به نام خدا

دلیل اینکه در این ساختار نیازی به offset نداریم این است که بلاک های ما 1 خطی هستند و به عبارتی بیت b که محل نسبی داده در بلاک را مشخص میکند برابر 0 خواهد بود. محل نسبی داده در بلاک همواره همان 1 خط موجود در بلاک است و دیگر نیازی به تصمیم گیری برای نشاندن داده در کدامین خط بلاک نخواهیم داشت.

در زیر به شرح هر یک از ماژول خواسته شده میپردازیم

Data_array :

در این ماژول داده های کش را نگه داری میکنیم به این صورت که این آرایه خط داده دارد و باید 2 تا از این آرایه ها در ماژول cache داشته باشیم.

کاری که در بدنه ی این ماژول انجام میدهیم این است که اگر wren که همان فرمان نوشتن در کش است فعال شود داده ی ورودی این ماژول که 16 بیتی است در آدرسی که به عنوان ورودی به این ماژول داده میشود ثبت میشود.

این ماژول یک آرایه ی 64 در 16 دارد که هر یک از این 64 خط یک داده را مشخص میکنند که هر کدام از این داده ها نیز خود 16 بیتی هستند

در انتها خروجی این ماژول آن خطی از این آرایه ی 64 در 16 تایی است که آدرس آن توسط ورودی address داده شده

data <= memory(to_integer(unsigned(address))) ;

در این خط کد data خروجی این ماژول و memory آرایه ی 2 بعدی و address آدرس ورودی این ماژول هستند.

Valid_tag_array:

در این ماژول هدف ما نگه داری بیت های تگ و ولید است

این ماژول ورودی wrdata را دارد که همان تگ ورودی است که درصورت 1 بودن wren در آرایه ی 2 بعدی موجود در این ماژول نوشته خواهد شد

این آرایه ی 2 بعدی شامل 64 خط دیتای 5 بیتی می باشد که چپ ترین بیت آن مربوط به valid و 4 بیت دیگر مربوط به tag میباشند

اگر 0 wren باشد در این آرایه ی 2 بعدی چیزی نوشته نمیشود و تنها خروجی output مساوی آن خط از این آرایه ی 2 بعدی که آدرس آن به عنوان ورودی به این ماژول داده میشود میگردد

این ماژول همچنین ورودی invaliddata دارد که در صورت 1 بودن باید بیت ولید آن خط از این آرایه ی 2 بعدی که آدرس آن داده شده را برابر 0 کند

این اتفاق زمانی میفتد که فرمان نوشتن در ram را داشته باشیم و آن داده ای که در رم در حال جایگزین شدن است در کش موجود باشد

این ماژول همچنین یک ورودی reset دارد که درصورت 1 بودن تمامی بیت های ولید و تگ این 64 خط را مساوی 0 خواهد کرد

ما برای پیاده سازی این پروژه باید 2 تا از این ماژول را در ماژول cache تعریف و استفاده کنیم نمونه ی کد این ماژول در صورت 1 بودن wren برای نوشتن در این ماژول عبارت است از

```
tag_memory (to_integer(unsigned(address)))(4) <= '1';

tag_memory (to_integer(unsigned(address)))(0) <= wrdata(0);

tag_memory (to_integer(unsigned(address)))(1) <= wrdata(1);

tag_memory (to_integer(unsigned(address)))(2) <= wrdata(2);

tag_memory (to_integer(unsigned(address)))(3) <= wrdata(3);

data <= tag_memory (to_integer(unsigned(address)));
```

Miss_hit:

این ماژول 3 input دارد

مقدار tag آن که توسط cpu مشخص و درخواست شده است

اما 2 مقدار w0 و w1 آن خروجی 2 ماژول valid_tag_array موجود در ماژول کلی cache هستند

در این ماژول 4 بیت سمت راست w0 با tag مقایسه میشود و درصورت مساوی بودن این 4 بیت و برابر بودن بودن بیت 5 ام w0 با 1 مطابقت رخ و تشخیص داده میشود و w0_valid برابر با 1 میگردد

به صورت مشابه همین اتفاق برای w1 نیز خواهد افتاد در صورت 1 بودن یکی از 2 بیت خروجی w1_valid و یا w0_valid مقدار hit برابر با 1 میگردد و درغیر

این صورت مقدار hit مساوی 0 میشود اما یک اشکالی که به این ماژول وارد بود این بود که نمیتوان مقدار خروجی را خواند و نمیتوان تشخیص داد که

آیا w0_valid یا w1_valid برابر 1 هستند یا نه

پس برای رفع این مشکل 2 سیگنال w0_signal و w1_signal را تعریف کردیم و هر کجا هر مقداری که w0_valid و w1_valid میگرفتند این مقدار را به این 2 سیگنال نیز دادیم تا در انتهای برای مشخص کردن مقدار hit مشکلی نداشته باشیم و با تشخیص مقدار این 2 سیگنال مقدار hit نیز مشخص میشود

Mru:

نام گذاری شده است وظیفه ی انتخاب یکی از mru که با نام most recenty updatad ماژول پر هستند را برعهده دارد. نحوه ی عملکرد این way برای جایگزینی در زمانی که هر 2 way 2 ای را انتخاب میکند که داده ی آن مدت زمانی کمتری است که way ماژول به این گونه است که update شده است.

برای اینکه مطمین شویم که باید جایگزینی رخ دهد یا اولین بار است که داده نوشته میشود دو آرایه ی 64 تایی را انتخاب کرده و مقدار اولیه ی 0 به همه ی آن ها میدهیم و اگر یکی از way ها خالی بود ومقدار این آرایه ها 0 بود در آن مینویسیم.در غیر این صورت به سراغ جایگزینی میرویم. یک روش معمول برای پیاده سازی این سیاست که غالبا استفاده میشود این است که یک آرایه ی 64 تایی را انتخاب کرده و مقدار اولیه ی 0 به همه ی آن ها بدهیم و هر بار که داده ای در cache نوشته و یا از آن حذف شد مقدار یکی از این 64 تا (که توسط آدرس مشخص میشود) را به خروجی این ماژول میدهیم. حال این خروجی را به wren1 و متمم آن را به wren0 میدهیم.

این ماژول را به یک متغیر حساس میکنیم و در صورت نوشته شدن و یا حذف شدن هر داده ای این متغیر را 1 بار تغییر میدهیم تا این ماژول 1 بار دیگر مجددا فراخوانی گردد

به این صورت همواره wren یکی از این 2 way که برای مدت زمانی کمتری است که دست خورده است فعال میشود

Cache :

در این ماژول تعدادی ورودی و خروجی را به شرح زیر مشخص کرده ایم! ورودی ها عبارتند از

clk,hole_write,write,reset, cache_request :in std_logic ;

address:in std_logic_vector(9 downto 0);
data_in :in std_logic_vector(31 downto 0);

خروجی ها عبارتند از

hit , cache_ready: out std_logic;
data_out : out std_logic_vector(31 downto 0)

برای مشخص کردن وضعیت write در ram از ورودی hole_write استفاده میکنیم هر گاه این ورودی برابر 1 بود یعنی تنها دلیل ما برای مراجعه به cache قرار دادن مقدار invaliddata1 یا invaliddata2 برابر 1 (در صورت موجود بودن داده ای که قرار است داده ی دیگری جایگزین آن گردد ، در کش) است

یعنی میخواهیم داده ای را که داده ی جدید جایگزین آن شده است از cache حذف کنیم ورودی بعدی write است که در صورت 1 بودن ، mru صدا زده میشود و خروجی mru به wren0 و

wren1 داده میشود! (همان طور که در شکل نشان داده شده است) و مقدار data_in در آدرس ورودی (address) یکی از way هایی که توسط mru انتخاب شده

است نوشته میشود. در صورت 0 بودن write داده ای در کش نوشته نشده و فقط منتظر جواب ماژول hit_miss خواهیم ماند.

مقدار reset نیز برای ریست کردن ماژول valid_tag_array به کار میرود و هر بار که reset مساوی 1 گردد تمامی داده های داخل کش با 0 شدن بیت valid و tag از داخل کش پاک میشوند

ورودی آخر این ماژول cache_request است که توسط controller به این ماژول ارسال میگردد.

هر بار که controller نیاز به خروجی cache داشت مقدار cache_request برابر 1 میگردد و پس از اتمام کار cache و آماده شدن خروجی این ورودی در controller برابر 0 میگردد

همانطور که مشاهده میکنید یکی از خروجی های این ماژول hit است که همان خروجی ماژول miss_hit خواهد بود.

خروجی دیگر این ماژول cache_ready است که درصورت آماده شدن جواب کش این مقدار برابر 1 میگردد و در کنترلر هر بار که این مقدار برابر 1 بود یعنی کار کش به اتمام رسیده و خروجی حاضر شده است

خروجی data_out این ماژول نیز مقدار دیتایی که آدرس آن توسط cpu درخواست شده است را برمیگرداند.

در این ماژول ما 2 ماژول data_array 2 ماژول valid_tag_array یک ماژول hit_miss و یک ماژول mru را ایجاد میکنیم.در این ماژول ما از یک state machine استفاده کرده ایم

زیرا برا مشخص شدن جواب نیاز داریم تا به جای 1 کلاک چندین کلاک طول بکشد(برای مثال برای آماده شدن ورودی miss_hit ما نیاز داریم تا خروجی valid_tag_array آماده شده باشد که خود نیاز به گذشت یک کلاک دارد

در این استیت ماشین ما برای آماده شدن خروجی نیازمند گذشت 4 کلاک و گذر از 5 استیت هستیم و در هر یک از این استیت ها با آماده شدن خروجی مورد نظر آن را به عنوان ورودی به ماژول دلخواه میدهیم

برای مثال منتظر میمانیم تا جواب mru آماده شود و تکلیف wren1 و wren2 مشخص گردد و آن 2 را به ماژول های valid_tag_array و data_array میدهیم

حال منتظر میمانیم تا به اندازه ی 1 کلاک زمان بگذرد و جواب valid_tag_array ها را به ماژول hit_miss دهیم و در صورت hit بودن مشخص میکنیم که خروجی کدام data_array را به عنوان خروجی اصلی انتخاب کنیم.

و وقتی جواب ماژول hit_miss آماده شد مقدار cache_ready را مساوی 1 میکنیم و به ابتدای کار باز میگردیم

Controller :

در این قسمت که کنترلر کننده ی حافظه ی نهان نام دارد تمام کار کنترلر که روابط بین کش و رم را مشخص میسازد با یک fsm توضیح میدهیم

شرح کامل وظایف و ویژگی های این کنترلر در تعریف پروژه آماده است ما برای انجام این وظایف یک fsm با 5 state را طراحی کرده ایم که عبارتند از :

استیت s0 : اگر در این استیت بودیم تحت هر شرایطی به استیت s1 میرویم این استیت شروع کننده ی fsm ما خواهد بود

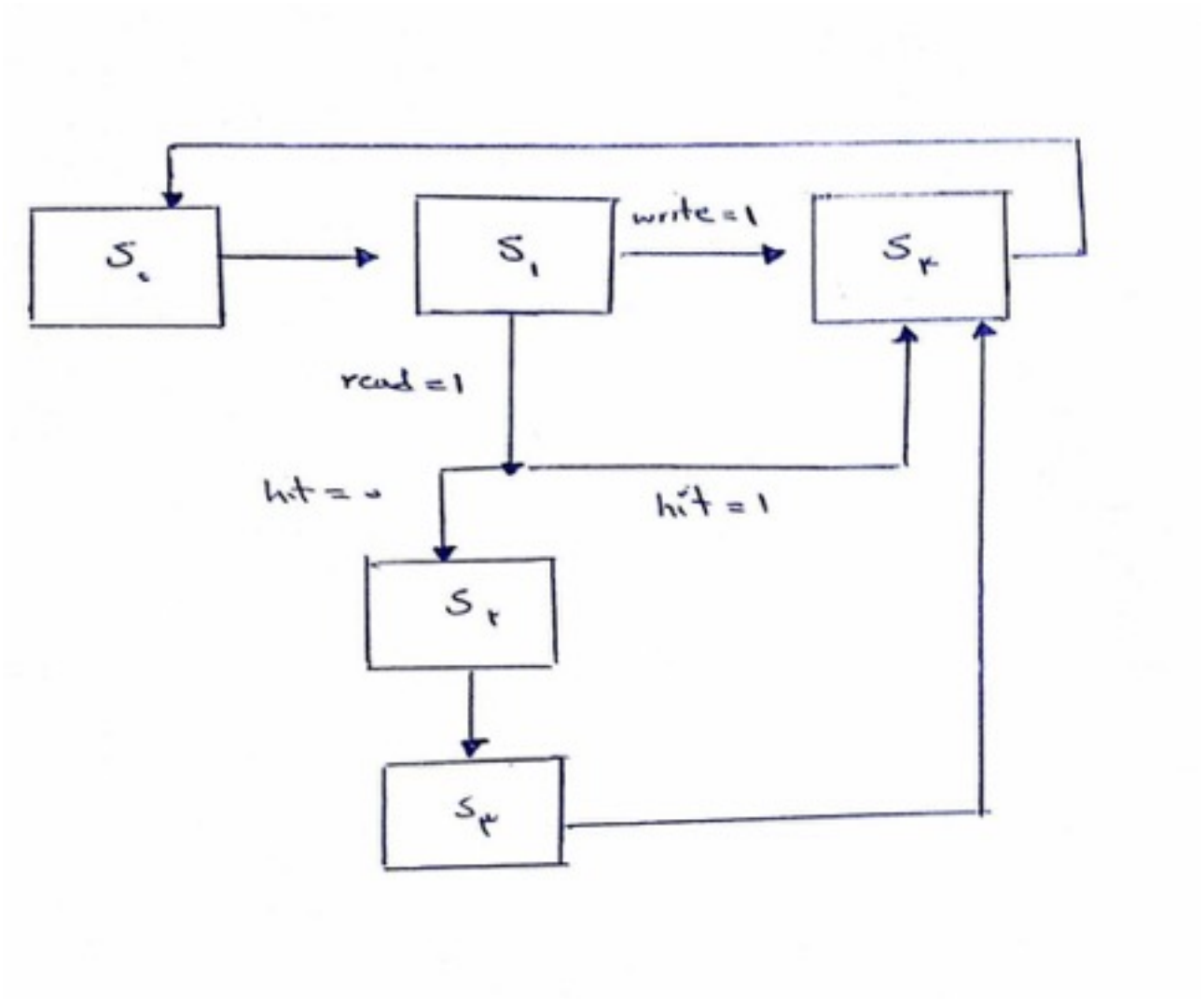
استیت s1 : در این استیت اگر write که توسط cpu درخواست داده میشود برابر 1 بود ما باید داده ی ورودی را در ram نوشته و برای cache درخواست اجرا بفرستیم و در cache بیت valid را برابر 0 کرده و هنگامی که کار کش تمام شد به استیت s4 خواهیم رفت

اما اگر در این استیت درخواست read توسط cpu داده شود به cache درخواست اجرا شدن فرستاده و هنگامی که پاسخ cache آماده بود اگر مقدار hit این پاسخ برابر 1 بود به استیت s4 و اگر برابر صفر بود به استیت s2 میرویم

استیت s2 : در این استیت منتظر جواب رم خواهیم ماند و در صورت آماده شدن جواب رم cache_write را برابر 1 قرار میدهیم و به استیت s3 میرویم

استیت s3 : در این استیت درخواستی به cache فرستاده و بعد از آماده شدن جواب به استیت s4 میرویم

استیت s4 : که استیت نهایی و دلخواد ماست مقدار finish که پایان کار کنترلر است را برابر 1 میکند و به استیت s0 رفته و منتظر درخواست بعدی خواهد ماند شکل این ماژول (statediagram) در زیر آمده است:



Main :

در این ماژول که ماژول کلی و نهایی است تعدادی ورودی و خروجی خواهیم داشت که در تعریف پروژه مشخص شده اند

در این ماژول ما 3 ماژول cache و ram و controller را map میکنیم و ورودی ها و خروجی های آن را به نحو زیر مشخص میکنیم تا controller ارتباط درستی بین این 2 ماژول برقرار کرده و به نحوی بازوی پروژه باشد

در این ماژول برای اینکه حرکات کنترلر تحت نظر باشد

این ماژول 1 خروجی finish دارد که هر بار برابر 1 شد یعنی کار کنترلر و به عبارتی کار درخواست cpu تمام شده است و ما باید منتظر آمادن داده ی جدید از سمت کنترلر و فرمان جدید باشیم که برای این منظور 1 متغیر به نام c تعریف کرده ایم که با هر بار تغییر address و یا write و یا data_in تغییر کرده

و با هر بار تغییر c مقدار request برای کنترلر برابر 1 میگردد و با هر بار مساوی شدن finish با 1 مقدار request 0 میگردد