برای نگهداری داده‌های حافظه اصلی مورد استفاده قرار می‌گیرد، به همین دلیل میدان داده هر خانه آن باید دارای طول ۸ بیت باشد. در هر خانه حافظه نهان علاوه بر میدان داده (۸ بیت)، میدان تگ (۲۲ بیت) و بیت اعتبار (یک بیت) نیز ذخیره می‌گردد، به همین دلیل اندازه حافظه نهان به صورت زیر محاسبه می‌گردد:

اندازه حافظه نهان به بیت = (تعداد خانه‌های حافظه نهان) × (اندازه هر خانه) = $2^k \times (1+22+8) = 2^{10} \times 31$

بارگذاری یک داده در حافظه نهان

بعد از اینکه داده از حافظه اصلی خوانده شد، قرار دادن یک کپی از آن داخل حافظه نهان سر راست می‌باشد. فرآیند کپی کردن که در شکل ۱۱ نشان داده شده است شامل مراحل زیر است:

۱. k بیت کم ارزش آدرس یک خانه از حافظه نهان را مشخص می‌نماید.
۲. m-k بیت بالای آدرس در میدان تگ آن خانه از حافظه نهان ذخیره می‌گردد.
۳. داده خوانده شده از حافظه اصلی در میدان داده آن خانه از حافظه نهان ذخیره می‌گردد.
۴. بیت اعتبار آن خانه با 1 مقداردهی می‌شود.



شکل ۱۱: نحوه کپی کردن یک داده به داخل حافظه نهان

اگر حافظه نهان پر شده باشد چه کاری باید انجام داد؟

سؤال سوم ما این بود که اگر حافظه نهان پر بود و یا اینکه ما نیاز داشته باشیم که یک خانه از حافظه نهان را که هم اکنون به یک آدرس از حافظه اصلی اختصاص داده شده، به یک آدرس دیگر اختصاص دهیم، در این صورت چه کاری باید انجام داد؟

ما به این سؤال قبلاً جواب داده‌ایم! یک عدم برخورد موجب می‌شود که یک داده جدید به داخل حافظه نهان کپی گشته و به طور اتوماتیک بر روی داده قبلی نوشته شود. این امر، یک نوع سیاست جایگزینی[1] به نام سیاست جایگزینی اخیراً کمتر استفاده شده[2] یا LRU می‌باشد که فرضش بر این است که داده‌های قدیمی‌تر نسبت به داده‌های جدیدتر کمتر مورد استفاده قرار می‌گیرند و به همین دلیل برای جایگزین شدن گزینه‌های مناسب‌تری هستند (سیاست LRU در ادامه بیشتر توضیح داده خواهد شد).

بهبود کارآیی حافظه نهان

در ادامه جهت بهبود کارآیی حافظه نهان و به عبارتی بهبود نرخ برخورد، سازمانهایی از حافظه نهان را ارائه خواهیم نمود. سعی خواهیم نمود تا به دو سؤال مهم جواب دهیم:

۱- آیا می‌توان مفهوم محلیت مکانی را هم به مانند محلیت زمانی در طراحی یک حافظه نهان لحاظ نمود؟

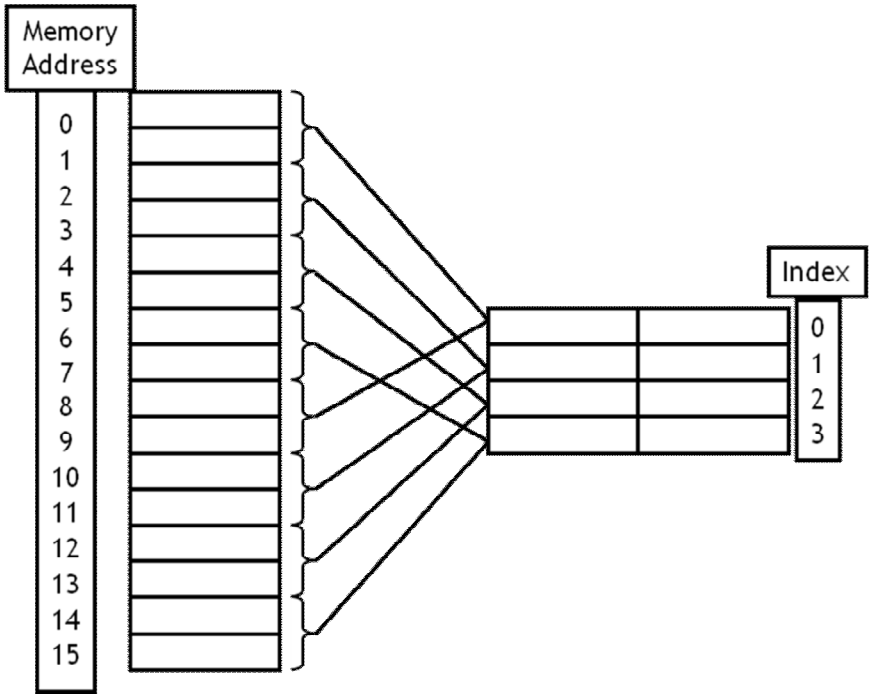
۲- چگونه می‌توان تعداد عدم برخوردها را در شرایطی که هنوز در داخل حافظه نهان خانه‌های خالی وجود دارند، کاهش داد (به چنین عدم برخوردهایی conflict misses گفته می‌شود)؟

بهره گیری از مفهوم محلیت مکانی

[1] - Replacement Policy
[2] - Least Recently Used replacement policy

مفهوم محلیت مکانی این است که اگر در حال حاضر آدرس "i" از حافظه مورد دسترسی قرار گیرد، در آینده‌ای نزدیک آدرس "i+1" نیز مورد دسترسی قرار خواهد گرفت. اگر بخواهیم از مفهوم محلیت مکانی بهره ببریم، باید به هنگام کپی کردن آدرس i به داخل حافظه نهان، چند آدرس بعدی را نیز کپی کنیم. ما در مثال‌های قبلی فرض کردیم که در هر آدرس از حافظه اصلی فقط یک بایت نگهداری می‌شود و همین طور میدان داده در هر خانه حافظه نهان نیز طول یک بایت را دارد. این فرض نمی‌تواند از مفهوم محلیت مکانی استفاده نماید.

برای بهره گیری از مفهوم محلیت مکانی، باید کاری کنیم که داخل هر خانه از حافظه نهان (هر بلوک حافظه نهان) بایت‌های داده بیشتری که مربوط به آدرس‌های متوالی می‌باشند، ذخیره گردد. در شکل ۱۲ نحوه نگهداری اطلاعات دو آدرس متوالی از حافظه اصلی در داخل یک بلوک (خانه) از حافظه نهان نشان داده شده است. در این شکل، چون هر بلوک حافظه نهان می‌تواند دو بایت داده از حافظه اصلی را نگهداری کند، به همین دلیل ما می‌توانیم در هر بار کپی کردن داخل حافظه نهان، مقدار دو بایت داده را داخل آن بنویسیم. به طور مثال، اگر ما بخواهیم از آدرس 12 حافظه اصلی بخوانیم، در این صورت داده‌های موجود در آدرس‌های 12 و 13 هر دو به بلوک شماره 2 حافظه نهان کپی می‌گردند.



شکل ۱۲: نحوه نگهداری اطلاعات دو آدرس متوالی از حافظه اصلی در هر بلوک (خانه) حافظه نهان

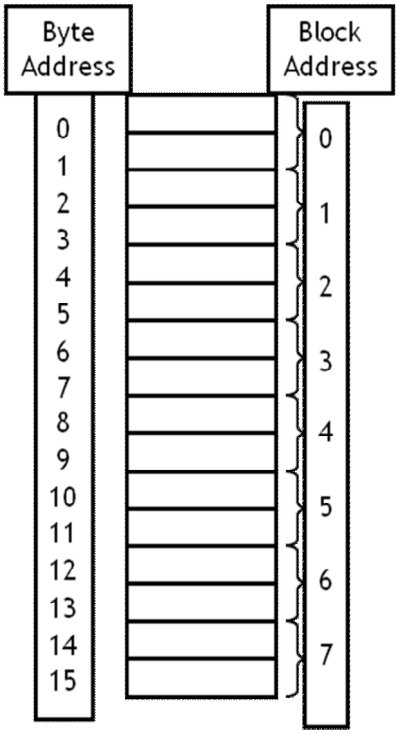
برای تشخیص اینکه هر بایت از حافظه اصلی داخل کدام بلوک از حافظه نهان قرار خواهد گرفت، باید شیوه آدرس دهی بلوکی استفاده کنیم. در این شیوه، اگر هر بلوک از حافظه نهان شامل 2^n بایت باشد، به صورت مفهومی می‌توان حافظه اصلی را هم به بلوک‌هایی با اندازه 2^n تقسیم نمود. در این حالت، بلوک مربوط به آدرس "i" حافظه اصلی در داخل حافظه نهان از فرمول زیر قابل محاسبه خواهد بود:

$i/2^n$ = شماره بلوک برای آدرس i حافظه اصلی

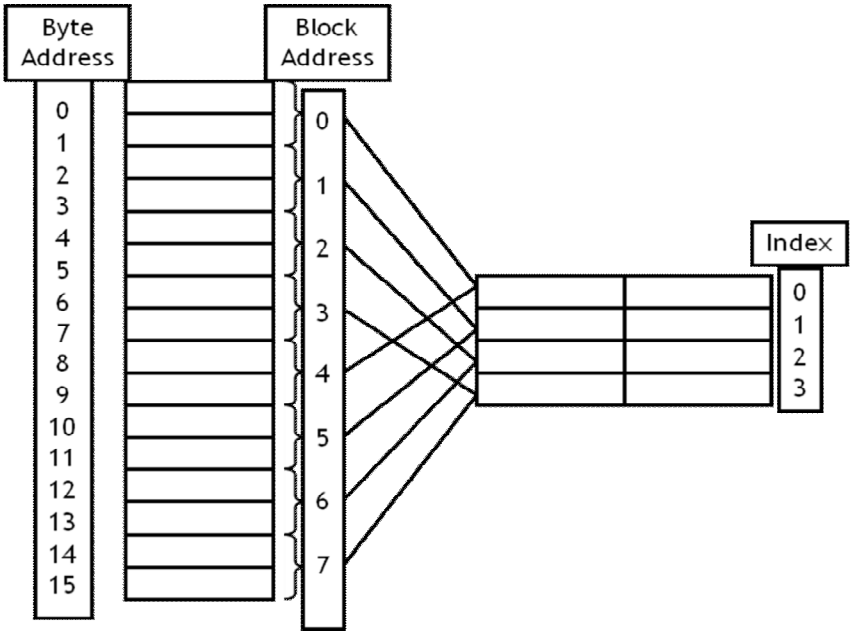
ما فرض کردیم که هر بلوک حافظه نهان شامل دو بایت است، به همین دلیل می‌توانیم یک حافظه اصلی با ۱۶ بایت را به صورت حافظه‌ای با ۸ بلوک در نظر بگیریم (شکل ۱۳). همان طور که در این شکل مشاهده می‌شود، آدرس‌های 12 و 13 از حافظه اصلی، به بلوک شماره 6 از حافظه اصلی تعلق دارند چون 12/2=6 و 13/2=6. اگر شماره بلوک مربوط به یک آدرس را بدانیم، می‌توانیم فرآیند نگاشت را همانند قبل انجام دهیم. در این حالت، برای اینکه بدانیم هر بلوک از حافظه اصلی به کدام بلوک از حافظه نهان نگاشته می‌شود، باید از عملگر mod استفاده کنیم و باقیمانده تقسیم صحیح شماره بلوک حافظه اصلی به تعداد بلوک حافظه اصلی را بدست آوریم:

(تعداد بلوک حافظه نهان) mod (شماره آن بلوک حافظه اصلی) = شماره بلوکی از حافظه نهان که یک بلوک از حافظه اصلی به آن نگاشت خواهد شد

به طور مثال بلوک شماره 6 از حافظه اصلی که شامل آدرس‌های 12 و 13 است به بلوک شماره 2 از حافظه نهان نگاشته خواهد شد چون 6 mod 4=2. نحوه نگاشت بلوک‌های حافظه اصلی به بلوک‌های حافظه نهان در شکل ۱۴ نشان داده شده است.

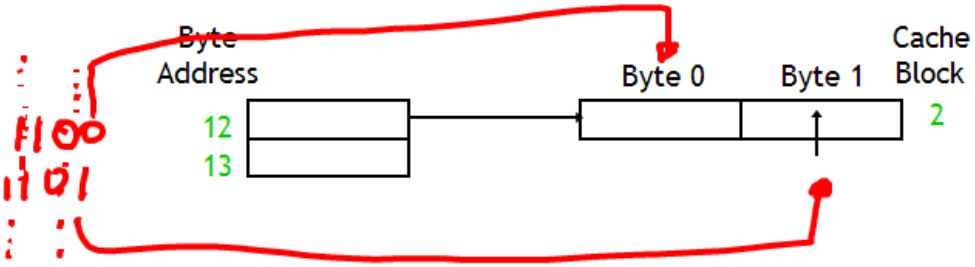


شکل ۱۳: یک حافظه اصلی با ۱۶ بایت که به صورت حافظه‌ای با ۸ بلوک در نظر گرفته شده است



شکل ۱۴: نحوه نگاشت بلوک‌های حافظه اصلی به بلوک‌های حافظه نهان

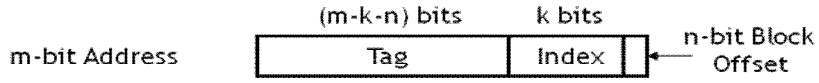
موقعیکه ما به یک بایت داده در حافظه اصلی دسترسی پیدا می‌کنیم، کل بلوک مربوط به آن بایت را به داخل حافظه نهان کپی می‌کنیم و این کار را با این امید انجام می‌دهیم که از مزایای محلیت مکانی استفاده کنیم. در مثال ما، اگر یک برنامه از آدرس 12 بخواند، ما کل بلوک 6 از حافظه اصلی را که آدرس 12 متعلق به آن است (آدرس‌های 12 و 13 حافظه اصلی) را به داخل بلوک شماره 2 حافظه نهان کپی می‌کنیم. توجه کنید که آدرس 13 هم به بلوک 6 حافظه اصلی تعلق دارد و خواندن از آدرس 13 هم موجب کپی شدن کل بلوک 6 از حافظه اصلی به داخل بلوک 2 حافظه نهان خواهد شد. برای اینکه کارها را ساده‌تر کنیم، بایت شماره j از یک بلوک حافظه اصلی به بایت شماره j از بلوک مربوط به آن در حافظه نهان کپی خواهد شد.



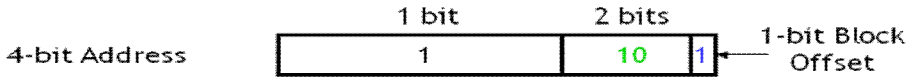
شکل ۱۵: بایت شماره j از یک بلوک داخل حافظه اصلی به بایت شماره j از بلوک مربوط به آن در حافظه نهان کپی می‌شود

پیدا کردن داده در یک حافظه نهان بلوک بندی شده

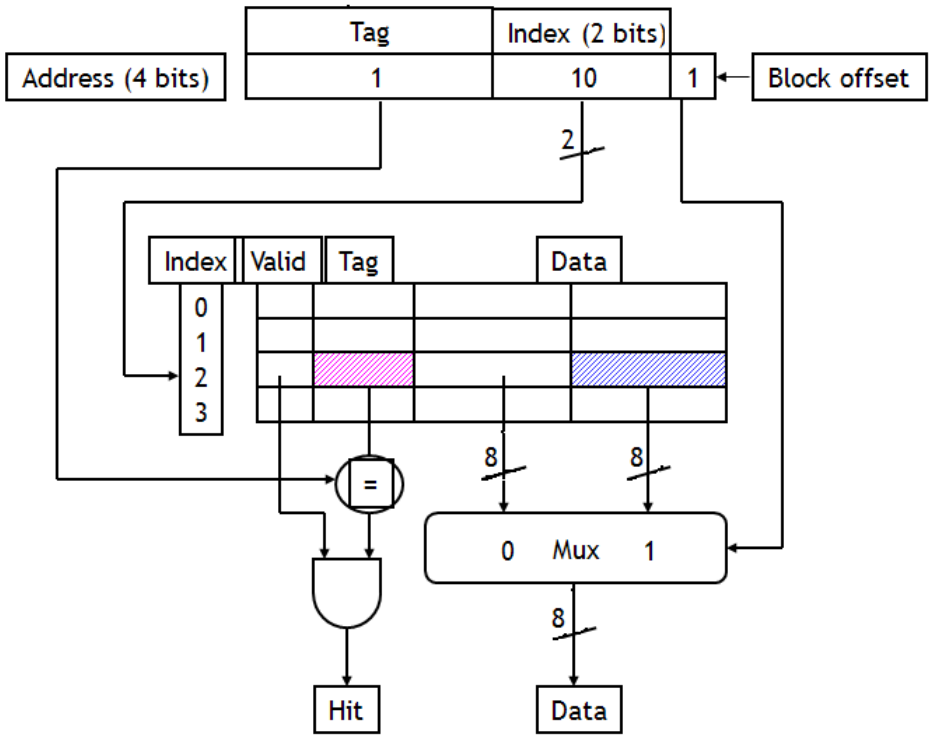
فرض کنید که یک حافظه نهان با 2^k بلوک و 2^n بایت داخل هر بلوک داشته باشیم. همچنین فرض کنید تعداد بیت آدرس حافظه اصلی مساوی m باشد. در این صورت برای مشخص کردن یک بلوک از حافظه نهان یا اندیس کردن آن به k بیت و برای مشخص یک بایت در داخل یک بلوک حافظه نهان (آفست بلوک) به n بیت نیاز داریم. بنابراین، تعداد m-k-n بیت برای تگ باقی خواهد ماند (شکل ۱۶). در واقع n بیت کم ارزش آدرس مشخص کننده آفست بلوک، k بیت بعدی نشان دهنده شماره بلوک و m-k-n بیت بالای آدرس تگ را تشکیل می‌دهد. در مثال قبلی، ما از یک حافظه نهان با 2^2 بلوک و تعداد 2^1 بایت داخل هر بلوک استفاده کرده‌ایم، در این صورت برای پیدا کردن آدرس 13 (1101) در حافظه نهان باید به بایت 1 از بلوک 2 حافظه نهان مراجعه نمود و تگ آن بلوک را با تگ این آدرس که 1 است مقایسه نمود (شکل ۱۷).



شکل ۱۶: نحوه تقسیم بندی بیت‌های آدرس برای جستجو در یک حافظه نهان بلوک بندی شده نگاشت مستقیم



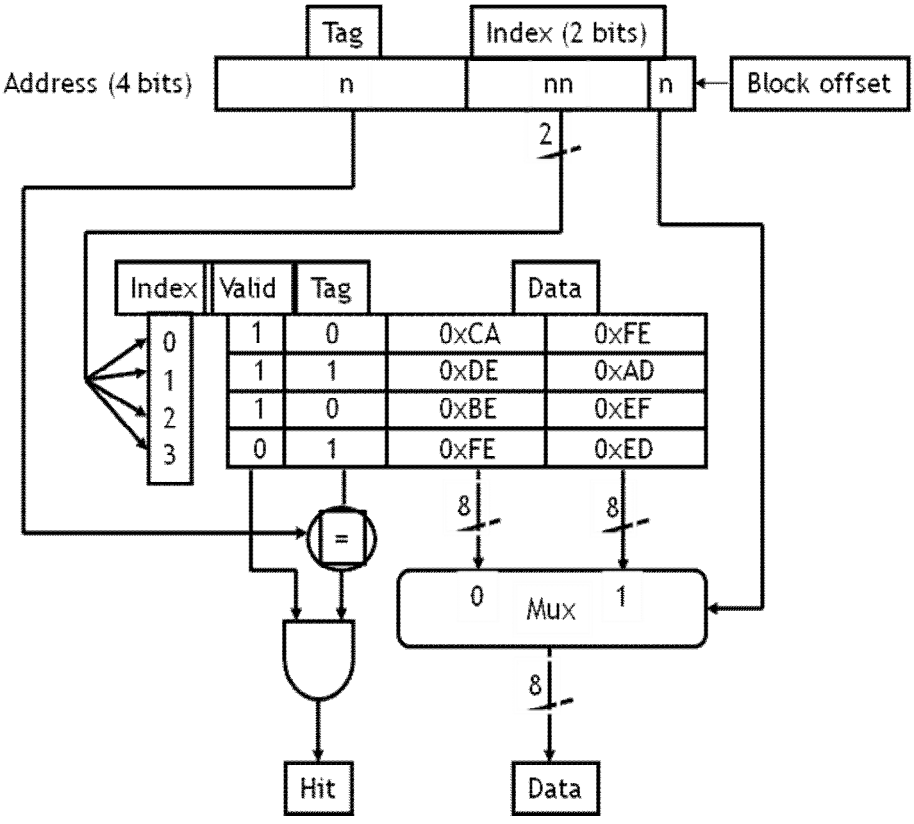
شکل ۱۷: محل ذخیره سازی آدرس 13 حافظه اصلی در یک حافظه نهان بلوک بندی شده با 4 بلوک 2 بایتی



شکل ۱۸: نحوه جستجوی یک آدرس در یک حافظه نهان بلوک بندی شده با دو بایت داخل هر بلوک

سازمان یک حافظه نهان بلوک بندی شده با دو بایت داده داخل هر بلوک در نشان شکل ۱۸ داده شده است. همان طور که در این شکل مشاهده می‌شود، داخل هر بلوک حافظه نهان غیر از دو بایت، یک میدان تگ و یک بیت اعتبار نیز نگهداری می‌گردد. نحوه کار به این صورت است که ابتدا با استفاده از بیت‌های اندیس یک بلوک (خانه) از حافظه مشخص می‌شود. سپس بیت اعتبار آن چک می‌شود و تگ آن بلوک نیز با تگ تولید شده توسط پردازنده مقایسه می‌گردد. اگر بیت اعتبار یک باشد و همزمان با آن، تگ موجود در این بلوک با تگ تولید شده توسط پردازنده مساوی باشد، در این صورت یک برخورد رخ داده و باید داده از داخل حافظه نهان به پردازنده فرستاده شود. این داده توسط یک مالتی پلکسر ۲ به ۱ از بین دو بایت داخل آن بلوک انتخاب می‌گردد. اینکه کدام بایت انتخاب شود، توسط n بیت از آدرس تولید شده توسط پردازنده که آفست بلوک نامیده می‌شود انجام می‌گیرد.

در اینجا m=4، k=2، n=1 می‌باشد، بنابراین اندازه این حافظه نهان مساوی $2^k\times(1+(m-k-n)+16))=72$ بیت خواهد بود.



شکل ۱۹: مثالی از سازمان حافظه نهان بلوکی نگاشت مستقیم با دو بایت داخل هر بلوک

مثال: شکل ۱۹ را در نظر بگیرید و مشخص کنید که برای هر یک از آدرس‌های باینری 1010، 1110، 0001 و 1101 چه بایتی از حافظه نهان خوانده می‌شود؟

جواب:

- برای آدرس 1010 چون اندیس مساوی باینری 01 یا عدد 1 می‌باشد به بلوک 1 حافظه نهان مراجعه می‌شود. بیت اعتبار این بلوک 1 است. پس اگر تگ‌ها مساوی باشند، برخورد رخ خواهد داد. تگ آدرس 1010 مساوی 1 است و تگ موجود در بلوک 1 نیز مساوی 1 است، بنابراین این دو تگ مساوی هستند و برخورد رخ می‌دهد. چون آفست بلوک در آدرس 1010 مساوی 0 است پس مالتی پلکسر بایت شماره 0 از بلوک شماره 1 حافظه نهان را که مساوی 0xDE است انتخاب نموده و به پردازنده ارسال می‌کند.
- برای آدرس 1110 چون اندیس مساوی باینری 11 یا عدد 3 می‌باشد به بلوک شماره 3 حافظه نهان مراجعه می‌شود. بیت اعتبار این بلوک مساوی 0 است به همین دلیل یک عدم برخورد اتفاق می‌افتد. بنابراین، باید به حافظه اصلی مراجعه نمود و محتوای آدرس 1110 از حافظه اصلی را به پردازنده ارسال نمود. چون این آدرس متعلق به بلوک 7 حافظه اصلی است، همزمان با فرستادن داده آدرس 1110 به پردازنده، بلوک 7 حافظه اصلی نیز به بلوک شماره 3 حافظه نهان کپی می‌گردد.
- برای آدرس 0001 برخورد رخ داده و داده 0xFE به پردازنده ارسال می‌گردد.
- برای آدرس 1101 با اینکه بیت اعتبار بلوک 2 مساوی 1 است ولی چون تگ‌ها مساوی نیستند، یک عدم برخورد رخ می‌دهد و باید جهت خواندن داده به حافظه اصلی مراجعه نمود.

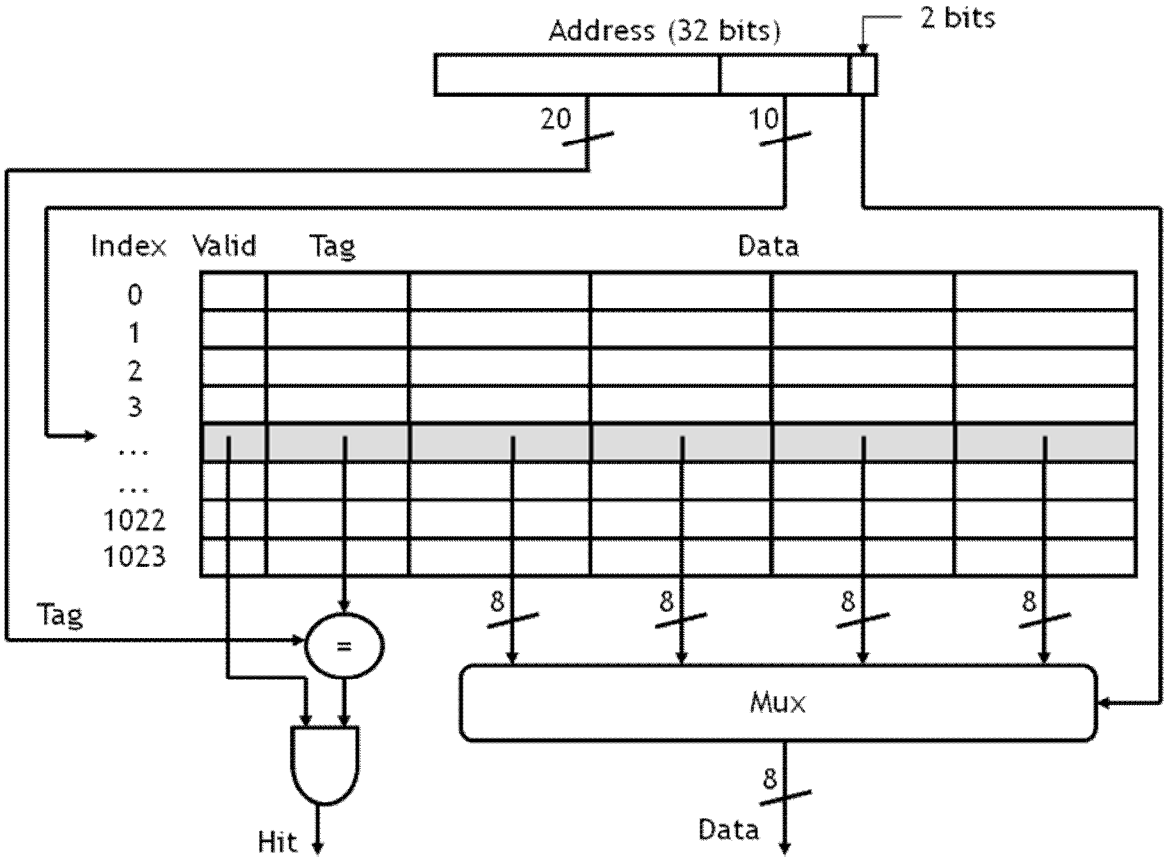
توجه: به روش دیگری نیز می‌توان آفست بلوک و شماره بلوک حافظه نهان را بدست آورد. آفست بلوک یک آدرس را به راحتی می‌توان از تقسیم آن آدرس به 2^n و بدست آوردن باقیمانده آن بدست آورد (عملگر mod). شماره بلوکی از حافظه نهان را که باید به آن مراجعه کنیم نیز با دو گام می‌توان بدست آورد. ابتدا حاصل تقسیم آدرس تولید شده به 2^n را بدست می‌آوریم که شماره بلوک در حافظه اصلی بدست می‌آید و سپس شماره بلوک حافظه اصلی را به 2^k تقسیم می‌کنیم و باقیمانده آن را بدست می‌آوریم (عملگر mod) تا شماره بلوک متناظر در حافظه نهان بدست آید.

مثال: یک حافظه نهان نگاشت مستقیم بلوکی با اندازه بزرگتر که دارای 1024 بلوک بوده و در هر بلوک آن ۴ بایت نگهداری می‌شود، در شکل ۲۰ نشان داده شده است. در این شکل فرض شده که طول آدرس تولید شده توسط پردازنده مساوی ۳۲ بیت است. برای این شکل مشخص کنید که آدرس 6146 حافظه اصلی به کدام بلوک و کدام آفست داخل آن بلوک نگاشت می‌شود؟

جواب: آفست بلوک 2 است که به این صورت بدست می‌آید: 6146 mod 4=2

دو گام برای بدست آوردن شماره بلوک حافظه نهان: بلوک مربوط به این آدرس در حافظه اصلی مساوی 6146/4=1536 و شماره بلوک متناظر در حافظه نهان مساوی 1536 mod 1024=512 خواهد بود. بنابراین، اندیس مربوط به آدرس 6146 مساوی 512 خواهد بود.

توجه کنید که چهار آدرس 6144، 6145، 6146 و 6147 متعلق به یک بلوک در حافظه اصلی هستند که شماره آن بلوک 1536 است. آفست این چهار آدرس به ترتیب مساوی 0، 1، 2 و 3 می‌باشد. بلوک متناظر این بلوک در حافظه نهان بلوک 512 می‌باشد که داده‌های چهار آدرس 6144، 6145، 6146 و 6147 به ترتیب در آفست‌های 0، 1، 2 و 3 آن کپی می‌گردند.



شکل ۲۰: یک حافظه نهان نگاشت مستقیم از نوع بلوکی که داخل هر بلوک آن ۴ بایت نگهداری می‌شود

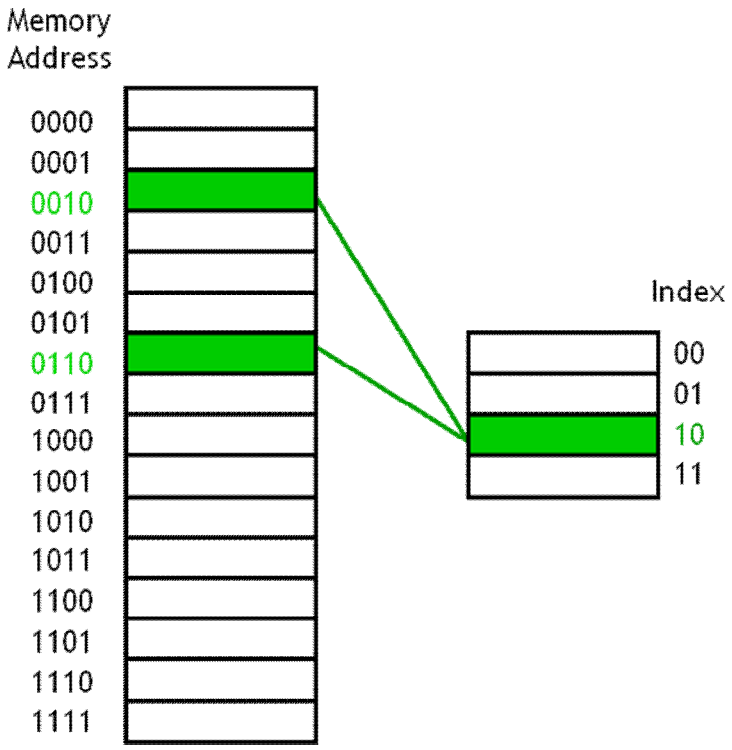
معایب حافظه نهان نگاشت مستقیم

حافظه‌های نهان نگاشت مستقیم ساده‌تر هستند. از آنجا که هر آدرس حافظه فقط متعلق به یک بلوک از حافظه نهان است، اندیس‌ها و آفست‌ها می‌توانند با استفاده از عملگرهای بیتی یا ریاضیات ساده به راحتی محاسبه گردند.

اما اگر آدرس‌های 2، 6، 2، 6 و ... به ترتیب تولید شوند (شکل ۲۱ را در نظر بگیرید)، در این صورت با توجه به اینکه آدرس‌های 2 و 6 به یک خانه از حافظه نهان نگاشت می‌شوند، با هر دسترسی به حافظه نهان یک عدم برخورد رخ خواهد داد که موجب خواهد شد به حافظه اصلی مراجعه کنیم و یک کپی از داده‌ها را به خانه 2

حافظه نهان منتقل نماییم. با اینکه این حافظه نهان دارای چهار خانه است ولی نگاشت مستقیم اجازه نمی‌دهد از همه آنها استفاده کنیم. در حالی که خانه‌های دیگر غیر از خانه 2 خالی هستند، با این وجود استفاده‌ای از آنها نمی‌شود و مرتباً داده آدرس 2 بر روی داده آدرس 6 نوشته می‌شود و بالعکس که این مسأله موجب عدم برخوردهای زیادی می‌گردد.

این عیب حافظه نهان نگاشت مستقیم توسط حافظه نهان انجمنی کامل مرتفع می‌گردد.



شکل ۲۱: یک شکل جهت نشان دادن معایب حافظه نهان نگاشت مستقیم

حافظه نهان انجمنی کامل

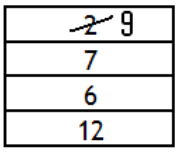
یک حافظه نهان انجمنی کامل[1] اجازه می‌دهد که داده در هر یک از خانه‌های حافظه نهان ذخیره گردد. وقتی که داده‌ای از حافظه اصلی خوانده می‌شود، آن داده می‌تواند در هر خانه بلا استفاده از حافظه نهان قرار گیرد. در این حالت تداخلی بین دو یا چند آدرس که می‌خواهند به یک آدرس از حافظه نهان نگاشت شوند (نگاشت مستقیم) پیش نخواهد آمد.

برای حافظه نهان انجمنی کامل، اگر آدرس‌های 2، 6، 2، 6 و ... به ترتیب تولید شوند، در این صورت خانه 2 حافظه نهان توسط آدرس 2 مورد استفاده قرار می‌گیرد و آدرس 6 هم می‌تواند به جای خانه 2 به خانه خالی 3

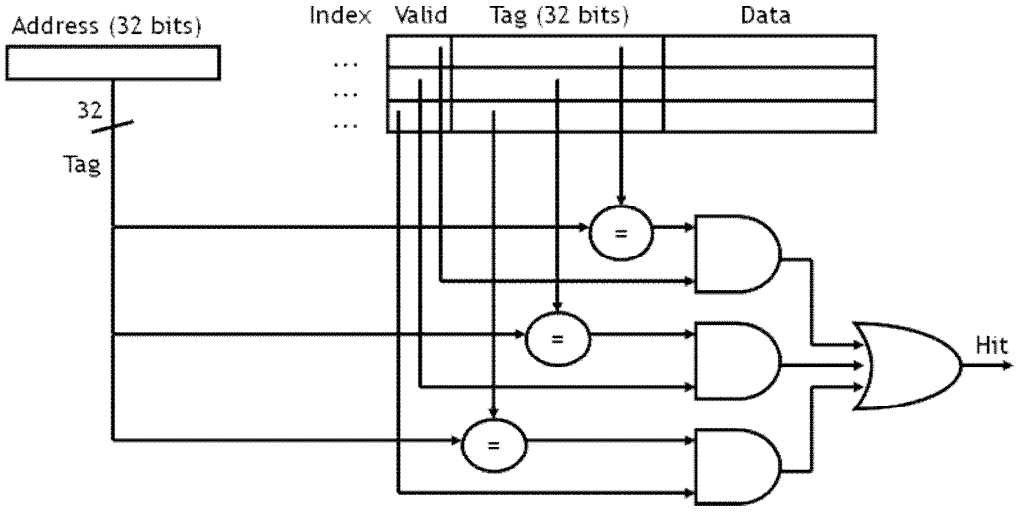
[1] - Fully-associative cache

نگاشت گردد. در این صورت، برای تمام مراجعات بعدی به آدرس‌های 2 و 6 برخورد رخ خواهد داد و عدم برخورد پیش نخواهد آمد.

در حافظه نهان انجمنی کامل، اگر همه خانه‌های حافظه نهان پر باشد و داده جدیدی را بخواهیم داخل حافظه نهان قرار دهیم، در این صورت باید یکی از خانه‌های حافظه نهان را انتخاب کنیم و داده آن را با داده جدید جایگزین نماییم و همین طور میدان تگ آن را نیز بروز کنیم تا منعکس کننده آدرس جدید باشد. می‌توانیم از سیاست جایگزینی LRU استفاده نماییم. با توجه به شکل ۲۲، با فرض اینکه آدرس‌های 2، 7، 6 و 12 به ترتیب تولید گردند، گنجایش حافظه نهان انجمنی کامل تکمیل می‌گردد، در این صورت با تولید آدرس 9 یک عدم برخورد رخ می‌دهد و سیاست جایگزینی LRU آدرس صفر حافظه نهان را که هم اکنون به آدرس 2 اختصاص یافته، انتخاب نموده و آدرس 9 را در آن خانه جایگزین آدرس 2 می‌کند.



شکل ۲۲: اعمال سیاست LRU برای حافظه نهان انجمنی کامل



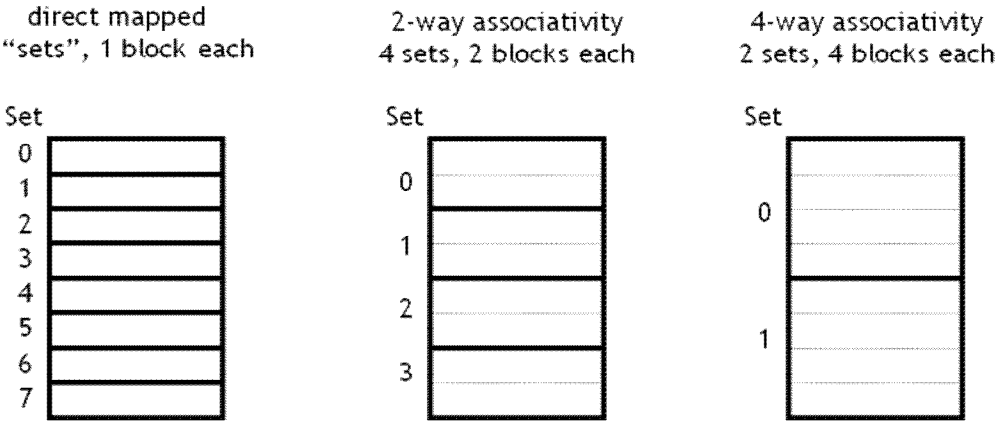
شکل ۲۳: سازمان یک حافظه نهان انجمنی کامل

عیب بزرگ حافظه نهان انجمنی کامل، هزینه بالای پیاده سازی آن می‌باشد. طراحی یک حافظه نهان انجمنی کامل در شکل ۲۳ نشان داده شده است. در حافظه نهان انجمنی کامل داخل هر خانه حافظه نهان، غیر از بیت اعتبار و بایت داده، تمام بیت‌های آدرس نیز ذخیره می‌گردد. از آنجا که هر آدرس می‌تواند داخل هر کدام از خانه‌ها نگاشت شود، به همین دلیل باید کل آدرس را در خانه‌ای که آن آدرس به آن نگاشت می‌گردد نگهداری کنیم (در واقع کل آدرس به عنوان تگ نگهداری می‌شود).

در حافظه نهان انجمنی کامل از آنجا که داده مربوط به یک آدرس می‌تواند در هر کدام از خانه‌های حافظه نهان قرار گیرد، به همین دلیل برای هر آدرسی که توسط پردازنده تولید گردد باید آن آدرس را با تگ تمام خانه‌های حافظه نهان مقایسه نمود تا اینکه تشخیص داد که آیا این آدرس هم اکنون به یکی از خانه‌های حافظه نهان نگاشت شده است یا نه. به همین دلیل، تعداد زیادی مقایسه کننده نیاز خواهیم و این موجب افزایش هزینه می‌گردد.

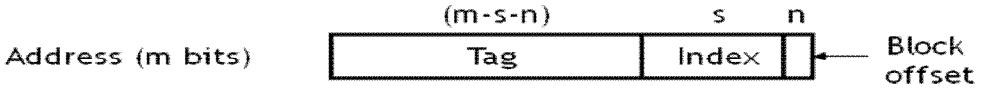
حافظه نهان انجمنی-گروهی

حافظه انجمنی-گروهی[۱] یک راهکار میانه است که در آن خانه‌های حافظه نهان به گروه‌هایی به نام مجموعه[۲] تقسیم بندی می‌شوند. هر آدرس از حافظه اصلی فقط به یک مجموعه از حافظه نهان نگاشت می‌گردد ولی در داخل آن مجموعه می‌تواند داخل هر کدام از خانه‌ها قرار گیرد. اگر هر مجموعه شامل 2^x خانه باشد، در این صورت آن حافظه نهان، حافظه نهان انجمنی 2^x راهه[۳] نامیده می‌شود. چند سازمان ممکن برای حافظه نهان انجمنی-گروهی با ۸ خانه، در شکل ۲۴ نشان داده شده است. توجه داشته باشید که بلوک معادل خانه است.



شکل ۲۴: سازمانهای مختلف برای یک حافظه انجمنی گروهی با ۸ خانه

برای پیدا کردن یک آدرس در حافظه نهان انجمنی گروهی، می‌توان مشابه با حافظه‌های نهان قبلی عمل نمود. اگر یک حافظه نهان شامل 2^s مجموعه و هر خانه دارای 2^n بایت باشد، در این صورت آدرس حافظه می‌تواند به صورت شکل ۲۵ تقسیم بندی گردد.



شکل ۲۵: نحوه تقسیم بندی آدرس حافظه در حافظه نهان انجمنی گروهی

[1] - Set-associative cache
[2] - Set
[3] - 2^x-way associative cache

بر اساس محاسبات ریاضی، محاسبات زیر را می‌توان انجام داد:

$$\text{آفست بلوک} = (\text{آدرس حافظه}) \bmod 2^n$$

$$\text{آدرس بلوک} = (\text{آدرس حافظه}) / 2^n$$

$$\text{اندیس مجموعه} = (\text{آدرس بلوک}) \bmod 2^s$$

مثال: با در نظر گرفتن شکل ۲۴، مشخص کنید که در هر یک از سه سازمان نشان داده شده، آدرس 6195 در کجا می‌تواند قرار گیرد؟ فرض کنید در داخل هر بلوک ۱۶ بایت قرار داشته باشد.

جواب: بر اساس فرمول‌های بالا داریم:

$$\text{آفست بلوک} = 6195 \bmod 16 = 3$$

$$\text{آدرس بلوک} = 6195 / 16 = 387$$

- برای حافظه نهان نگاشت مستقیم چون تعداد مجموعه‌ها مساوی ۸ است داریم:

$$\text{اندیس مجموعه} = 387 \bmod 8 = 3$$

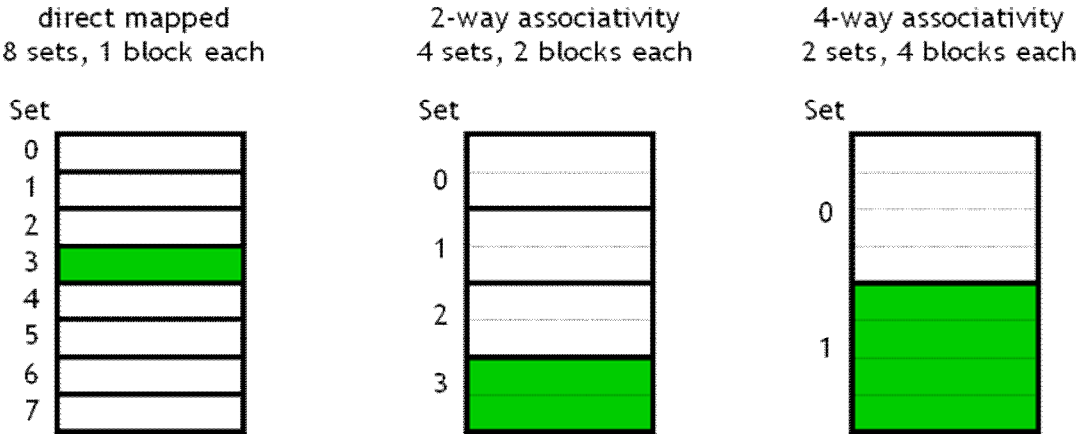
- برای حافظه انجمنی-گروهی ۲ راهه که در اینجا شامل ۴ مجموعه است، داریم:

$$\text{اندیس مجموعه} = 387 \bmod 4 = 3$$

- برای حافظه انجمنی-گروهی ۴ راهه که در اینجا شامل ۲ مجموعه است، داریم:

$$\text{اندیس مجموعه} = 387 \bmod 2 = 1$$

با توجه به محاسبات صورت گرفته، آدرس 6195 می‌تواند در هر یک از بلوک‌های پر رنگ شده نشان داده شده در شکل ۲۶ قرار گیرد. توجه کنید که بلوک‌های پررنگ شده در هر سازمان، همگی جزو یک مجموعه می‌باشند.



شکل ۲۶: محل نگاشت آدرس 6195 برای سه سازمان مختلف از حافظه نهان نگاشت انجمنی-گروهی

جایگزینی بلوک در حافظه نهان انجمنی-گروهی

یک بلوک خالی در داخل یک مجموعه صحیح، می‌تواند برای ذخیره داده مورد استفاده قرار گیرد. اگر بلوک خالی وجود نداشته باشد، در این صورت کنترلر حافظه نهان ممکن است بر اساس سیاست LRU یکی از بلوک-های داخل آن مجموعه را برای جایگزینی انتخاب نماید. اگر درجه انجمنی یک حافظه نهان انجمنی گروهی بالا باشد (به تعداد بلوک داخل هر مجموعه درجه انجمنی گفته می‌شود)، در این صورت پیدا کردن بلوک اخیراً

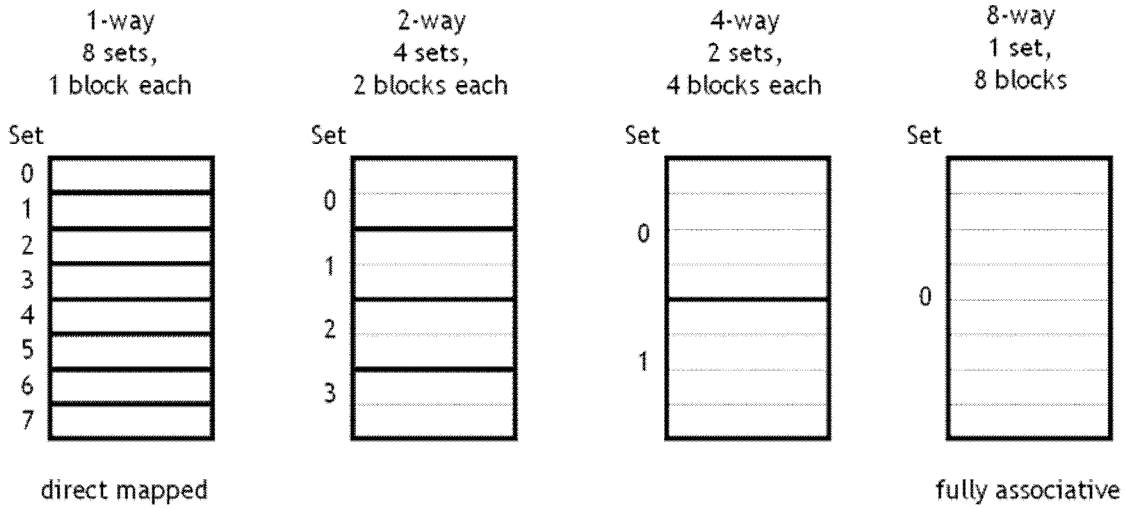
کمتر استفاده شده بر اساس سیاست LRU مشکل‌تر و گرانتر خواهد بود. به همین دلیل از تقریب هم در این زمینه استفاده می‌گردد. ما خیلی وارد جزئیات این مسأله نمی‌شویم.

حافظه نهان انجمنی گروهی به عنوان یک ایده عمومی

حافظه نهان انجمنی گروهی را می‌توان به عنوان یک ایده عمومی مطرح نمود که بقیه طراحی‌های حافظه نهان را می‌توان حالت خاصی از آن در نظر گرفت.

- اگر ما به حافظه نهان انجمنی-گروهی یک راهه (1-way) توجه کنیم، می بینیم که همان حافظه نهان نگاشت مستقیم است.
- به طور مشابه، اگر یک حافظه نهان دارای 2^k بلوک باشد، در این صورت یک حافظه انجمنی-گروهی 2^k راهه (2^k-way) همانند حافظه انجمنی کامل خواهد شد.

شکل ۲۷ نشان می‌دهد که چگونه می‌توان از حافظه نهان انجمنی-گروهی به عنوان یک راهکار عمومی استفاده نمود.



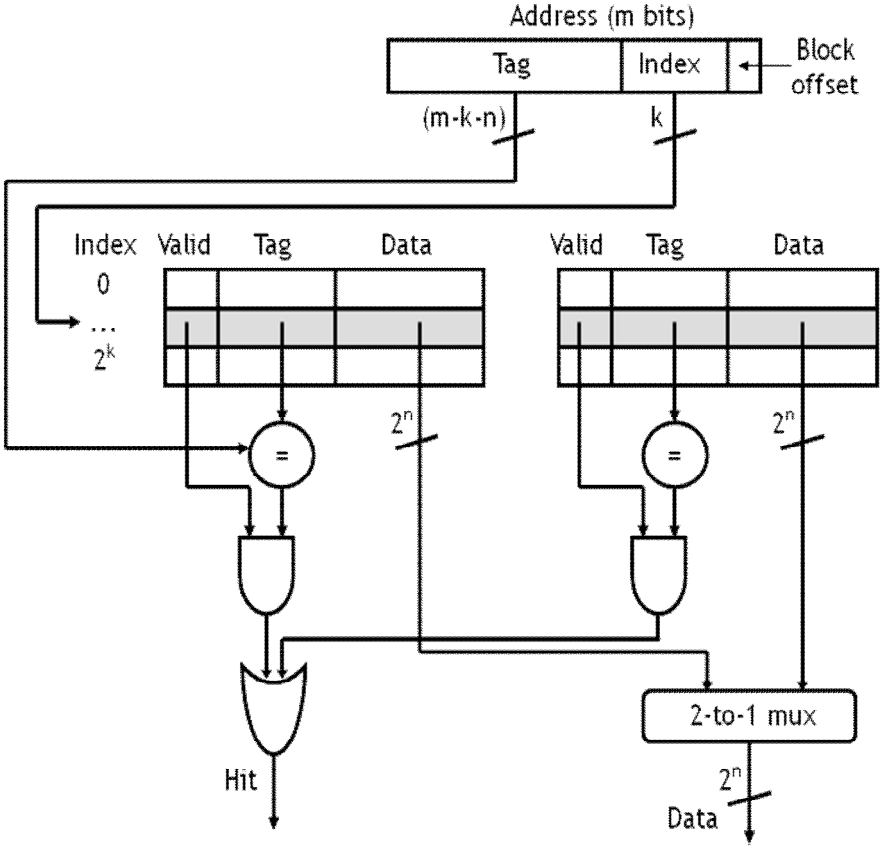
شکل ۲۷: حافظه نهان انجمنی-گروهی به عنوان یک راهکار عمومی

پیاده سازی یک حافظه نهان انجمنی-گروهی ۲ راهه

شکل ۲۸ پیاده سازی یک حافظه انجمنی-گروهی ۲ راهه را نشان می‌دهد. در این حافظه نهان، داخل هر مجموعه دو بلوک قرار دارد که هر بلوک بیت اعتبار، میدان تگ و میدان داده مخصوص به خود را دارد. هر آدرسی که توسط پردازنده تولید می‌گردد، دارای یک بخش اندیس می‌باشد که یک مجموعه را در داخل حافظه نهان مشخص می‌کند. همچنین آدرس تولید شده، دارای یک بخش تگ می‌باشد که باید با هر دو تگ داخل آن مجموعه مقایسه گردد. اگر تگ تولید شده با یکی از این دو تگ مساوی گردد و همچنین بیت اعتبار برای بلوکی که تگ آن مساوی تگ تولید شده گردیده، برابر با 1 باشد، در این صورت یک برخورد رخ داده و داده موجود در

بلوک مورد نظر به پردازنده ارسال می‌گردد. در غیر این صورت یک عدم برخورد رخ می‌دهد و باید برای آدرس تولید شده توسط پردازنده به حافظه اصلی مراجعه کنیم. همزمان با خوانده شدن از حافظه به پردازنده، یک کپی از بلوک مربوط به آن آدرس به داخل یکی از دو بلوک داخل آن مجموعه در حافظه نهان ریخته می‌شود. اگر هر دو بلوک آن مجموعه پر باشند، بر اساس سیاست جایگزینی LRU یکی از آن دو بلوک برای جایگزینی انتخاب می‌گردد.

اگر بخواهیم پیاده سازی حافظه نهان انجمنی-گروهی ۲ راهه را با حافظه نهان انجمنی کامل مقایسه کنیم، می-بینیم که به جای تعداد زیادی مقایسه کننده، در حافظه نهان انجمنی-گروهی ۲ راهه فقط دو مقایسه کننده وجود دارد. همچنین طول میدان تگ نیز در حافظه نهان انجمنی-گروهی ۲ راهه نسبت به حافظه نهان انجمنی کامل کاهش پیدا کرده است.

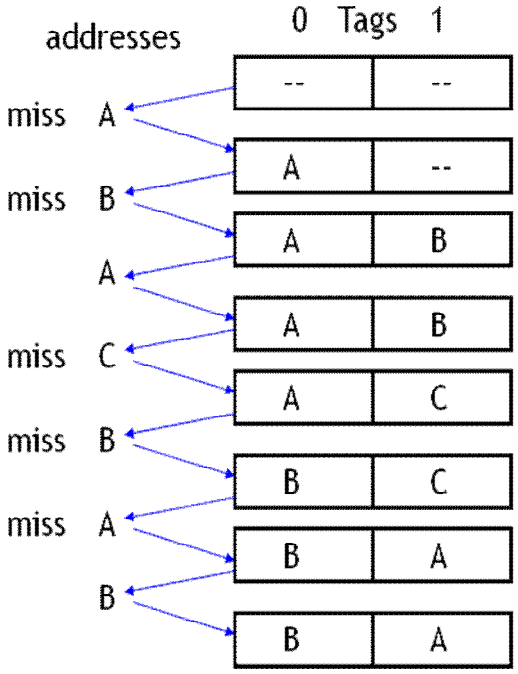


شکل ۲۸: پیاده سازی یک حافظه انجمنی- گروهی ۲ راهه

مثال: فرض کنید که یک حافظه نهان انجمنی کامل با دو بلوک در اختیار داشته باشیم. اگر از سیاست جایگزینی LRU برای این حافظه نهان استفاده کنیم، در این صورت برای دنباله آدرس‌های زیر چند عدم برخورد (miss) اتفاق خواهد افتاد؟

دنباله آدرس‌ها از راست به چپ: A، B، A، C، B، A، B

جواب: مطابق با راه حل نشان داده شده در شکل ۲۹ تعداد عدم برخوردها مساوی ۵ خواهد بود.



شکل ۲۹: مثال تعداد عدم برخورد برای یک حافظه نهان انجمنی کامل با دو بلوک با فرض تولید دنباله‌ای از آدرس‌ها