گزارش فاز اول پروژه درس معماری کامپیوتر

پارمیدا جوادیان

پرنیان رضوی پور

دنیا نوابی

کیانا موسی زاده

دانشگاه صنعتی شریف

بهار 1401

هدف از این پروژه طراحی پردازنده میپس با استفاده از زبان وریلاگ یا سیستم وریلاگ است.

برای این کار کد صفر و یک داده شده را با توجه به نوع دستور به بخش های مختلف تقسیم می‌کنیم و برای هر دستور متناسب با خواسته آن عملیات متفاوتی را انجام می‌دهیم.

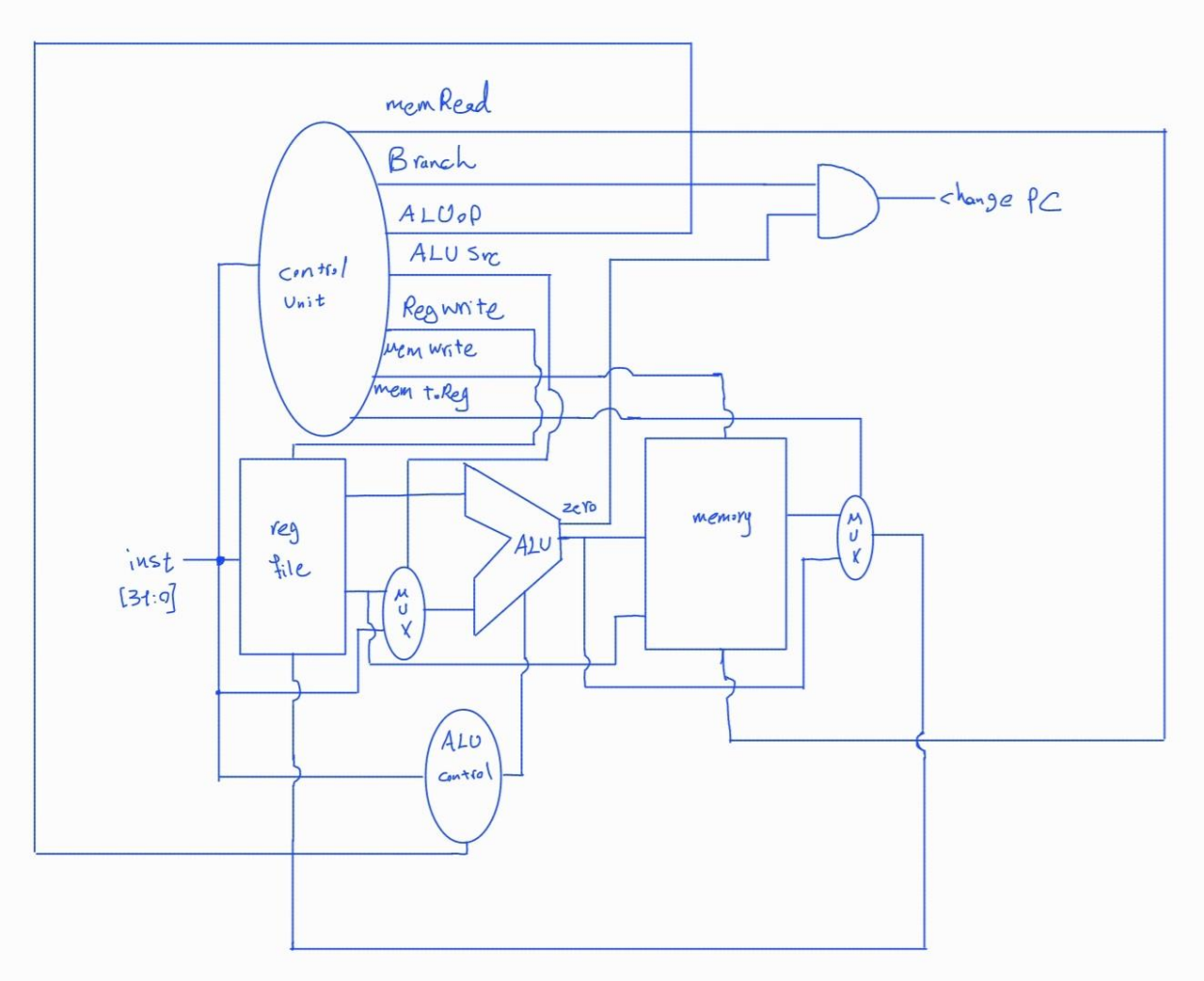
همانطور که می‌دانید دستورات میپس 32 بیت طول دارند و از سه فرمت R-type و J-type ، I-type پیروی می‌کنند.

در فرمت R-type، 6 بیت به opcode، 5 بیت به هر یک از rs و rt (که رجیستر های مبدا هستند)، 5 بیت به rd (که رجیستر مقصد است)، 5 بیت به مقدار شیفت (Sh.Amount) و 6 بیت به Func اختصاص می‌یابد. این فرمت شامل دستوراتی شامل or و and و xor و add و sub و sll و srl و slt و jr و... است. در این نوع دستور مقدار opcode همواره صفر است و با توجه به مقدار Func، دستور مشخص شده و با توجه به سایر ورودی ها (rs,rt,Sh.Amount)، مقدار خروجی محاسبه شده و در rd ذخیره می‌شود.

در فرمت I-type نیز 6 بیت به opcode، 5 بیت به rs (که رجیستر ورودی است)، 5 بیت به rd (که رجیستر خروجی است) و 16 بیت به مقدار Immediate (که یک عدد صحیح است) اختصاص می‌یابد. این نوع دستور شامل دستوراتی مثل ADDi و ANDi و XORi و BEQ و BGTZ و LW و SW و SLTi و... است.

در فرمت J-type نیز 6 بیت به opcode و 26 بیت به address تعلق می‌یابد و شامل دستوراتی اعم از J و JAL می‌شود.

در ادامه دورنمایی از طراحی انجام شده را مشاهده می‌کنید:



پردازش با بررسی inst شروع می‌شود؛ منظور از inst ، یک instruction (دستور) 32 بیتی از صفر و یک ها است.

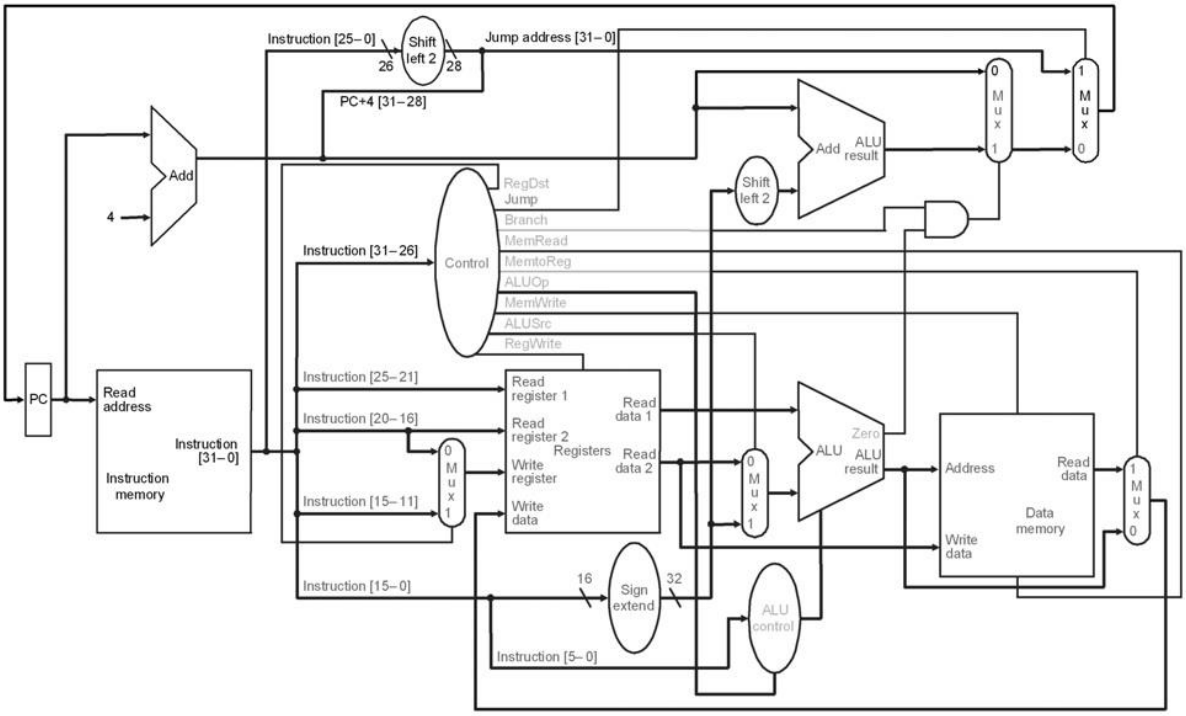
همانطور که در شکل مشخص است در ابتدا این دستور به regFile داده می‌شود و این ماژول با توجه به بیت‌هایی که مربوط به رجیستر های ورودی و خروجی هستند، داده های لازم را به عنوان خروجی به سایر بخش‌های پردازنده می‌دهد.

در مرحله بعد به واحد های ALU می‌رسیم، این بخش از دو ماژول ALU و ALU Control تشکیل شده است. وظیفه ALU Control این است که با توجه به instruction ، عملیات را مشخص کند تا این عملیات در ALU با توجه به ورودی هایی که از regFile دریافت شده است، انجام شود.

حال به توضیح واحد Control Unit می‌پردازیم، این واحد نیز با دریافت instruction و بررسی بیت‌های آن، تعدادی flag خروجی به ما می‌دهد تا با استفاده از آنها ادامه پردازش را انجام دهیم.

در ادامه در صورتی که طبق خروجی های بخش های قبل شامل Control Unit لازم باشد چیزی در حافظه بنویسیم یا چیزی از حافظه بخوانیم، به واحد memory می‌رویم و داده مد نظر(که مقدار آن را از regFile دریافت کرده ایم) را در محل مد نظر(که آن را از ALU دریافت کرده ایم) می‌نویسیم یا از آن می‌خوانیم.

همچنین برای مشاهده جزئیات بیشتر می‌توان شکل زیر را بررسی کرد:



حال به بررسی ماژول های طراحی شده با استفاده از زبان وریلاگ می‌پردازیم.

اولین ماژول، ماژول adder است که در آن دو ورودی 32 بیتی in1 و in2 و یک خروجی 32 بیتی out داریم که در این ماژول مقدار out برابر با حاصل جمع in1 و in2 محاسبه شده و به عنوان خروجی برگردانده می‌شود.

در ماژول multiplier، دو عدد 32 بیتی a و b به عنوان ورودی و یک عدد 32 بیتی به عنوان خروجی در نظر گرفته می‌شود. در این ماژول حاصل ضرب a و b محاسبه شده و 32 بیت کم ارزش آن به عنوان خروجی داده می‌شود. همچنین در این ماژول یک 64 net بیتی برای ذخیره مقدار کامل حاصل ضرب در نظر گرفته می‌شود تا در ادامه 32 بیت کم ارزش آن به عنوان خروجی داده شود.

به همین ترتیب در ماژول divider نیز دو عدد 32 بیتی a و b به عنوان ورودی و یک عدد 32 بیتی quotient به عنوان خروجی در نظر گرفته می‌شود. در این ماژول حاصل تقسیم a و b محاسبه شده و حاصل تقسیم به عنوان quotient توسط خروجی این ماژول به سایر واحدها داده می‌شود.

در ماژول alu، دو ورودی 32 بیتی reg1_data و reg2_data، یک ورودی 5 بیتی alu_control، یک خروجی 32 بیتی result و یک خروجی تک بیتی z_flag داریم. این ماژول با توجه به مقدار alu_control یک عملیات حسابی یا منطقی روی ورودی های reg1_date و reg2_data انجام می‌دهد به این صورت که reg1_data به عنوان operand اول و reg2_data به عنوان operand دوم مدنظر قرار می‌گیرد. شیوه محاسبه z_flag به این صورت است که در صورتی که مقدار result برابر با صفر باشد، مقدار z_flag برابر با 1 قرار می‌گیرد و در غیر این صورت مقدارش برابر با صفر در نظر گرفته می‌شود (این بیت برای نشان دادن اینکه خروجی صفر است یا ناصفر به کار می‌رود). در این ماژول همچنین از دو ماژول divider و multiplier، instance گرفته شده و تعدادی net نیز درون این ماژول تعریف شده است شامل دو 32 net بیتی mul_result و div_result و یک net تک بیتی carry.

در ماژول alu_control یک ورودی 6 بیتی func ، یک ورودی 4 بیتی alu_opcode و یک خروجی 5 بیتی alu_control داریم که در صورتی که دستور از فرم R-type تبعیت کند، با توجه به func (دستورات R-type همگی یک alu_opcode یکسان دارند.) و در غیر این صورت با توجه به alu_opcode دستور مد نظر را مشخص می‌کنیم و alu_control متناسب با آن را به عنوان خروجی ماژول در اختیار سایر واحد ها قرار می‌دهیم.

در ماژول controlUnit دو ورودی 6 بیتی func و opcode، یک خروجی 4 بیتی ALUop و نه خروجی یک بیتی dstReg، jmp، branch، MemWrite، MemtoReg، ALUSrc، RegWrite، halted و shift داریم که مقدار خروجی ها را با توجه به دستور مربوط به opcode ورودی مشخص می‌کنیم.

در ماژول mux دو ورودی 32 بیتی in1 و in2، یک ورودی تک بیتی s و یک خروجی 32 بیتی out داریم و با توجه به مقدار s، یکی از مقادیر in1 و in2 به عنوان خروجی به سایر بخش‌های داده می‌شوند.

در ماژول pc یک ورودی تک بیتی clk، یک ورودی 32 بیتی in و یک خروجی 32 بیتی out داریم. این ماژول در لبه های بالارونده کلاک مقدار out را برابر با مقدار in قرار می‌دهد.

در ماژول signExtend دو ورودی 16 بیتی input_data و i_sign و یک خروجی 32 بیتی out داریم. در این ماژول عمل signExtend روی input_data انجام می‌شود تا این ورودی به یک خروجی 32 بیتی تبدیل شود. در صورتی که i_sign برابر با صفر باشد، این عملیات بدون در نظر گرفتن علامت انجام شده و در غیر این صورت، علامت حفظ می‌شود.

در ماژول mips_core یک ورودی 32 بیتی inst، یک ورودی 8 بیتی mem_data_out، دو ورودی یک بیتی clk و rst_b، دو خروجی 32 بیتی inst_addr و mem_addr، یک خروجی 8 بیتی mem_data_in و دو خروجی تک بیتی halted و mem_write_en داریم. این ماژول شامل تعداد زیادی net و instance است و با توجه به flag های خروجی controlUnit، واحدهای مختلف را طبق datapath به هم متصل می‌کند.