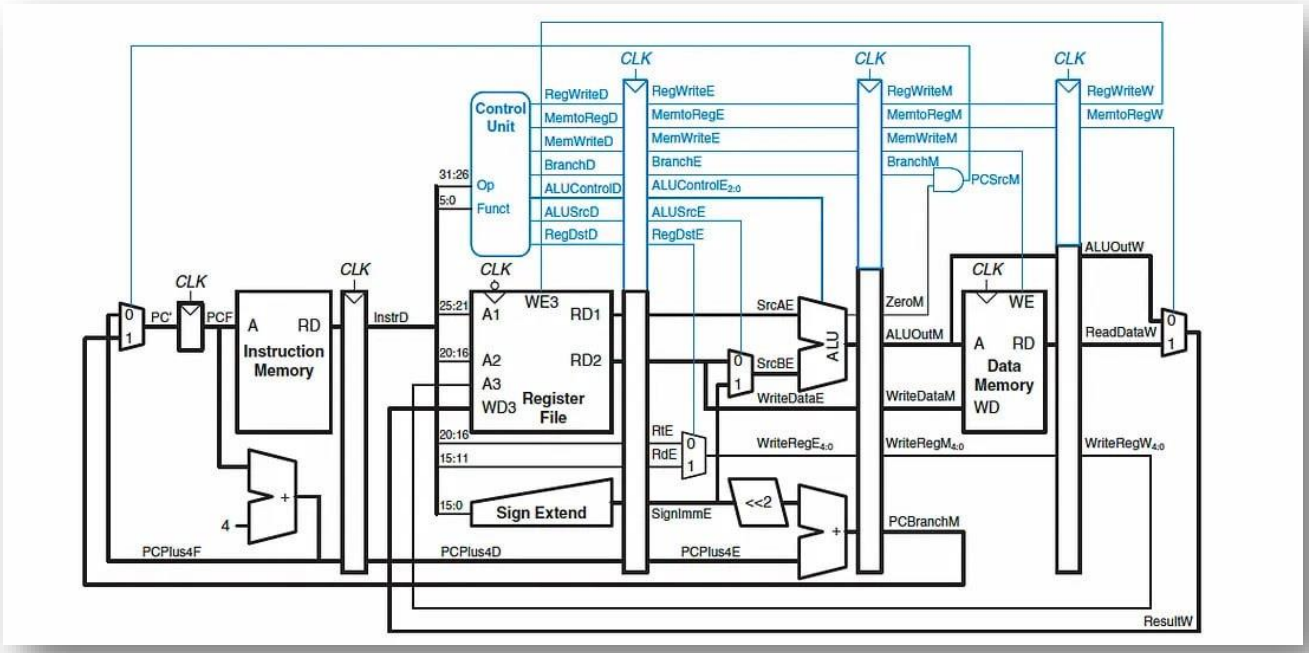
تمرین هفتم - عملی (مسیر داده و تست جمع برداری)

مسیر داده (PipeLine MIPS DataPath):

برای پیاده‌سازی این بخش از datapath زیر استفاده کردیم و با اعمال تغییراتی امکان اجرای دستورات ذکرشده در صورت سوال را فراهم کردیم:



این datapath به طور کلی در کلاس بررسی شده و بسیار مشابه همان datapath پردازنده‌ی single-cycle است؛ البته تفاوت‌هایی دارد که در ادامه به آن‌ها خواهیم پرداخت. در ادامه، به بررسی تفاوت‌های این پردازنده با پردازنده‌ی single-cycle و همچنین تغییراتی که باید در datapath موجود در تصویر اعمال کنیم تا همه‌ی دستورات مورد نظر پوشش داده‌شوند، می‌پردازیم:

- **تفاوت PipeLine با SingleCycle**

تفاوت اصلی این پردازنده با single-cycle در این است که به دلیل اجرای مرحله به مرحله‌ی دستورات و این که هر مرحله باید در یک cycle انجام شود، باید بین مراحل رجیسترهایی را در نظر بگیریم تا داده‌هایی که لازم است به مرحله‌ی بعد منتقل شوند را در خود ذخیره کند و در لبه‌ی بالارونده‌ی کلاک به مرحله‌ی بعدی منتقل کند.

در پردازنده‌ای که پیاده‌سازی کرده‌ایم، دستورات در ۵ مرحله‌ی Instruction Fetch، Instruction Decode/Register Read، Execution، Memory Access و Writeback اجرا می‌شوند و هر مرحله یک cycle طول می‌کشد. تنها استثنای این قاعده دستور mult است؛ در این دستور مرحله‌ی Execution بیش از یک cycle طول می‌کشد و پس از روشن شدن سیگنال ready به مرحله‌ی بعد می‌رویم. تا زمانی‌که جواب ضرب آماده نشده‌است، دستور بعدی وارد pipeline نمی‌شود و این محدودیت توسط Hazard Unit کنترل می‌شود.

برای پیاده‌سازی این مجموعه رجیسترها، از چند رجیستر در هر بلوک استفاده کرده‌ایم و در مدار قرار داده‌ایم:

- **تغییرات تصویر DataPath**

در ابتدا لازم به ذکر است که از سمت چپ مجموعه‌ی رجیسترها به ترتیب PC، Registers1، Registers2، Registers3 و Registers4 نام‌گذاری شده‌اند.

۱. اولین تفاوت در این است که mux پیش از PC را ۴ به ۱ گذاشته‌ایم تا امکان اجرای دستورات j و jr فراهم شود. ورودی‌های ۲ و ۳ این mux به ترتیب مربوط به آدرس jr و j هستند که به ترتیب از رجیستر R7 و بخش immediate دستور خوانده می‌شوند.

۲. یک mux پیش از Registers1 قرار می‌گیرد تا در صورتی که Hazard Unit بخواهد، به جای دستور فعلی، noop یا همان no-operation را در Registers1 ذخیره کند. ورودی شماره 0 این mux همان دستور فعلی و ورودی شماره 1 معادل دستور noop است. سیگنال مورد نیاز برای این mux توسط Hazard Unit تولید می‌شود.

۳. در این تصویر به سیگنال MemRead اشاره‌ای نشده‌است؛ ولی آن را اضافه کرده‌ایم و پس از عبور آن از Registers2 و Registers3 به Data Memory ورودی داده می‌شود.

۴. سیگنال Branch خروجی از control unit را حذف و به جای آن سیگنال دو بیتی PCPath را در نظر گرفته‌ایم. این سیگنال پس از عبور از Registers2 و Registers3 آپدیت می‌شود و سپس خروجی آن یعنی FinalPCPath از Registers4 عبور می‌کند و به عنوان سیگنال کنترلی mux پیش از PC وارد آن می‌شود. آپدیت شدن آن نیز به این صورت است که اگر دستور beq یا bnq باشد ولی سیگنال zero روشن نشده‌باشد، مقدار FinalPCPath برابر همان 00 می‌شود تا پس از اتمام این دستور، دستور بعدی اجرا شود.

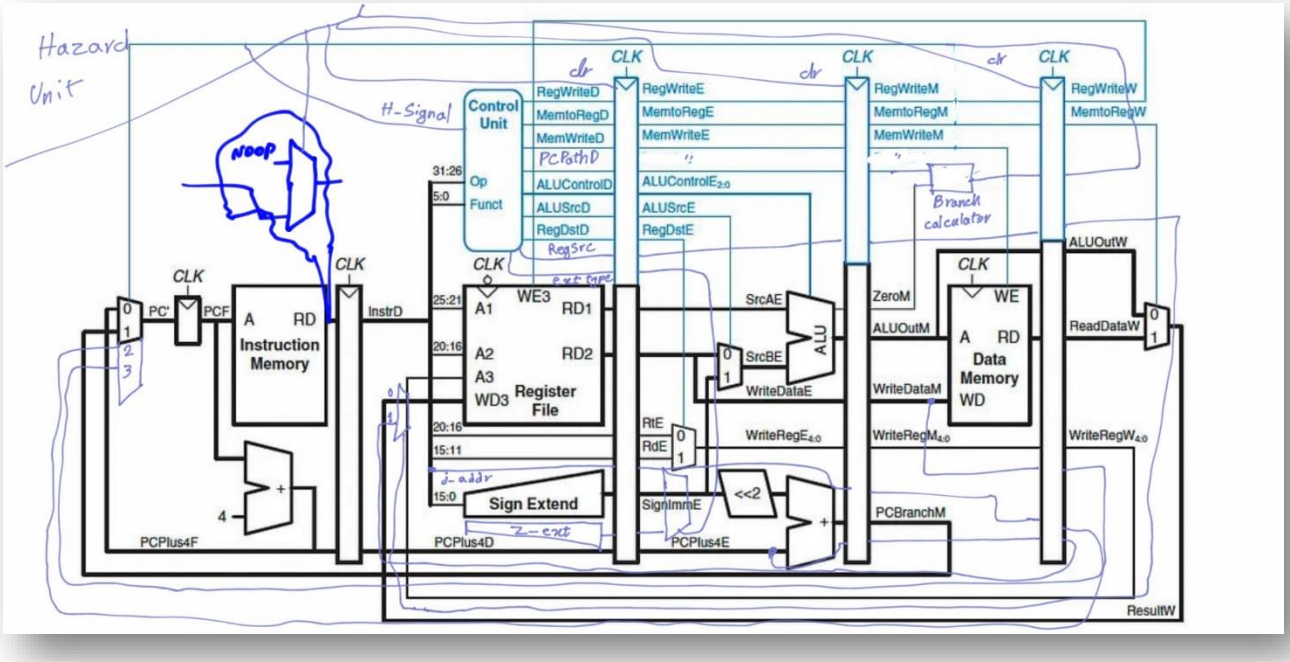
۵. یک mux پیش از ورودی WriteData در بلوک Register File قرار دادیم که ورودی شماره 0 آن معادل همین حالت فعلی است و ورودی شماره 1 آن معادل PC + 2 است، تا بتوانیم دستور jal را پیاده‌سازی کنیم. سیگنال مورد نیاز این mux نیز با عنوان RegSrc از control unit خروجی گرفته می‌شود. این سیگنال پس از عبور از Registers2، Registers3 و Registers4 اعمال می‌شود.

۶. در کنار بلوک SignExtend یک بلوک ZeroExtend نیز قرار دادیم، زیرا در دستورات ANDi و Ori نیاز داریم. سپس یک mux پس از این دو بلوک قرار دارد تا یکی از آن‌ها را خروجی دهد. ورودی شماره 0 این mux متعلق به SignExtend و ورودی شماره 1 مربوط به ZeroExtend است. سیگنال مورد نیاز این mux نیز با عنوان ExtType از control unit خارج می‌شود. این سیگنال پس از عبور از Registers2 اعمال می‌شود.

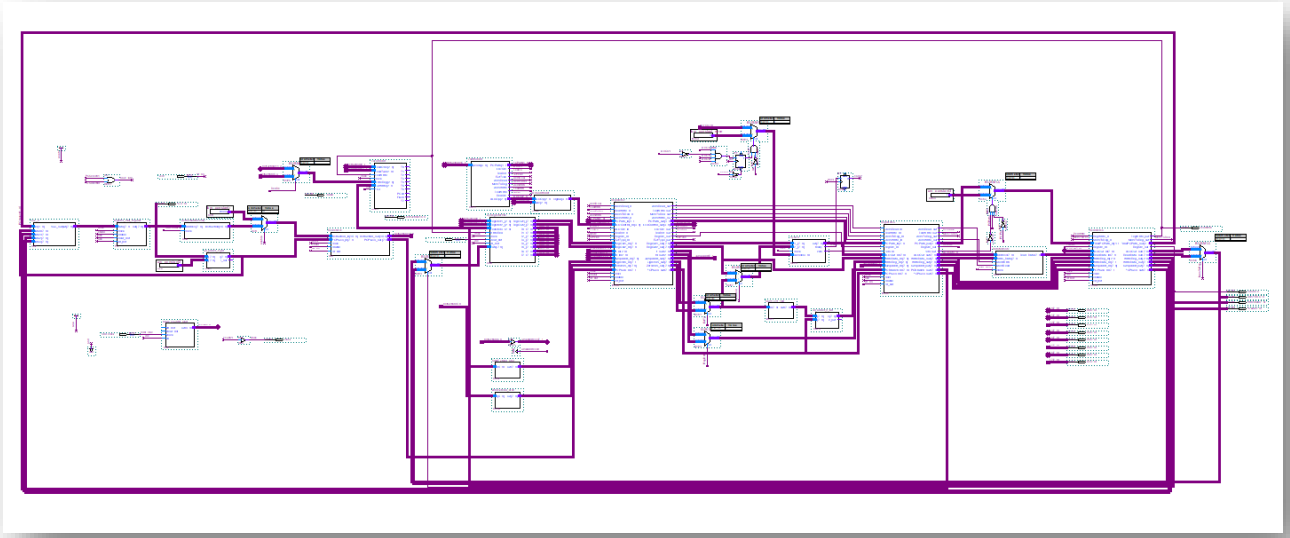
۷. در مرحله‌ی Instruction Decode بیت‌های مربوط به immediate نیز از Registers2، Registers3 و Registers4 عبور می‌کنند و به عنوان ورودی 2 در mux پیش از PC قرار می‌گیرند. این بیت‌ها در واقع همان JumpAddress را مشخص می‌کنند و پیش از انتقال ۱ بیت به سمت چپ شیفت می‌خورند تا به جای ۷ بیت به ۸ بیت تبدیل شوند.

۸. تفاوت دیگر نیز حضور Hazard Unit است تا بتواند، وابستگی‌ها را کنترل کند و در صورت مواجهه با وابستگی PipeLine را متوقف کند تا مقدار رجیستر مورد نظر آماده شود؛ سپس به اجرای ادامه‌ی دستورات بپردازد. این بلوک سیگنال‌های noop، H-signal، enable و clear_not را تولید می‌کند. سیگنال H-signal به control unit ورودی داده می‌شود. سیگنال‌های enable و clear_not نیز برای کنترل آن ۵ رجیستر هستند.

تصویری از تغییرات اعمال‌شده به طور کلی به صورت زیر است:



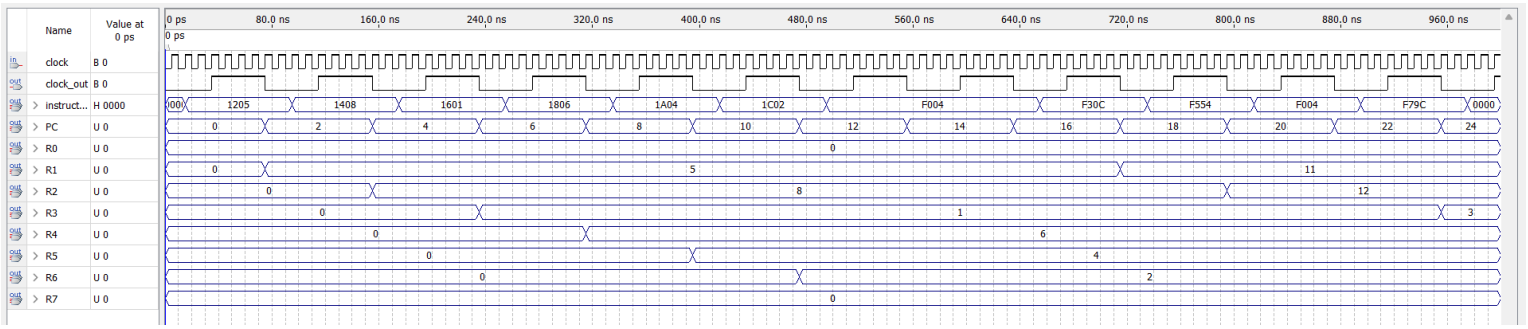
تصویر نهایی DataPath نیز به این صورت است (این تصویر از نمای دور است و فایل آن شامل تمامی جزئیات ضمیمه شده‌است):



تست پردازنده:

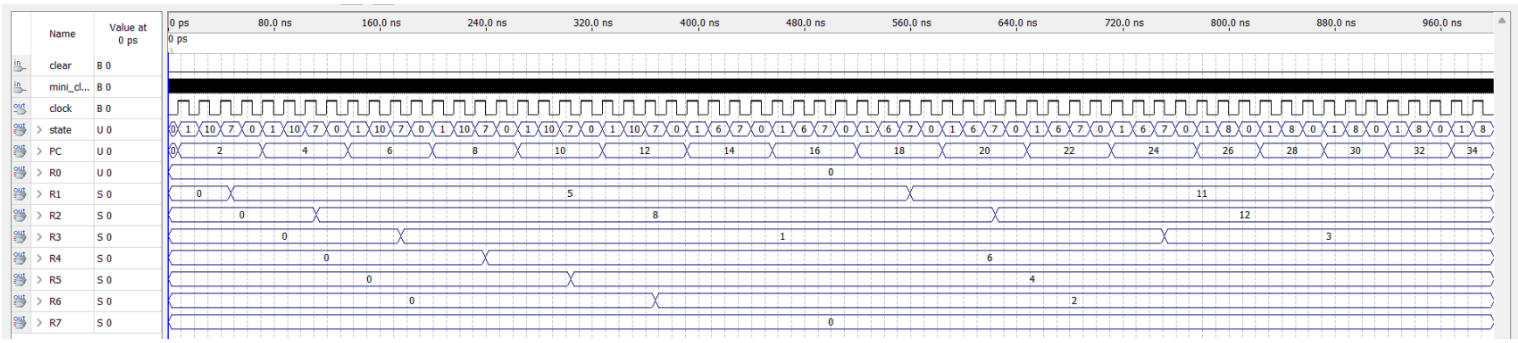
برای تست این پردازنده و مقایسه‌ی آن با single-cycle و multi-cycle یک برنامه‌ی vector addition نوشته‌ایم. این برنامه دو بردار ۳-تایی را با یکدیگر جمع می‌کند. درایه‌های بردار اول در رجیسترهای R1 و R2 و R3 قرار دارند و به ترتیب ۵ و ۸ و ۱ مقداردهی شده‌اند. درایه‌های بردار دوم در رجیسترهای R4 و R5 و R6 قرار دارند و به ترتیب ۶ و ۴ و ۲ مقداردهی شده‌اند. خروجی نیز آرایه‌ای است که درایه‌های آن رجیسترهای R1 و R2 و R3 هستند.

ابتدا waveform این برنامه را در پردازنده‌ی single-cycle مشاهده می‌کنید:



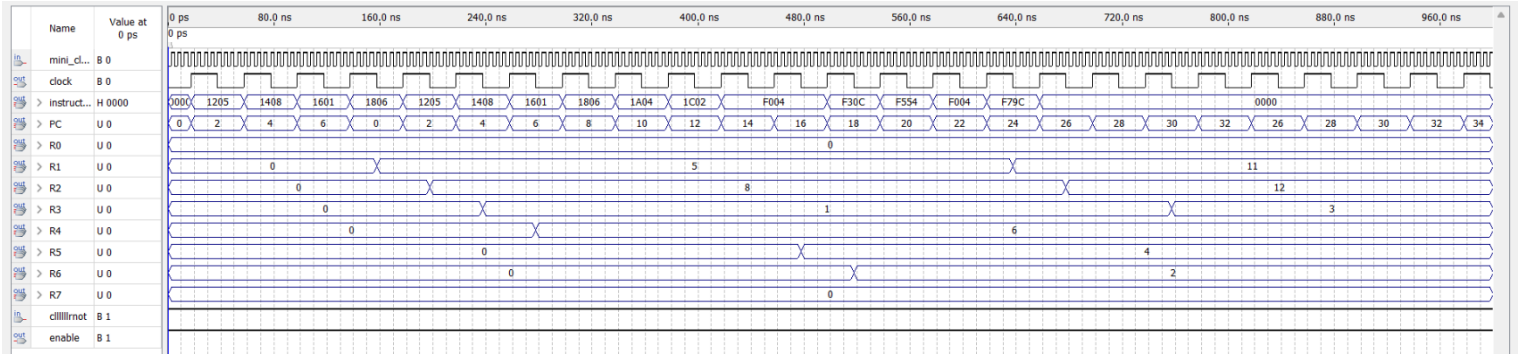
همان‌طور که مشخص است برنامه در ۱۲ cycle اجرا شده‌است.

حالا waveform همین برنامه را در پردازنده‌ی multi-cycle مشاهده می‌کنید:



همان‌طور که مشخص است برنامه در ۴۸ cycle اجرا شده‌است.

در نهایت waveform برنامه را در پردازنده‌ی pipeline مشاهده می‌کنید:



همان‌طور که مشخص است برنامه در ۱۹ cycle انجام شده‌است.

نکته‌ی قابل توجه این است که تناوب کلاک در پردازنده‌ی single-cycle برابر 10ns، در پردازنده‌ی multi-cycle برابر 1ns و در پردازنده‌ی pipeline برابر 5ns است. با این اوصاف این سه پردازنده به ترتیب برنامه‌ی مشابه را در 120ns، 48ns و 75ns اجرا کرده‌اند.

به نظر می‌آید با توجه به استفاده از no-op، در این برنامه multi-cycle بهتر از pipeline عمل کرده‌است، ولی در pipeline به خوبی تعداد cycle ها در مقایسه با multi-cycle کاهش یافته‌است.