معماری کامپیوتر - دکتر عطارزاده

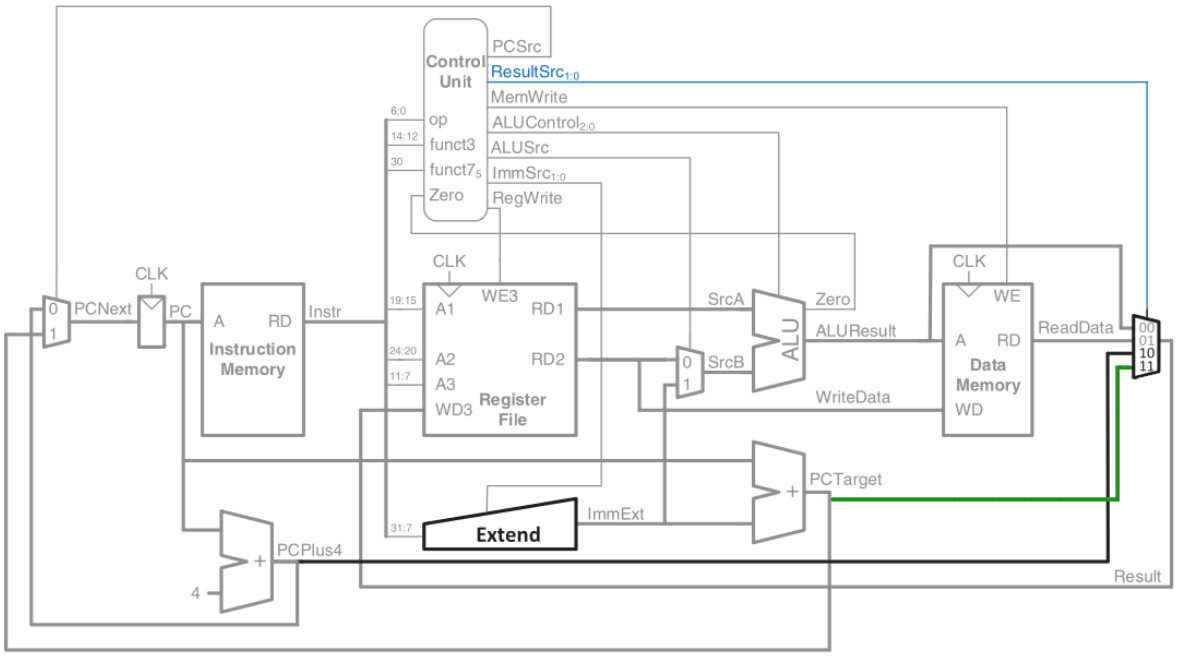
تمرین سری ۵ - بهار ۱۴۰۱

سوال ۱ - الف)

نیازی به اضافه کردن بلوک جدیدی نیست. زیرا برای اجرای دستور، تنها نیاز داریم تا مقدار immediate موجود در دستور را با مقدار فعلی‌اش جمع کنیم و در رجیستری در register file که در دستور مشخص شده (یا به طور دقیق‌تر rd) ذخیره کنیم. طراحی فعلی همه بلوک‌های موردنیاز را برای انجام این عملیات دارد. (تنها نیاز به تعدادی سیگنال جدید داریم که در قسمت‌های بعدی پاسخ بررسی می‌شود).

سوال ۱ - ب)

دو تغییر مورد نیاز است: ابتدا باید سیگنال `PCTarget` که حاصل جمع immediate و pc است را به مالتی‌پلکسر `Result` که به register file می‌رود وصل کنیم؛ و در نتیجه این مالتی‌پلکسر ۴ مقدار می‌پذیرد. این اتصال در شکل زیر با رنگ سبز نشان داده شده است.



همچنین باید واحد Extend نیز تغییر یابد و حالت جدیدی به آن اضافه شود. جدول زیر، جدول درستی واحد Extend را نشان می‌دهد که حالت جدید در سطر آخر آن اضافه شده است.

ImmSrc	ImmExt
000	{{20{Instr[31]}}, Instr[31:20]}
001	{{20{Instr[31]}}, Instr[31:25], Instr[11:7]}
010	{{20{Instr[31]}}, Instr[7], Instr[30:25], Instr[11:8], 1'b0}
011	{{12{Instr[31]}}, Instr[19:12], Instr[20], Instr[30:21], 1'b0}
100	**{Instr[31:12], 12'b0}**

سوال ۱ - ج)

سیگنال کنترلی جدیدی نیاز نیست. تنها تغییر لازم، ۳ بیتی شدن سیگنال ImmSrc است که پیش از این ۲ بیتی بود.

سوال ۱ - د)

جدول درستی Main decoder به صورت زیر در می‌آید.

Instruction	Opcode	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp	Jump
lw	0000011	1	000	1	0	01	0	00	0
sw	0100011	0	001	1	1	xx	0	00	0
R-type	0110011	1	xxx	0	0	00	0	10	0
beq	1100011	0	010	0	0	xx	1	01	0
I-type ALU	0010011	1	000	1	0	00	0	10	0
jal	1101111	1	011	x	0	10	0	xx	1
auipc	**0010111**	**1**	**100**	**x**	**0**	**11**	**0**	**xx**	**0**

جدول درستی ALU Decoder تغییری نمی‌کند. در واقع در هنگام اجرای این دستور از ALU موجود در مدار اصلا استفاده نمی‌شود.

سوال ۳ - الف)

اگر سیگنال مشکل دار RegWrite باشد، در این صورت محتویات register file با اجرای هر دستوری که برای آن‌ها سیگنال RegWrite برابر 0 است، به طور نامشخصی تغییر می‌کند؛ یعنی رجیستری نامشخص مقداری نامشخص می‌گیرد. (در اجرای دستوراتی که برای آن‌ها این سیگنال مقدار 1 دارد، تغییر نامشخصی صورت نمی‌گیرد). در نتیجه اگر از دستورات با RegWrite برابر 0 استفاده کنیم، **عملکرد دستورات بعدی که مقداری از register file می‌خوانند (یعنی تقریبا همه دستورات!) تحت تاثیر قرار می‌گیرد**؛ چون رجیستری با مقداری (احتمالا) تصادفی را می‌خوانند و با این مقدار تصادفی عملیات انجام می‌دهند. اما اگر برنامه ما فقط از دستورات با RegWrite با مقدار 1 استفاده کند، به مشکل نخواهد خورد. چون تغییر نامشخصی در register file رخ نخواهد داد.

سوال ۳ - ب)

اگر سیگنال مشکل دار MemWrite باشد، در این صورت محتویات data memory با اجرای هر دستوری که برای آن‌ها سیگنال MemWrite برابر 0 است، به طور نامشخصی تغییر می‌کند؛ یعنی خانه‌ی نامشخصی از data memory، مقداری نامشخص می‌گیرد (در اجرای دستوراتی که برای آن‌ها این سیگنال مقدار 1 دارد، تغییر نامشخصی صورت نمی‌گیرد). در نتیجه اگر دستوراتی با MemWrite برابر 0 استفاده کنیم، **عملکرد**

دستورات بعدی که مقدار سیگنال ResultSrc آن‌ها برابر 01 است، که یعنی از data memory مقداری را می‌خوانند، تحت تاثیر قرار می‌گیرد. به طور خاص، دستور `lw` احتمالا اشتباه کار خواهد کرد. زیرا ممکن است مقداری که این دستور می‌خواند، به طور اشتباهی در دستورات قبل با مقداری نامشخص عوض شده باشد و به جای مقدار درست ذخیره شده در حافظه، مقداری تصادفی را در register file قرار دهد.

سوال ۳ - ج)

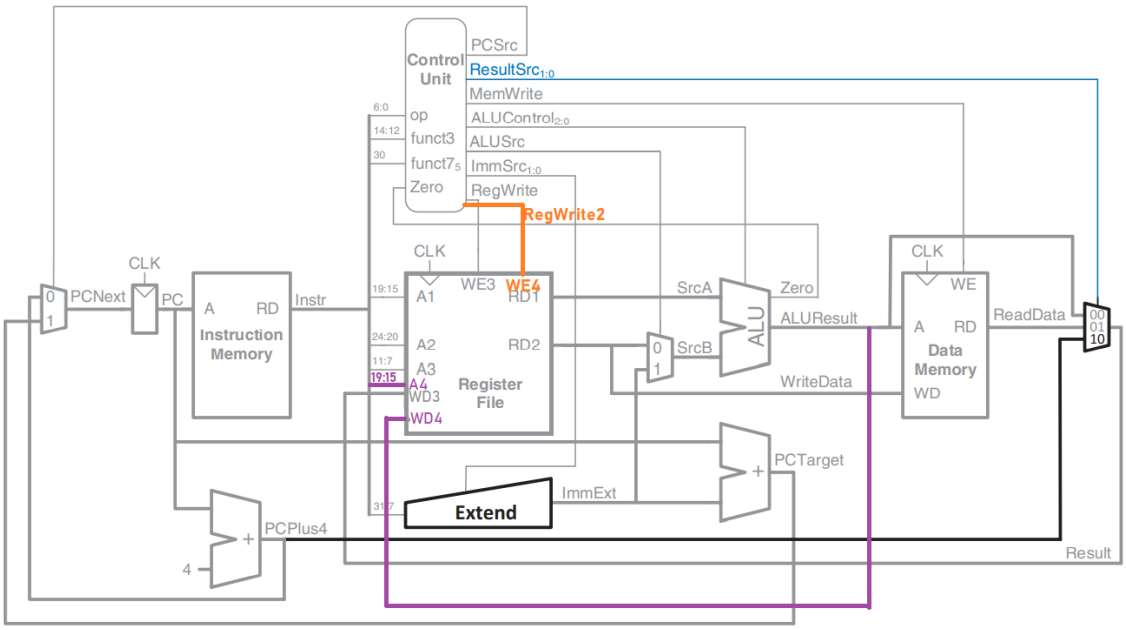
خراب بودن سیگنال $ALUOp_1$ باعث می‌شود عملکرد ALU تحت تاثیر قرار بگیرد. **دستوراتی که دارای ALUOp با مقدار 10 هستند، درست کار خواهند کرد؛** چون مقدار $ALUOp_1$ آن‌ها در حالت عادی نیز 1 است.

برای دستوراتی که ALUOp آن‌ها مقدار 01 دارد، خروجی واحد ALU Decoder نامشخص خواهد بود، در حالت خرابی، سیگنال 11 به عنوان ورودی ALUOp به واحد ALU Decoder داده می‌شود و چون در جدول درستی ورودی 11 نداریم، خروجی کاملا به شیوه پیاده‌سازی مدار بستگی خواهد داشت و نمی‌توان روی مقدار آن حساب کرد. در نتیجه **دستورات با ALUOp با مقدار 01 احتمالا به مشکل خواهند خورد.**

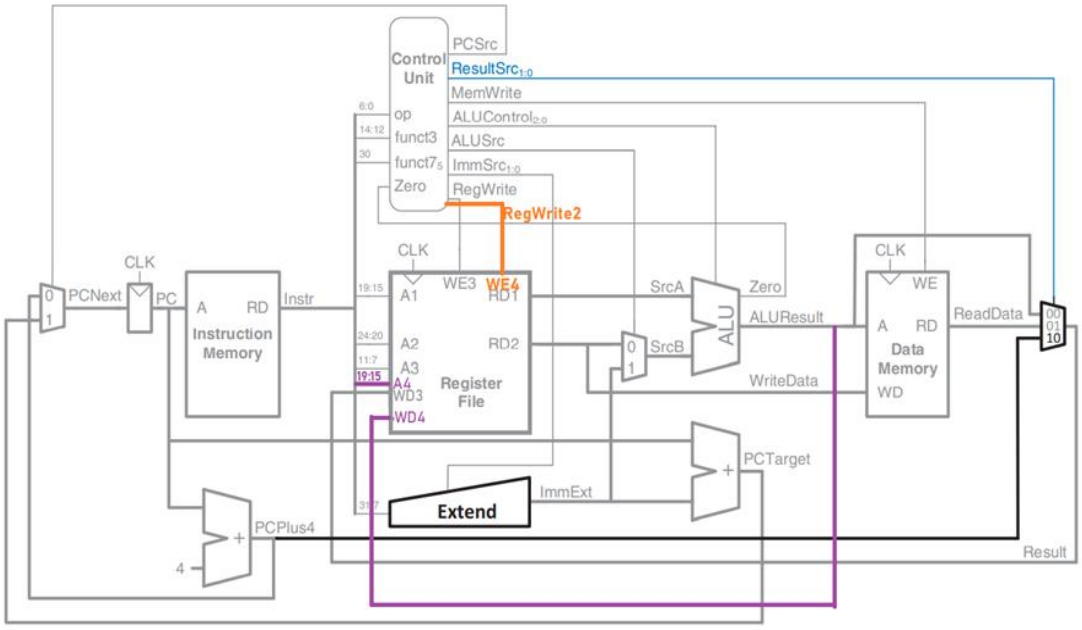
برای دستورات با ALUOp با مقدار 00، خرابی این سیگنال باعث می‌شود مقدار 01 به واحد ALU Decoder داده شود، و این واحد نیز بر اساس مقدار `funct3` عملکرد ALU را مشخص می‌کند. برای این دستورات، ALU باید عمل جمع را انجام بدهد، اما ممکن است مقدار `funct3` به گونه‌ای باشد که ALU Decoder عملکرد جمع را انتخاب نکند. در نتیجه **دستورات با ALUOp با مقدار 00 نیز احتمالا به مشکل خواهند خورد.**

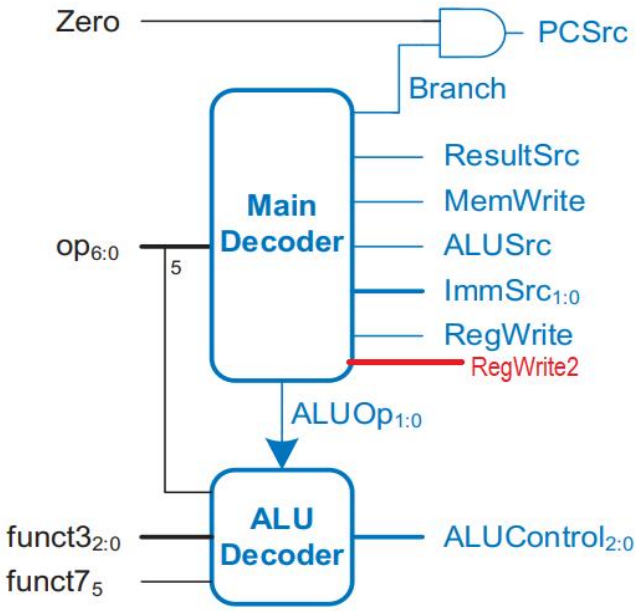
2-الف)-خیر نیازی به اضافه کردن بلوک منطقی جدیدی نیست. زیرا ما باید در دو رجیستر در یک سایکل بنویسیم و رجیستر فایل ما در حال حاضر فقط پورت برای نوشتن در یک رجیستر را دارد. دردستور *lw rd, imm(rs)* با استفاده از ALU، Offset(imm) را با آدرس پایه جمع می‌کنیم . حال به ادرس تولید شده در مموری می‌رویم و در اقداماتی نتیجه را در رجیستر rd می‌نویسیم. در دستور addi rs, rs, imm، می‌بینیم که نتیجه عملیات انجام شده در ALU در رجیستر rs نوشته می‌شود. پس طبق مرجع هریس باید در بیت های 15 تا 19 نوشته شود. رجیستر A4 را که بیت های 15 تا 19 را پوشش می‌دهد، قرار می‌دهیم. یک پورت جدید برای نوشتن در رجیستر سورس، در register file اضافه می‌کنیم(مثلا نام آن را WD3 می‌گذاریم) که در رجیستر A4 ، مقدار گذاشته شده در پورت را بنویسد و ALUResult را به ان وصل می‌کنیم و با اضافه کردن یک سیگنال به واحد کنترل که به رجیستر فایل وصل باشد(مثلا RegWrite2)، نوشتن در رجیستر سورس را کنترل می‌کنیم.

2-ب) اضافه شدن پورت جدید WD4 و ریجسترA4 به رجیستر فایل (بخش بنفش)



2-ج) سیگنال کنترلی RegWrite2 به بخشmain decod اضافه شده است. (بخش نارنجی)





سوال ۲ - د)

جدول درستی Main decoder به صورت زیر در می‌آید.

Instruction	Opcode	Reg Write	Reg Write2	Imm Src	ALU Src	Mem Write	Result Src	Branch	ALUOp	Jump
lw	0000011	1	0	000	1	0	01	0	00	0
sw	0100011	0	0	001	1	1	xx	0	00	0
R-type	0110011	1	0	xxx	0	0	00	0	10	0
beq	1100011	0	0	010	0	0	xx	1	01	0
I-type ALU	0010011	1	0	000	1	0	00	0	10	0
jal	1101111	1	0	011	x	0	10	0	xx	1
lwpreinc	**1111111***	**1**	**1**	**000**	**1**	**0**	**01**	**0**	**00**	**0**

* با توجه به موجود نبودن Opcode برای این دستور، سعی شده یک Opcode فرضی برای آن درنظر گرفته شود.

جدول درستی ALU Decoder تغییری نمی‌کند.

4- با کم شدن 20 پیکو ثانیه تاخیر ALU ، زمان اجرای دستور lw به شکل زیر خواهد بود:
از زمان اجرای دستور lw استفاده میکنیم چون دستوری است که بیش از بقیه دستور ها زمان برای اجرا شدن استفاده میکند.

Tc_single =Tpcq_pc + 2tmem + trfread + talu + tmux +trfsetup= 40 + 2(200) + 100 + 100 + 30 + 60 = 730ps

Tsingle = (100 × 10^9 instruction) (1 cycle/instruction) (730 × 10^−12 s/cycle) = 73 seconds