# گزارش فاز سوم پروژه درس معماری کامپیوتر

پارمیدا جوادیان

پرنیان رضوی پور

دنیا نوابی

کیانا موسی زاده

دانشگاه صنعتی شریف

تابستان 1401

# فهرست

**مقدمه**

در دو فاز اول پروژه پردازنده میپس همراه به دسترسی به حافظه و در نظر گرفتن زمان مورد نیاز طراحی کردیم. انجام یک دستور العمل در این پردازنده شامل چند مرحله است که به صورت متوالی انجام می‌شوند و در هر مرحله دستور العمل مدنظر در یک بخش از مدار سخت افزاری قرار دارد.

برای استفاده بهینه از این پردازنده می‌توان ساختار آن را به صورتی تغییر داد که هر بخش در حال پردازش اطلاعات مربوط به خود باشد.

از طرفی می‌دانیم پردازش یک دستور باید به صورت متوالی صورت بگیرد بنابراین همه بخش های مدار نمی‌توانند به صورت موازی یک دستور را اجرا کنند؛ ولی می‌توان چند دستور را به طور همزمان پردازش کرد، به این معنی که یک بخش پردازش یک دستور را انجام دهد و سپس آن دستور به مرحله بعد برود و بخش فعلی به پردازش دستور بعدی بپزدازد. به این کار اجرا به صورت خط لوله گفته می‌شود.

در این فاز از پروژه، ما پردازنده خود را به یک خط لوله 5 مرحله ای تبدیل می‌کنیم.

این 5 مرحله شامل مراحل زیر هستند:

1) IF(Instruction Fetch)
   در این مرحله دستورالعمل را از حافظه مربوط به دستورات می‌خوانیم.
2) ID(Instruction Decode)
   در این مرحله دستورالعمل را کدگشایی کرده و ورودی‌های لازم را از رجیسترها می‌خوانیم.
3) EX(Execute)
   در این مرحله محاسبات انجام می‌شود.(این مرحله شامل محاسبه آدرس و سایر محاسبات منطقی می‌شود)
4) MEM(Memory Access)
   در این مرحله دسترسی به حافظه اتفاق می‌افتد و در صورت نیاز اطلاعات از حافظه خوانده یا در آن نوشته می‌شوند.
5) WB(Write Back)
   در این مرحله در صورت نیاز خروجی مدار در ثبات مورد نظر نوشته می‌شود.

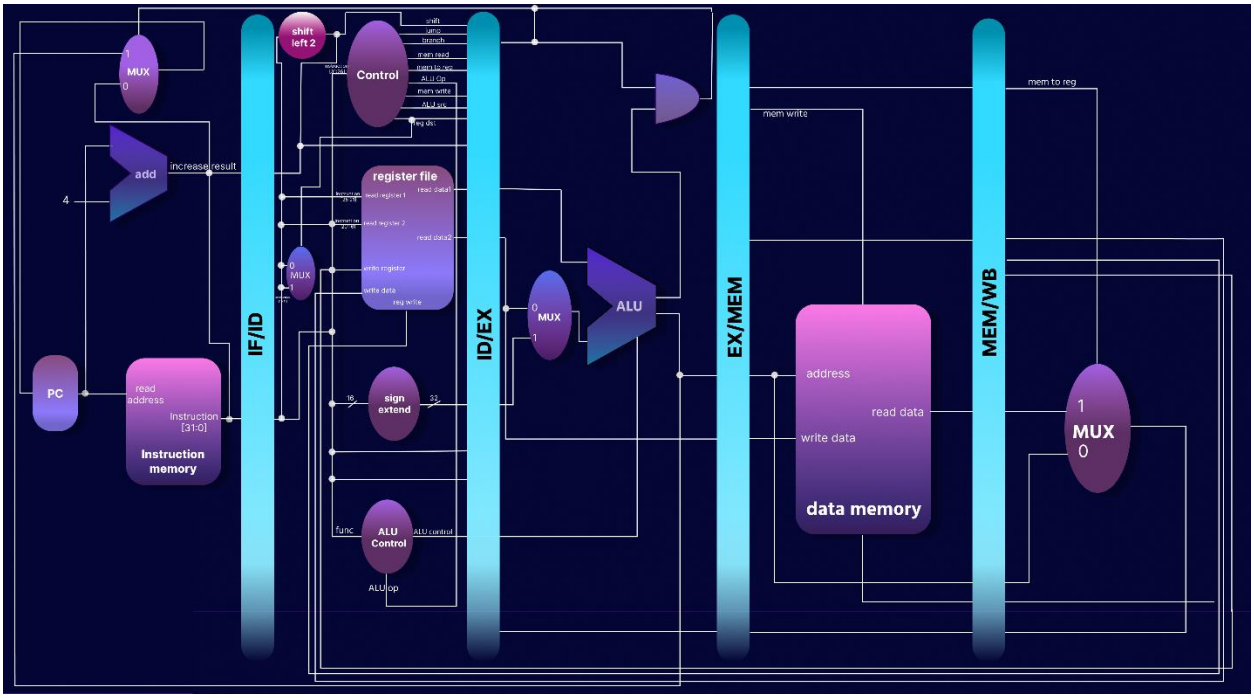
برای پیاده سازی خط لوله ما نیز پردازنده خود را به 5 stage با نام های IF_stage، ID_stage، EX_stage، MEM_stage و WB_stage تقسیم کردیم. در فازهای قبل همه عملیات در mips_core

انجام می‌شد و از سخت افزارهای مورد نیاز در این ماژول instance گرفته می‌شد. در این فاز عملیات‌ها به stage های نام برده انتقال یافته و در ماژول mips_core از این stage ها instance گرفته شده است.

همچنین چهار ثبات با نام‌های IF_ID_reg(ثبات برای ذخیره مقدار میانی دو IF stage و ID)، ID_EX_reg(ثبات برای ذخیره مقدار میانی دو ID stage و EX)، EX_MEM_reg(ثبات برای ذخیره مقدار میانی دو EX stage و MEM) و MEM_WB_reg(ثبات برای ذخیره مقدار میانی دو stage MEM و WB) تعریف شده است. هدف از تعریف این ثبات‌ها این است که خروجی یک stage را ذخیره کند تا برای stage بعد قابل استفاده باشد.

## نحوه پیاده سازی

در ادامه نقشه طراحی پردازنده خط لوله را مشاهده می‌کنید:



## ماژول‌های مورد استفاده در معماری خط لوله

حال به بررسی ماژول‌های اضافه شده به طور دقیق‌تر می‌پردازیم:

### 1) ماژول IF_stage

این ماژول شامل ورودی‌هایی اعم از کلاک، ریست، next_pc، branch و hold و خروجی های pc و pc_increase_result است. در این ماژول با instance گرفتن از ماژول‌های دیگر، دستور بعدی را از روی آدرس برحسب اینکه پرش(شامل دستورات branch و jump) رخ داده یا نه تعیین می‌کنیم.(انجام یا عدم انجام پرش‌ها در مرحله EX_stage مشخص می‌شوند وآدرس پرش و برقراری یا عدم برقراری شرط پرش به این مرحله منتقل می‌شود.)

### 2) ماژول IF_ID_reg

این ماژول buffer ارتباطی دو ماژول IF_stage و ID_stage است و اطلاعاتی مثل دستور فعلی و آدرس دستور بعدی را از مرحله instruction fetch به مرحله instruction decode منتقل می‌کند. شیوه کار نیز به این صورت است که ماژول در انتظار لبه بالارونده سیگنال کلاک می‌ماند و پس از فرارسیدن آن اگر سیگنال hold حافظه نهان فعال نباشد، خروجی‌های IF_stage را به ورودی‌های ID_stage متصل می‌کند.

### 3) ماژول ID_stage

ورودی‌های این ماژول شامل کلاک، ریست، دستور تحت پردازش، regWrite، pc_increase_result، rd و rd_data است.

تمام ورودی‌های این ماژول بجز سه ورودی rd، rd_data و regWrite از مرحله instruction fetch به این مرحله منتقل می‌شوند و این سه ورودی از مرحله write back دریافت می‌شوند زیرا برای محاسبه اطلاعاتی که باید در ثبات هدف نوشته شود، نیاز است این اطلاعات یا در مرحله execute توسط alu محاسبه شوند و یا در مرحله memory access از حافظه خوانده شوند، بنابراین وقتی دستور در مرحله instruction decode قرار دارد این اطلاعات هنوز آماده نیستند و نیاز است آنها را پس از مشخص شدن نتیجه از مرحله write back دریافت کنیم.

خروجی های این ماژول شامل aluctrl، برخی از سیگنال‌های کنترلی تولید شده توسط control unit مثل alusrc, shift, jump, branch, memtoreg, MemWrite و is_byte، شماره رجیسترها مانند destReg، src1_out و src2_out و ... است.

در این ماژول عملیاتی اعم از sign_extend کردن و خواندن اطلاعات از رجیسترها صورت می‌گیرد.

### 4) ماژول ID_EX_reg

این ماژول buffer ارتباطی دو ماژول ID_stage و EX_stage است. وظیفه این ماژول مواردی شامل انتقال اطلاعات درون ثبات‌ها و مقدار sign_extend شده ورودی immediate از مرحله instruction decode به مرحله execute است.

شیوه کار این ماژول اینگونه است که با رسیدن لبه بالارونده کلاک، در صورتی که در وضعیت hold حافظه نهان نباشیم، ورودی‌های دریافت شده از ID_stage را به خروجی‌های متصل به EX_stage منتقل می‌کند.

### 5) ماژول EX_stage

در این ماژول نیز با instance گرفتن از سایر ماژول‌ها مواردی اعم از خروجی alu و آدرس خروجی پرش‌ها محاسبه می‌شوند.

برخی سیگنال‌های کنترلی که مربوط به خواندن از حافظه، نوشتن در حافظه یا نوشتن روی ثبات‌ها هستند در این مرحله مورد نیاز نیستند ولی چون در مراحل بعدی به آنها نیاز داریم، آنها را از به عنوان ورودی‌های این ماژول دریافت می‌کنیم تا بتوانیم آنها را stage های بعدی انتقال دهیم.

### 6) ماژول EX_MEM_reg

این ماژول buffer ارتباطی دو ماژول EX_stage و MEM_stage است و مانند سایر ماژول های ثبات، با فرارسیدن لبه بالارونده کلاک در صورت فعال نبودن سیگنال hold حافظه نهان، ورودی‌های دریافت شده از ماژول EX_stage را به خروجی‌های MEM_stage منتقل می‌کنند.

### 7) ماژول MEM_stage

این ماژول با دریافت سیگنال‌های کنترلی حافظه، وظیفه ارتباط با حافظه نهان را برعهده دارد تا حافظه نهان عملیات‌های درراستای ارتباط با حافظه اصلی را انجام دهد.

### 8) ماژول MEM_WB_reg

این ماژول buffer ارتباطی دو ماژول MEM_stage و WB_stage است. این ماژول نیز در صورت قرار داشتن در لبه بالارونده کلاک، در صورتی که سیگنال hold حافظه نهان فعال نباشد، خروجی‌های ماژول MEM_stage را به ورودی‌های ماژول WB_stage متصل می‌کند.

### 9) ماژول WB_stage

این ماژول ورودی‌های مورد نیاز را به ID_stage می‌دهد تا در صورت نیاز اطلاعات روی رجیستر هدف نوشته شود و به این صورت اجرای دستور فعلی پایان می‌یابد.