به نام خدا

پروژه درس معماری کامپیوتر

منتورهای پروژه: رضا علیدوست و زهرا امیری

اعضای گروه7:

سینا علی‌نژاد (99521469)

ریحانه هاشم زاده (99400010)

نوید ابراهیمی (99521001)

تیر 1401

بخش Processor

این بخش نماد cpu در road map ماست که در اینجا Virtual Address تولید میشود و سپس با استفاده از کنترلر مرکزی این VAها به TLB داده خواهند شد.

در هر مرحله که کنترلر این کامپوننت را صدا میزد و یک VA توسط این کامپوننت ریترن میشود.

Input: index: integer

Output: vr_add: std_logic (16bits)

Architecture:

Index در این قسمت وظیفه مشخص کردن VAیی که باید به CONTROLLER پاس داده شود را دارد. این VARIABLE در کنترلر یکی یکی اضافه و به اینجا داده میشود.

VR_ADD خروجی ماست که عنصر V_ADDR[INDEX] را به کنترلر باز میگرداند.

بخش TLB:

با استفاده از VAهایی که ورودی میگیرد اگر PHYSICAL ADDRESS را دارا بود، PA را به CACHE میدهد و درغیر این صورت TLB_MISS 1 شده و به کنترلر اعلام میشود که این VA در TLB موجود نمیباشد.

INPUT:

UPDATE_TLB: 1BIT – IF UPDATE_TLB == 0 => UPDATE ELSE READ FROM TLB

Vr_add: 16Bits: std_logic_logic

Write_ppn: 4Bits: std_logic_logic

OUTPUT:

Physical_add: 11Bits: std_logic_vector

TLB_Miss: 1Bit: Bit => IF Tlb_Miss == 0 => Found Else Miss

Done: integer: => Signal for Controller

همانطور که در داک گفته شد باید دو مدل TLB را پیاده سازی کنیم که یکی full و دیگری four-way میباشد.

A) Fully

در ابتدا tlb_miss را یک میگذاریم و این مورد صفر نمیشود مگر اینکه داده موردنظر پیدا شود. در این صورت مقدارش را به صفر تغییر میدهیم.

یک سیگنال به عنوان COUNTER تعریف میکنیم که نشان دهنده سطری است که میخواهیم آدرس را در آن سطر بنویسیم و هردفعه که آدرس نوشته شود، یکی به آن اضافه میشود تا سطر خالی را به ما

بدهد. به دلیل 48سطری بودن TLB ، بعد از اضافه کردن مقدار آن، یک باقی مانده به 48 نیز میگیریم تا از ظرفیت TLB بیشتر نشود.(FIFO است)

Update_tlb مشخص کننده کاری است که این کامپوننت قرار است برای ما انجام دهد. حال اگر برابر یک باشد، باید یک سطر tlb شامل valid bit، VPN و PPN داخل سطر خالی که توسط COUNTER مشخص میشود، نوشته شود.

و اگر برابر صفر باشد، یعنی باید تک تک سطرهای TLB پیمایش شوند تا اگر VALID BIT آن یک و TAG آن سطر برابر VPNی که توسط کنترلر به این قطعه داده میشود باشد، داده موردنظر یافت شده و داخل Physical_Add قسمت tag آن و page offsetپی که از اول داشتیم را قرار میدهیم و چون داده پیدا کرده‌ایم، tlb_miss صفر میشود.

حال که اجرای این کامپوننت تمام شده است، done را تغییر داده و خارج میشویم.

B) Four-Way

چون در اینجا از روش four_way استفاده میکنیم و 48 Entry نیز داریم، 12ست داریم که هرکدام از این ست‌ها 4سلول دارد که از 0 تا 3 نامگذاری میشوند. برای مشخص کردن Setپی که باید برای نوشتن درآن به آنجا برویم از vpn_mod و برای تعیین اینکه در کدام سلول بنویسیم از counter استفاده میکنیم و هرکدام از ست‌ها یک counter مختص به خود را دارد که این counterها در آرایه Counters ذخیره میشوند.

اگر update_tlb برابر یک باشد، کاری که انجام میدهیم write است. با توجه به counters(vpn_mod) که مشخص کننده یکی از سلول‌های ما (یک tlb خاص) است، حال که مکان آن مشخص شد، یک سطر TLB شامل valid bit، VPN و PPN داخل سطر خالی که توسط COUNTER مشخص میشود، نوشته شود و در ادامه counter یکی اضافه میشود و بخاطر 4 way بودن نوع TLB، یک باقی مانده بر 4 میگیریم.

اگر update_tlb برابر صفر باشد، باید عمل read انجام دهیم که در آن باید تمامی سلول‌های set را پیمایش کنیم. در اینجا چک میکنیم اگر vpn هرسطر برابر vpn دریافتی از کنترلر باشد، آن داده یافت شده و Physical Address که شامل VPN و PO است را خروجی میگیریم و سپس tlb_miss برابر با صفر میشود.

حال که اجرای این کامپوننت تمام شده است، done را یک میکنیم و خارج میشویم.

بخش page table:

این بخش زمانی اجرا میشد که آدرس VA در TLB موجود نبود و MISS میشد و ما برای پیدا کردن PA به یک سطح پایین‌تر یعنی PAGE TABLE میرویم. اگر در VA PAGE TABLE پیدا شود، HIT رخ میدهد به PA را به TLB برمیگردانیم ولی اگر PAGE FAULT رخ دهد، page_fault یک میشود و آن را به کنترلر اعلام میکنیم.

INPUT:

Vpn: 9Bits: std_logic_vector

Update_pagetable: integer:=0

Write_ppn_from_RAM:integer

OUTPUT:

Ppn: 4Bits:std_logic_vector

Page_fault:1Bit:=’0’

Done:integer:=’0’

Architecture:

اگر update_pagetable برابر یک باشد، باید write انجام دهیم، پس برای قسمت ppn، sppn که از RAM آمده است به همراه یک 1 به عنوان VALID BIT در آن مینویسیم.

ولی اگر صفر باشد، باید read انجام دهیم. اگر valid bit صفر باشد، page fault رخ میدهد و آن را به controller اعلام میکنیم ولی اگر موجود بود، 4بیت اول را که همان ppn هم را در خروجی مینویسیم.

حال که اجرای این کامپوننت تمام شده است، done را تغییر میدهیم و خارج میشویم.

بخش cache:

یکی از یونیت هایی که داریم Cache هست که داخل این پروژه دو مدل دارد و در کل کاری که انجام میدهد این است که وقتی TLB هیت شد ،داخل Cache به دنبال tag موردنظر میگردد و اگر داخل Cache بود Hit میشود و دیتا را برای Processor میفرستد و اگه miss شد، آدرس را به RAM میدهد و وقتی که دیتا پیدا شد از RAM میگیرد و داخل Cache مینویسد و بعد به دیتا را Processor میدهد.

INPUT:

Write: integer=> if write == 0 => read data from cache else write data to cache

Datain: memory_block

Addr: 11Bits: std_logic_vector

OUTPUT:

Dataout: 32Bits: std_logic_vector

Cache_miss: 1Bit => 0 => found, 1=>miss

Done: integer

Architecture:

همانطور که در داک گفته شد باید دو مدل cache را پیاده سازی کنیم که یکی directed و دیگری 2-way میباشد.

1) Directed:

کم ارزشترین دو بیت آدرس برای byte offset میگذاریم که دلیل آن 4بایتی بودن wordهاست. هربلاک دو word است، پس سومین بیت کم ارزش برای نمایش آن است. همچنین cache 32ورودی میگیرد، پس به 5 بیت برای نمایش آن (بیت 3 تا 7) نیاز داریم. پس 3بیت باقی مانده برای tag است.

اگر write برابر یک باشد، عمل write را انجام میدهیم. یعنی در سطری که با index مشخص شده dataیی که از ورودی گرفتیم میریزیم و در قسمت tag، Tag Address را مینویسیم و در آرایه valids valid مربوطه را 1 میکنیم.

اگر write برابر صفر باشد، عمل read را انجام میدهیم. یعنی اگر tagیی که از آدرس استخراج کردیم با tag خانه indexم cache برابر باشد و valid bit 1 باشد، word موردنظر از خانه indexم cache را با word offset انتخاب کرده و خروجی میدهیم و cache_miss را صفر میکنیم. اگر داده موردنظر پیدا نشد، cache_miss را یک میگذاریم.

حال که اجرای این کامپوننت تمام شده است، done را تغییر میدهیم و خارج میشویم.

2) 2-Way

کم ارزشترین دو بیت آدرس برای byte offset میگذاریم که دلیل آن 4بایتی بودن wordهاست. هربلاک دو word است، پس سومین بیت کم ارزش برای نمایش آن است. همچنین cache 32ورودی میگیرد که به دلیل 2-way بودن، 16ست داریم که با 4بیت برای نمایش آن (بیت 3 تا 6) نیاز داریم. پس 4بیت باقی مانده برای tag است.

چون در اینجا از روش 2_way استفاده میکنیم و 32 Entry نیز داریم، 16ست داریم که هرکدام از این ستها 2سلول دارد که 0 و 1 نامگذاری میشوند. برای مشخص کردن Setیی که باید برای نوشتن درآن به آنجا برویم از index و برای تعیین اینکه در کدام سلول بنویسیم از counter استفاده میکنیم و هرکدام از ستها یک counter مختص به خود را دارد که این counterها در Counters ذخیره میشوند. که

هر wayیِ tag متناظر با خود را دارد که در یک آرایه نگهداری میشود. تگ‌های مربوط به way0 در tags0 و تگ های مربوط به way1 در tags1 نگهداری میشوند.

اگر write برابر یک باشد، کاری که انجام میدهیم write است. حال با توجه به counters(index) که مشخص کننده یکی از سلول‌های ما (یک way خاص) است، حال که مکان آن مشخص شد، datain که از ورودی گرفته شده داخل way(i)(index) و tagیِ که از آدرس استخراج شد، داخل tag(i)(index) میگذاریم. همچنین valid bit را یک میکنیم. در آخر counter را یکی زیاد کرده و به دلیل 2-way بودن بر 2 باقی‌مانده میگیریم.

اگر write برابر صفر باشد، باید عمل read انجام دهیم که در آن چک میکنیم tagیِ که از آدرس استخراج کردیم با tag در ست موردنظر برابر بوده و valid bit نیز 1 باشد، word موردنظر از خانه pindex way(i) را با word offset انتخاب کرده و خروجی میدهیم و cache_miss را صفر میکنیم. اگر داده موردنظر در 2مجموعه پیدا نشد، cache_miss را یک میگذاریم.

حال که اجرای این کامپوننت تمام شده است، done را تغییر داده و خارج میشویم.

بخش Main Memory:

در دو صورت به این بخش مراجعه میشود.

1- TLB Hit => Cache Miss => finding data and writing to cache
2- TLB Miss => Page Table Hit => TLB Update => Cache Miss => finding data and writing to cache
3- TLB Miss => Page Table Miss => Hard-Disk => Writing to RAM

INPUT:

W_bit: 1Bit => 0 => read from memory and write in cache else write from hard-disk in memory

Ph_Add: 11Bits: std_logic_vector

Write_data_from_disk: page_type

OUTPUT:

Read_data: memory_block

Written_ppn: integer

Done:integer

در داک گفته شده RAM از 512 ورد تشکیل شده است.

سایز page table به صورت

(4+1)*2^9/2^3 * 2^2 = 5 * 2^4 = 80 word

به دست می آید.

این سایز برای page table است و برای بدست آوردن فضای ذخیره سازی دیتا در RAM

512 – 80 = 432 word

انجام میدهیم. حال برای بدست آوردن تعداد pageهایی که در این فضای باقی‌مانده جا میگیرد،

432/80 = 13

را انجام میدهیم. پس 13پیج داخل RAM جای میگیرد.

چون page size = 128 است، پس به 7 بیت برای نمایش Page Offset نیاز داریم. از آنجایی که 16 پیج در RAM موجود است، پس 4بیت برای نشان دادن پیج یا همان PPN کافی است.

اگر w_bit برابر یک شد، عمل write را انجام میدهیم.

به این صورت که در اولین page خالی RAM که با Counter مشخص میشود دیتایی که از دیسک آمده را مینویسیم. سپس ppn را با استفاده از counter مقداردهی میکنیم (چون counter شماره pageپی که دیتا در آن ذخیره شده را دارد و ppn نیز همین کار را میکند، اینکار را انجام میدهیم).

همچنین برای اطلاع کنترلر برای شماره pageپی که دیتا در آن ذخیره شده، counter را در written_ppn میریزیم. در آخر counter را یکی زیاد کرده و به دلیل 13پیجی بودن RAM به 13 مود میگیریم.

در ادامه به دلیل بهینه‌تر شدن کد عمل READ را انجام میدهیم که از دوباره صدا زدن این کامپوننت جلوگیری کنیم.

نحوه کار به صورت استخراج page دلخواه توسط data(ppn) است (چون ppn شماره page دلخواه ما را داراست). چون هر page 32ورد است و هر بلاک هم 2 ورد، پس هر page 16بلاک دارد که با 4بیت قابل نمایش است. پس داخل block_offset (2 downto 5)physical را میریزیم. 2 بیت کم ارزش برای byte offset است.

بلاک موردنظر را با استفاده از خانه block_offsetم RAM استخراج کرده و به خروجی read_data میدهیم.

حال که اجرای این کامپوننت تمام شده است، done را یک میکنیم و خارج میشویم.

بخش hard-disk:

اگر RAM خطای Page Fault دهد، باید یک page جدید از hard-disk بیاوریم و داخل آن قرار دهیم.

INPUT:

call_hd : integer

S_add: 9Bits: std_logic_vector

OUTPUT:

Page: page_type

Done: integer

Architecture:

تغییر مقدار call_hd باعث شروع process میشود.

با استفاده از s_add، عضو ps_add hard-disk برای ریترن به خروجی page داده میشود.

حال که اجرای این کامپوننت تمام شده است، done را یک میکنیم باقیمانده بر 2 گرفته و خارج میشویم.

بخش Controller:

کنترلر نقش هسته مرکزی پروژه ما را برعهده دارد. به این صورت که ورودی‌های هر کامپوننت را مشخص کرده و مشخص میکند خروجیی که یک کامپوننت میدهد باید به کدام کامپوننت برود.

همچنین وظیفه ترتیب شروع کامپونت‌ها و مدیریت شروع شدن یک کامپوننت بعد از تمام شدن کامپوننت قبلی را برعهده دارد.

وظیفه مشخص کردن نوع cache و TLBیی که در ساختار Controller را نیز برعهده دارد.

INPUT:

tlb_version : integer := 1

cache_version : integer := 1

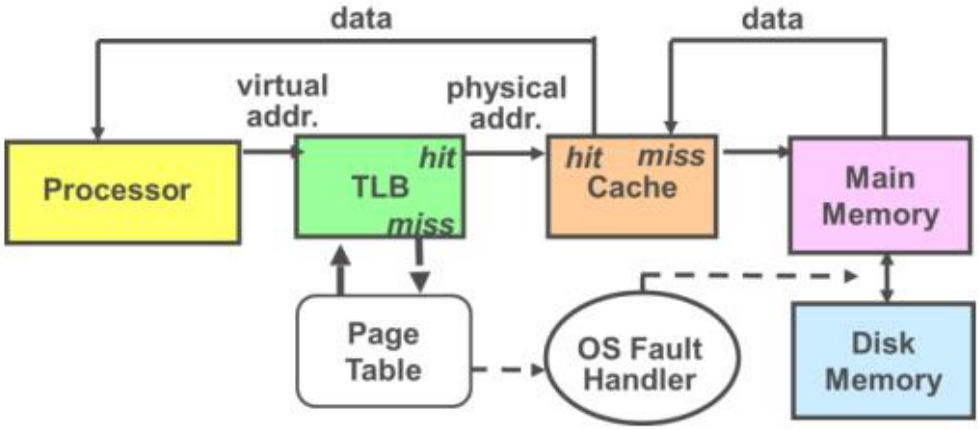
OUTPUT:

Output: 32Bits: std_logic_vector

Architecture:

از تمام entityهایی که ساختیم یک کامپوننت در Controller میسازیم. سپس از هریک Object گرفته و با سیگنال‌هایی که از پیش ساخته‌ایم مقداردهی/مقادیرشان را ذخیره میکنیم.

همه کامپونت‌ها موازی باهم درحال ران شدن هستند و با تغییر یک المان از آرگومانشان دوباره ران میشوند.



در ابتدا CPU تعدادی Virtual Address را تولید میکند. حال با استفاده از کامپوننت‌هایی که ساختیم، آن آدرس‌ها را یکی یکی به TLB میدهیم.

حال در TLB هستیم. اگر TLB_HIT = 1 شود، Physical Addressی که خروجی TLB است را به ورودی CACHE میدهیم. حال در Cache هستیم. اگر Cache_Miss = 0 شود، دیتا را به پراسسور برمیگردانیم و کار تمام میشود. در غیر این صورت با Physical Addressدیتا به سراغ Main Memory میرویم. چون Physical Address را داریم پس حتما Main Memory دیتا ما را دارد. حال دیتا را به Cache باز میگردانیم و در آن مینویسیم و از آنجا دیتا را به پراسسور میدهیم.

اگر در TLB miss داشتیم، به سراغ Page Table میرویم. اگر در اینجا با توجه به virtual address دیتای موردنظر یافت شد، آن را به TLB میبیریم و TLB را آپدیت کرده و روند پاراگراف قبل را انجام میدهیم. درغیر این صورت با استفاده از OS Fault Handler پیج موردنظر را از hard-disk بازگردانده و در RAM مینویسیم. حال در ادامه آدرسی که در RAM نوشتیم را در PT و TLB را آپدیت کرده و مطابق پاراگراف اول کار را ادامه میدهیم.