دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده مهندسی برق و کامپیوتر

آزمایشگاه معماری کامپیوتر

گزارش 7 (cache)

امیرعلی آرزم جو - 810198342

فرناز فیض - 810198454

توضیحات کلی:

در جلسه ی گذشته حافظه خارجی SRAM به همراه کنترلر آن به پردازنده اضافه گردید. مراجعه به حافظه خارجی نیاز به کلاک سایکل بیشتری دارد و در نتیجه سرعت انجام عملیات را کاهش میدهد. در این بخش حافظه نهان cache را به داخل ARM اضافه میکنیم. این مزیت وجود دارد که حافظه نهان سرعت بالاتری از SRAM دارد اما به علت مساحتی که اشغال میکند و هزینه ی بالای آن محدودیت حافظه داریم. اگر به حافظه نهان برای خواندن داده رجوع کنیم، نیاز به متوقف کردن پردازنده براي انجام عملیات حافظه نیست اما برای مراجعه به حافظه خارجی باید عملیات برای چند سیکل متوقف میشد. حافظه ی نهان که قرار است طراحی کنیم ویژگی های زیر را دارد:

دو طرفه باشد (two way)

اندازه هر کلمه: 32 بیت

اندازه هر بلاك: 64 بیت (2 کلمه)

تعداد مجموعه ها(64 :)set

اندازه حافظه نهان: 1 کیلو بایت براي داده (همچنین حدود 1 کیلو بایت هم براي نشانه‌ها و غیره مورد نیاز است.)

گذرگاه آدرس: 19 بیت

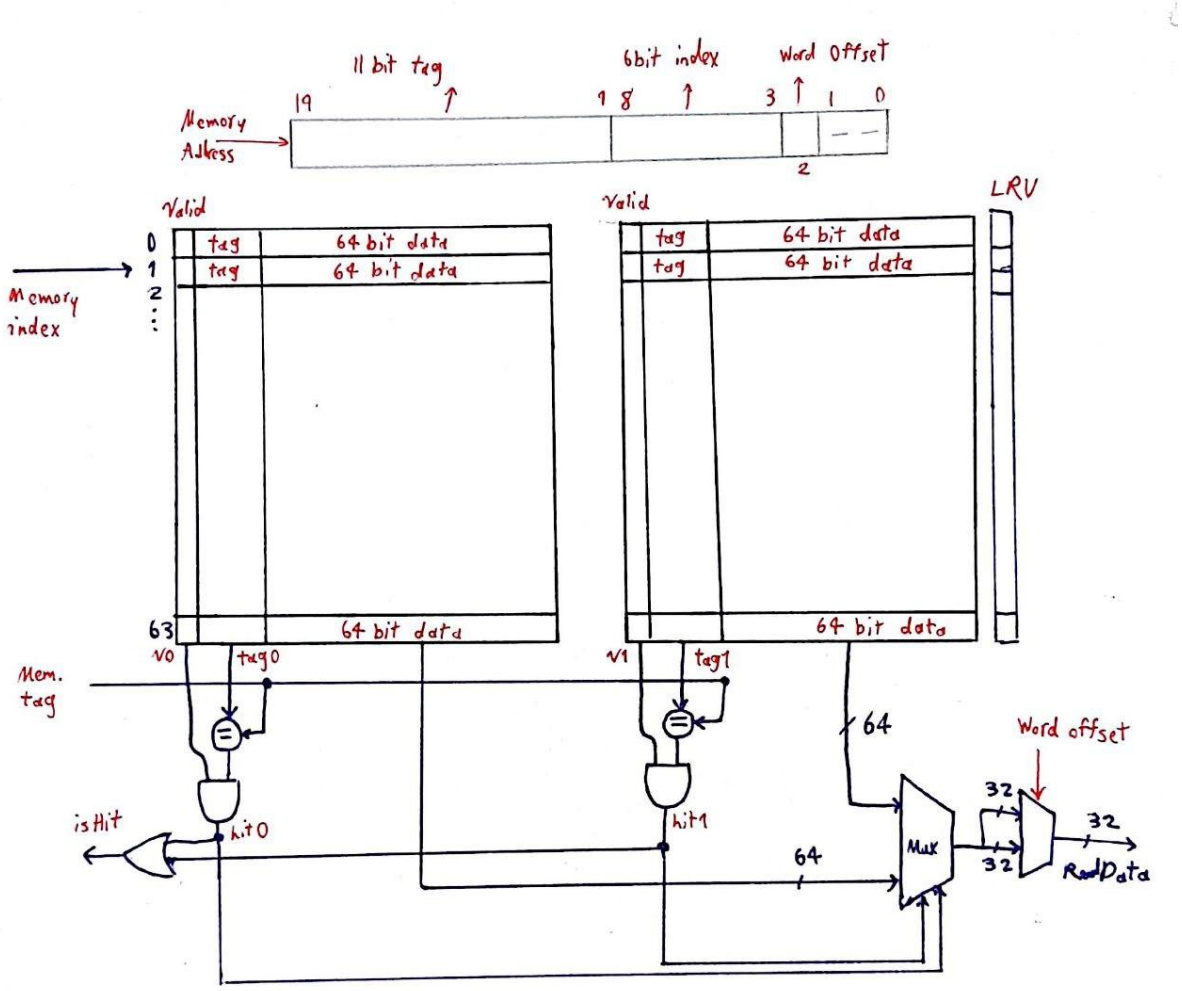
تعداد بیت مورد نیاز برای نشانه (10 :)tag بیت

تعداد بیت شاخص (6 :)index بیت داراي بیت اعتبار (valid)براي هر بلاك هر مجموعه نیاز به یک بیت (براي used) سیاست جایگزینی LRU دارد.

از الگوریتم LRU استفاده میکنیم :در صورتی که این بیت 0 باشد یعنی آخرین بار داده در cache0 نوشته شده و داده ی جدید در cache1 نوشته میشود. پس از آن مقدار بیت LRU برابر 1 میشود.

توضیحات cache:

شکل زیر شمای کلی حافظه نهان را نشان میدهد:



طریق کار حافظه نهان:

هنگامی که درخواست از پردازنده ارسال میشود، در حافظه نهان به دنبال داده خواسته شده میگردیم.اگر داده یافت شد hit=1 میشود و با بیت های valid متوجه میشویم که از کدام ستون باید داده بخوانیم. (سیگنال های کنترل ماکس). اما در صورتی که در حافظه نهان داده موجود نباشد hit=0 و مجبور هستیم پردازنده را freeze کنیم و به حافظه اصلی خارج از پردازنده مراجعه نماییم.در صورت miss شدن زمان مورد نیاز از مراجعه خالی به SRAM بیشتر است. تفاوت راندمان کلی بستگی به نرخ hit شدن دارد. در این آزمایش با فرض آن که برای SRAM شش کلاک در نظر گرفته باشیم.سرعت و کارایی با حافظه نهان پایین تر آمده است.

ماژول حافظه نهان با نام cache در فایل cache.v قرار دارد.

کارکرد cache:

کلاک و ریست مانند سایر ماژول ها است و وقتی ریست 1 شود عملیات پاک میشود.

LRU_update: وقتی 1 بشود مقدار یعنی باید مقدار LRU تغییر یابد و نشان میدهد که آخرین بار از کدام ستون استفاده شده است.

Write en: وقتی 1 بشود در لبه ی بالا رونده ی کلاک در cache مقدار writeData نوشته میشود.

Invalidate: برای صفر کردن بیت valid مربوط به خانه ای است که قرار است داده جدید در آن خانه در حافظه ی اصلی نوشته شود.

Read data: داده 32 بیتی خوانده شده از cache، به صورت asynchronous

Writedata: دو تا داده 32 بیتی کنار هم قرار میگیرند و در یک بلاک از حافظه نوشته میشود.

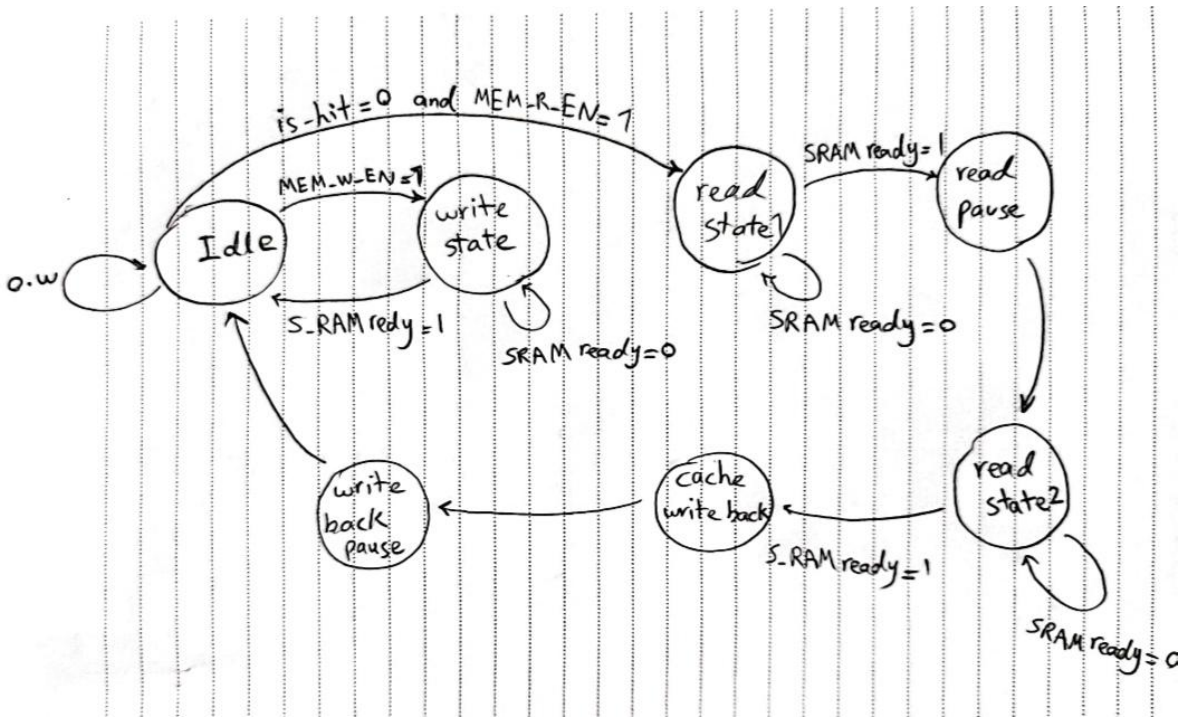
LRU: نشانگر ستونی که آخرین بار استفاده شده

Offset: برای انتخاب یکی از دو word ذخیره شده در data block

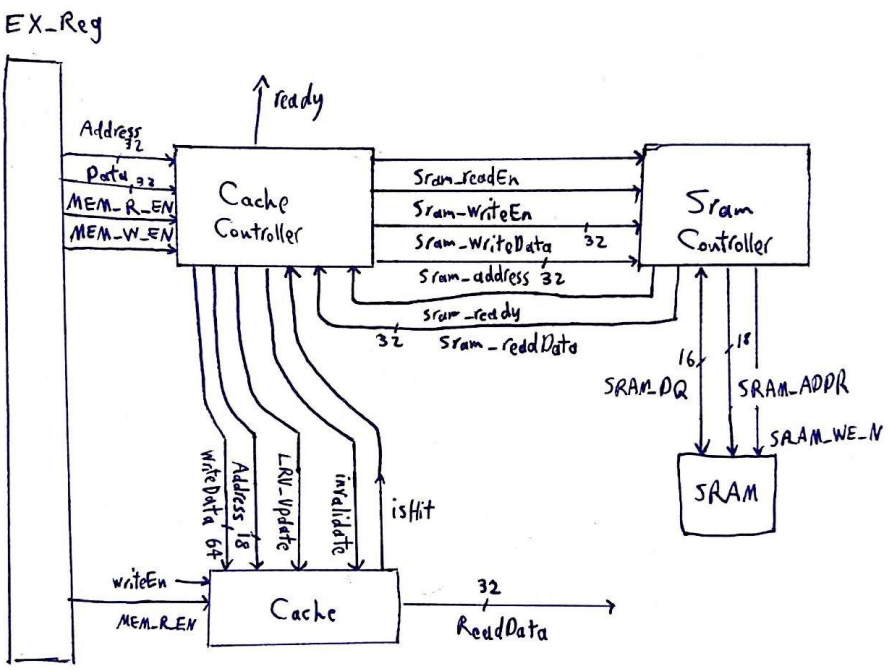
isHit: اگر داده در حافظه نهان باشد 1 میشود و اگر نباشد 0 میشود و باید برای داده به SRAM مراجعه کرد. این متغیر حاصل or دو متغیر hit0و hit1 است. یعنی اگر در هر کدام از ستون ها داده یافت شد اعلام میشود. Hit0و hit1 از تطبیق tag ها و valid به دست می آید.

توضیحات cache controller:

استیت ماشین آن به شکل زیر است:



شمای کلی مرحله MEM به شکل زیر است:



هنگام عملیات نوشتن در حافظه، تنها وظیفه ی مجموعه ی cache و controller آن این است که اگر آدرسی که قرار است در SRAM نوشته شود، در cache موجود بود، invalidate شود، پس invalidate به صورت زیر تعریف می‌شود:

```
Invalidate = 1 if present state = idle and next state = writeState
```

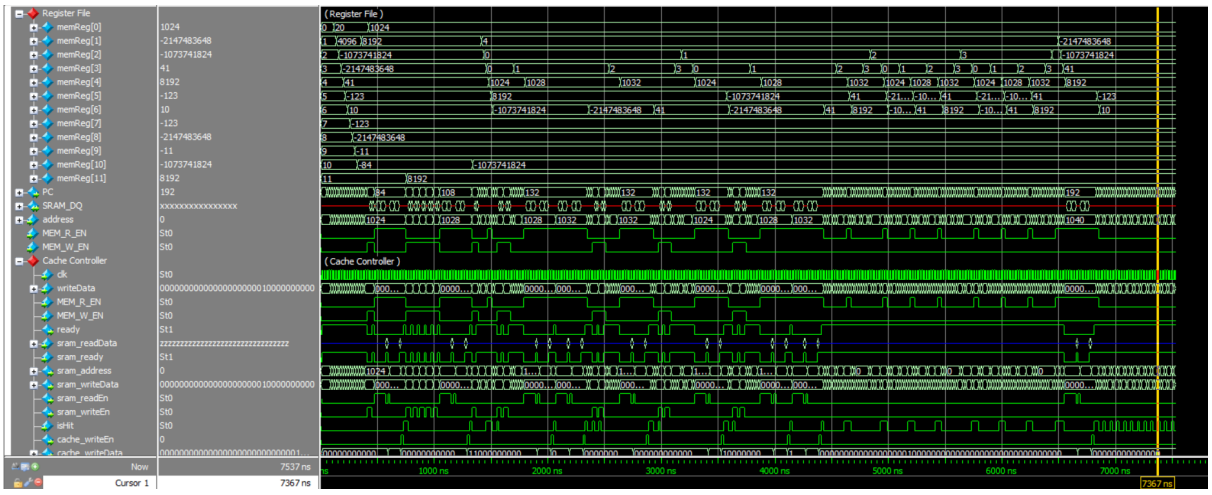
هنگام عملیات خواندن باید بررسی شود که آیا داده در cache موجود است یا خیر؟ که یعنی isHit = 1 یا خیر، اگر موجود باشد، LRU_Update فعال می‌شود و مقدار LRU اشاره کننده به آدرس مربوط، update میشود و در همان state باقی می‌ماند؛ اگرداده مورد نظر در cache یافت نشود، باید cache controller فرمان خواندن داده ی مورد نظر و داده ی دیگر با word offset متفاوت را به sram controller را بدهد، در دو state که readState1 و readState2 نام دارند این کار انجام می‌شود، تا زمانی که sram_ready دریافت شود در همین state ها باقی می‌ماند و با ready شدن sram، دیتای 32 بیتی در پورت sram_readData را در دو رجیستر 32 بیتی ذخیره می‌کند و سپس در cacheWriteBack دو 32 بیت را در writeData ی cache قرار داده و در آدرس مربوطه می‌نویسد. استیت های pause برای جلوگیری از رخ دادن هازارد داده ای قرار داده شده اند، این استیت های اضافی و پیاده سازی بسیار ساده شده ی cache در کنار عملکرد سریع SRAM بر خلاف حافظه های اصلی واقعی، (و کد محک که hit rate خیلی پایینی دارد) باعث شد بر خلاف انتظار سرعت پردازنده در مقایسه با حالت قبل که cache وجود نداشت کاهش یابد.

شبیه سازی Modelsim و تست بر روی FPGA:

برای بررسی صحت کارکرد ابتدا پردازنده را با استفاده از یک ماژول که رفتار SRAM را mimic می‌کند توسط یک testbench با نام testbench_cache شبیه سازی کرده و سپس در صورت مطلوب بودن شبیه سازی سراغ تست بر روی FPGA و درستی سنجی با استفاده از signaltap می‌رویم.

تست Modelsim

شکل wave بدست آمده به شکل زیر است:



که همان نتایج مطلوب کد محک استفاده شده را می‌دهد.

تست SignalTap:

پس از سنتز و پروگرم شدن بورد، نتایج زیر بدست می آیند:

مقدار های سیگنال های isHit و nv ،ready در کنار sram_readEn و sram_writeEn و SRAM_DQ به نمایش درآمده اند و مقادیر رجیستر فایل نیز برابر با مقادیر برنامه محک است.