گزارش فاز دوم پروژه درس معماری کامپیوتر

پارمیدا جوادیان

پرنیان رضوی پور

دنیا نوابی

کیانا موسی زاده

دانشگاه صنعتی شریف

بهار 1401

همانطور که می‌دانید در دنیای واقعی ارتباط با حافظه با تاخیر همراه هست. هدف از این فاز تغییر فاز قبلی به گونه ای است که بخش ارتباط با حافظه آن با اضافه کردن تاخیر به واقعیت نزدیک شود. تاخیری که باید برای حافظه در نظر بگیریم 4 کلاک است به این معنی که پس از اجرای دستورات مربوط به حافظه باید 4 کلاک توقف کرده و سپس پردازش را ادامه دهیم.

اما همانطور که می‌دانید سرعت واحدهای پردازش مرکزی کامپیوترهای امروزی بسیار بالاست و حیف است که از این سرعت بالا بهره نبریم و بعد از هر دستور مربوط به حافظه 4 کلاک صبر کنیم. برای حل این مشکل می‌توان از حافظه نهان (cache) استفاده کرد.

حافظه پیاده سازی شده در این فاز 8 کیلوبایت ظرفیت دارد و اندازه هر بلوک آن یک word است(یعنی 32 بیت یا 4 بایت). این حافظه از نوع write back است و برای نگاشت اطلاعات روی آن از روش direct mapping استفاده شده است.

همانطور که می‌دانید ظرفیت حافظه نهان بسیار کمتر از ظرفیت حافظه اصلی است و همه اطلاعات موجود در حافظه اصلی، در حافظه نهان موجود نیستند. بنابراین در صورتی که اطلاعاتی از حافظه اصلی را بخواهیم، ممکن است در حافظه نهان موجود نباشد. در صورتی که این اطلاعات در حافظه نهان موجود بود می‌گوییم یک hit رخ داده و در غیر این صورت(در صورتی که این اطلاعات در حافظه نهان موجود نبود) می‌گوییم یک miss رخ داده است.

در صورتی که hit رخ دهد، بدون معطلی و در همان کلاک اطلاعات از حافظه نهان خوانده می‌شود و روند اجرایی برنامه ادامه می‌یابد.

و در صورتی که miss رخ دهد، باید اطلاعات را از حافظه اصلی به حافظه نهان انتقال دهیم. برای این بخش نیز دو حالت اتفاق می‌افتد. حالت اول این است که در خانه ای که آدرس مدنظر به آن map می‌شود، داده ای نداشته باشیم یا داده ای داشته باشیم که مقدار آن در cache با مقدار آن در حافظه تفاوتی نداشته باشد. در این حالت نیازی به نوشتن مقدار فعلی خانه مدنظر از cache در حافظه اصلی نیست و می‌توانیم اطلاعات را از حافظه اصلی بخوانیم و در حافظه نهان ذخیره کنیم. برای این کار از زمان بندی تعیین شده در جدول پایین استفاده می‌کنیم:

Cycles	1	2	3	4	5	6
cache_addr	0x00001000	0x00001000	0x00001000	0x00001000	0x00001000	0x00001000
mem_addr	0x00001000	x	x	x	x	x
cache_hit	0	0	0	0	0	1
mem_data_out	x	x	x	x	0x12345678	x
cache_data_out	x	x	x	x	x	0x12345678

حالت دوم به این صورت است که در خانه مدنظر، از قبل داده ای موجود باشد که مقدار آن با مقدار موجود در حافظه اصلی متفاوت است. برای این حالت ابتدا باید داده موجود در حافظه نهان را در حافظه اصلی بنویسیم و سپس داده جدید مدنظر را از حافظه اصلی خوانده و در حافظه نهان ذخیره کنیم. برای این حالت نیز از جدول دو جدول زیر استفاده می‌کنیم:

Cycles	1	2	3	4	5	6
cache_addr	0x00001000	0x00001000	0x00001000	0x00001000	0x00001000	0x00001000
mem_addr	0x00001200	0x00001200	0x00001200	0x00001200	0x00001200	0x00001000
cache_hit	0	0	0	0	0	0
mem_we	1	0	0	0	0	0
mem_data_in	0x87654321	x	x	x	x	x
cache_data_out	x	x	x	x	x	x

Cycles	1	2	3	4	5	6
cache_addr	0x00001000	0x00001000	0x00001000	0x00001000	0x00001000	0x00001000
mem_addr	0x00001000	x	x	x	x	x
cache_hit	0	0	0	0	0	1
mem_data_out	x	x	x	x	0x12345678	x
cache_data_out	x	x	x	x	x	0x12345678

نکته: کلاک ششم جدول بالا منطبق بر کلاک اول جدول پایین است.

همچنین حافظه نهان باید قابلیت reset نیز داشته باشد به این صورت که حافظه پس از reset شدن باید خالی شود. برای این کار هم محتوای حافظه و هم بیت‌های dirty و valid را صفر می‌کنیم.

در این فاز ماژول pc نسبت به فاز قبلی تغییراتی داشته است. به گونه ای که یک سیگنال ready به عنوان خروجی ماژول cache تعیین شده که این سیگنال ورودی pc است و هدف از آن این است که هنگامی که پردازش‌های در حال انجام در cache به اتمام رسیدند، pc به وسیله سیگنال ready از این مسئله آگاه شده و تا قبل از آن به سراغ پردازش دستور بعد نرود.

همچنین در این فاز یک ماژول cache نیز به طراحی اضافه شده که همان حافظه نهان است و در ادامه به بررسی آن می‌پردازیم:

همانطور که بالاتر ذکر شد، طول هر بلاک از حافظه نهان به اندازه یک ورد در نظر گرفته شده(زیرا دسترسی ما به حافظه تنها از طریق ورودی‌ها و خروجی‌های ماژول mips_core است که به صورت 32 بیتی است. بنابراین اگر بخواهیم طول بلاک را بیش از یک word بگیریم نیاز است چند بار به حافظه مراجعه کنیم که این کار بهینه نیست زیرا هربار مراجعه به حافظه باعث افزایش تاخیر می‌شود.) و سایز حافظه نهان به طور کل 8 کیلوبایت(2^13 بایت) است و می‌دانیم هر word 4 بایت اشغال می‌کند. بنابراین تعداد بلاک‌های حافظه نهان برابر با 2^11 می‌شود.

برای ذخیره مقادیر حافظه نهان نیز یک رجیستر دو بعدی به نام cache تعریف می‌کنیم و برای ذخیره tag ها نیز از یک رجیستر دو بعدی استفاده می‌کنیم.

همچنین دو آرایه برای ذخیره valid بودن یا نبودن و dirty بودن یا نبودن تعریف کرده ایم.

همانطور که می‌دانید، مقادیر موجود در هر یک از خانه‌های حافظه قبل از اینکه برای اولین بار چیزی در آن خانه نوشته شود، غیرمعتبر هستند. برای آگاهی از اینکه مقدار موجود در این خانه از cache توسط خودمان نوشته شده و متعبر است یا خیر، از آرایه valid استفاده می‌کنیم.

از طرفی به علت write back بودن حافظه نهان، ممکن است مقدار موجود در حافظه اصلی با مقدار موجود در cache متفاوت باشد و نیاز باشد که مقدار موجود در حافظه نهان به حافظه اصلی منتقل شود. به این منظور آرایه dirty را تعریف می‌کنیم و در صورتی که مقدار آن در خانه مدنظر برابر با 1 باشد، یعنی مقدار موجود در حافظه نهان با مقدار موجود در حافظه اصلی متفاوت است و پیش از نوشتن مقدار جدید در حافظه نهان، نیاز است مقدار موجود در حافظه اصلی با توجه به مقدار موجود در حافظه نهان update شود.

همچنین یک متغیر clock counter تعریف شده که با توجه به آن متوجه می‌شویم در کلاک چندم قرار داریم و با توجه به این مسئله3 تاخیرها را مدیریت می‌کنیم.

وقتی reset برابر با صفر بود، مقدار آرایه های valid_array، dirty_array و cache را برابر با صفر قرار می‌دهیم.

در صورتی که بخواهیم داده ای را بخوانیم، اگر hit رخ دهد، داده مدنظر را از حافظه نهان خوانده و مقدار data_out را مساوی با آن قرار می‌دهیم.

در صورتی که بخواهیم داده ای را بخوانیم ولی miss رخ دهد و بیت dirty برابر با 0 باشد، ابتدا داده را از حافظه اصلی به حافظه نهان منتقل می‌کنیم و سپس آن را از حافظه نهان به data_out انتقال می‌دهیم.

و در نهایت در صورتی که بخواهیم داده ای را بخوانیم ولی miss رخ دهد و بیت dirty برابر با 1 باشد، ابتدا مقدار موجود در حافظه نهان را در حافظه اصلی ذخیره می‌کنیم و به انتقال داده جدید از حافظه اصلی به حافظه نهان می‌پردازیم و در نهایت data_out را برابر با مقدار موجود در حافظه نهان قرار می‌دهیم

نکته: در کلیه مراحل بالا در صورتی که کل آن word را بخواهیم، همه بایت های آن را در data_out قرار می‌دهیم و در صورتی که فقط یک بایت از word را بخواهیم، تنها byte مدنظر را در بایت کم ارزش data_out قرار می‌دهیم و بقیه بایت‌های data_out را صفر می‌کنیم.

حال به بررسی نوشتن در حافظه نهان می‌پردازیم:

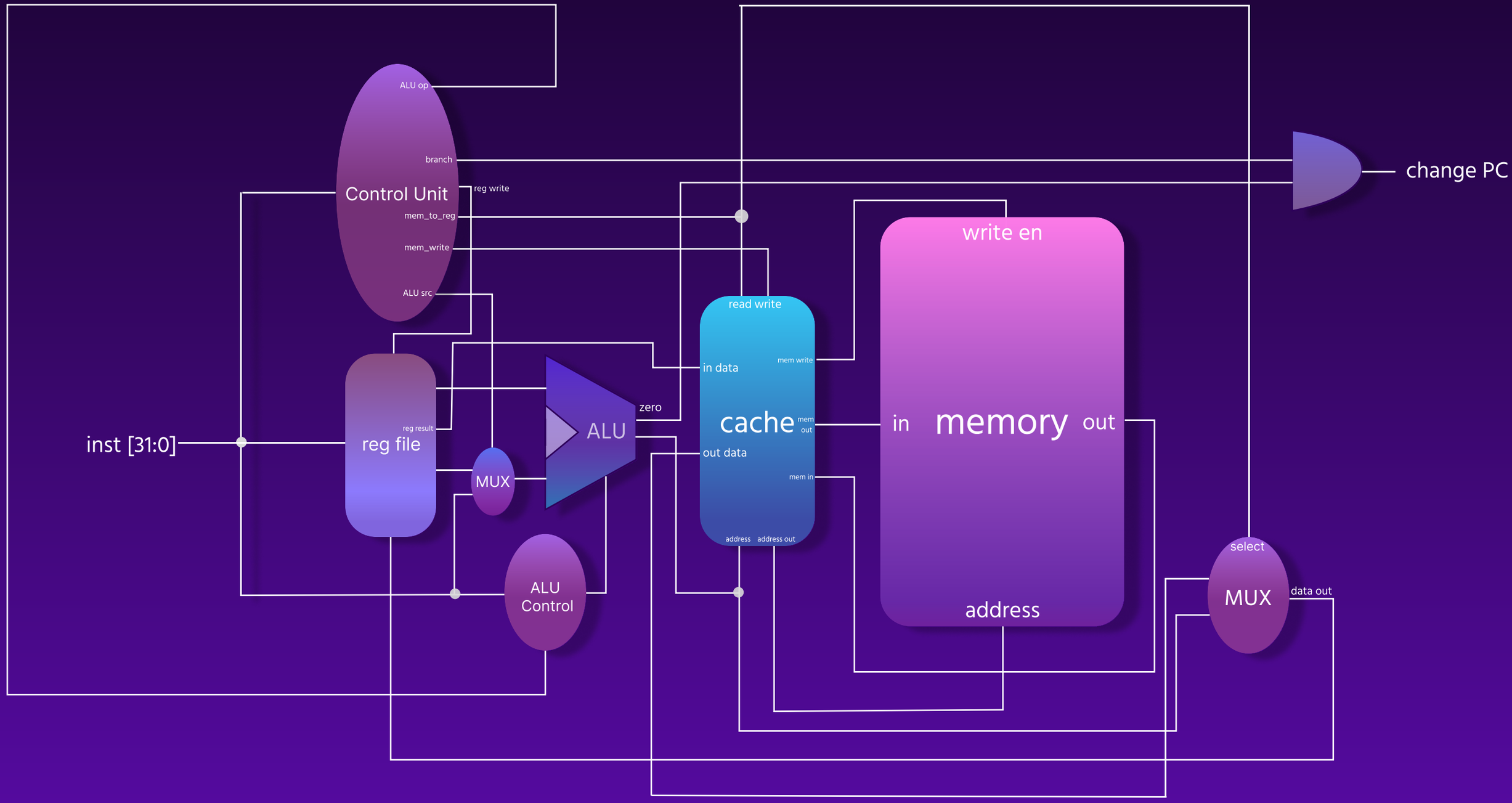
در صورتی که hit رخ دهد، مقدار مدنظر را data_in دریافت کرده و در حافظه نهان ذخیره می‌کنیم.

در صورتی که miss رخ دهد و بیت dirty برابر با 0 باشد، نیازی نیست مقدار فعلی موجود در حافظه نهان را به حافظه اصلی منتقل کنیم. پس اطلاعات مدنظر را از حافظه اصلی به حافظه نهان منتقل می‌کنیم و سپس data_in را در حافظه نهان ذخیره می‌کنیم.

در صورتی که miss رخ دهد و بیت dirty برابر با 1 باشد، یعنی مقدار موجود در حافظه نهان با مقدار موجود در حافظه اصلی متفاوت است، بنابراین ابتدا اطلاعات موجود در حافظه نهان را به حافظه اصلی منتقل می‌کنیم، سپس اطلاعات جدید مدنظر از حافظه اصلی را به حافظه نهان منتقل کرده و بعد data_in را در این خانه از حافظه نهان ذخیره می‌کنیم.

نکته: در تمام مراحل نوشتن اگر کل یک word را مدنظر داشته باشیم، کل آن را روی خانه مدنظر از حافظه نهان ذخیره می‌کنیم و در صورتی که فقط یک بایت از word مدنظر باشد، بایت مربوطه را با توجه به بایت کم‌ارزش data_in تغییر می‌دهیم و سایر بایت‌ها را بدون تغییر می‌گذاریم.

در ادامه خلاصه ای از شیوه پیاده سازی و ارتباط ماژول ها با هم را مشاهده می‌کنیم:


ALU op
branch
Control Unit
reg write
mem_to_reg
mem_write
ALU src
change PC
write en
read write
in data
mem write
zero
ALU
cache
mem out
in
memory
out
reg result
inst [31:0]
reg file
MUX
out data
mem in
address
address out
ALU Control
select
MUX
data out
address