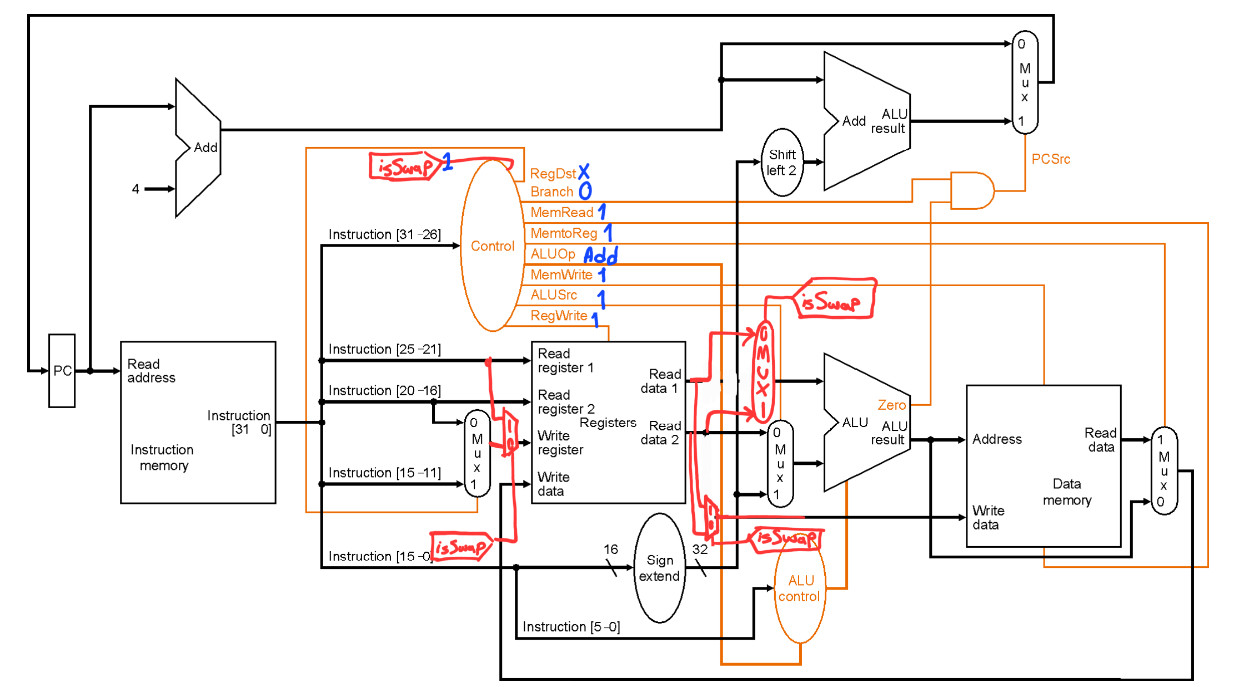
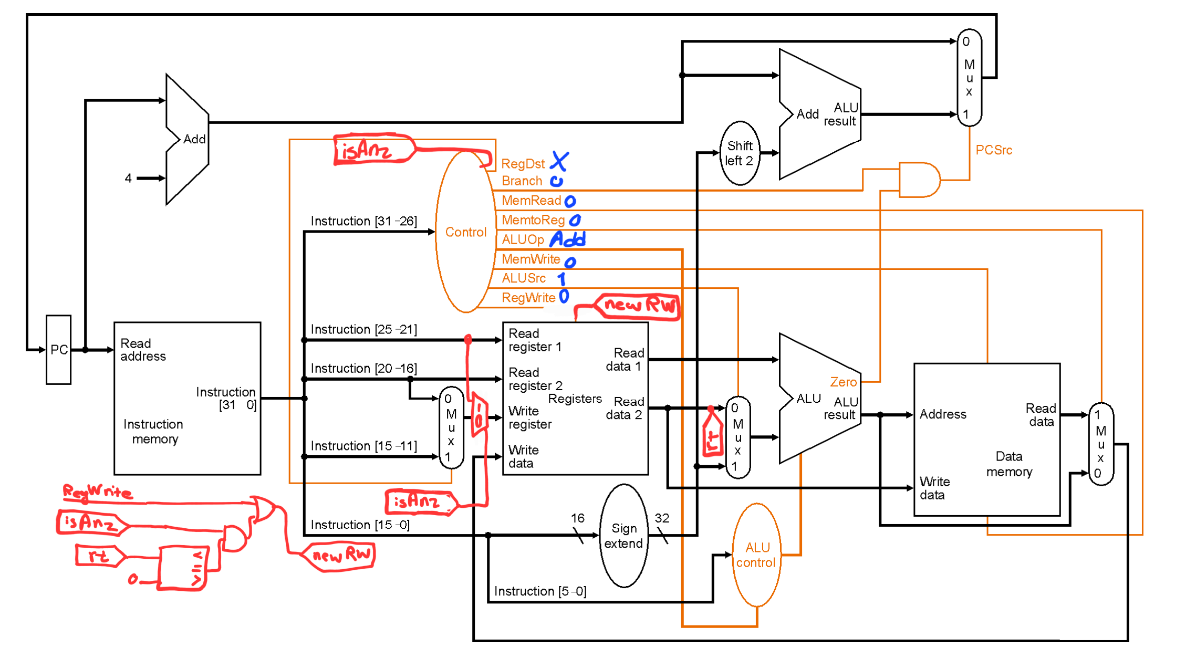
**1**. اگر در بخش swap جای rs,rt واثعا اشتباه نشده باشه، تغییرات لازم (3 تا مالتی پلکسر و سیگنال جدید isSwap که ورودی سلکت مالتی پلکسر هاست) و سیگنال های کنترلی رو در شکل زیر مشخص کردم:

****

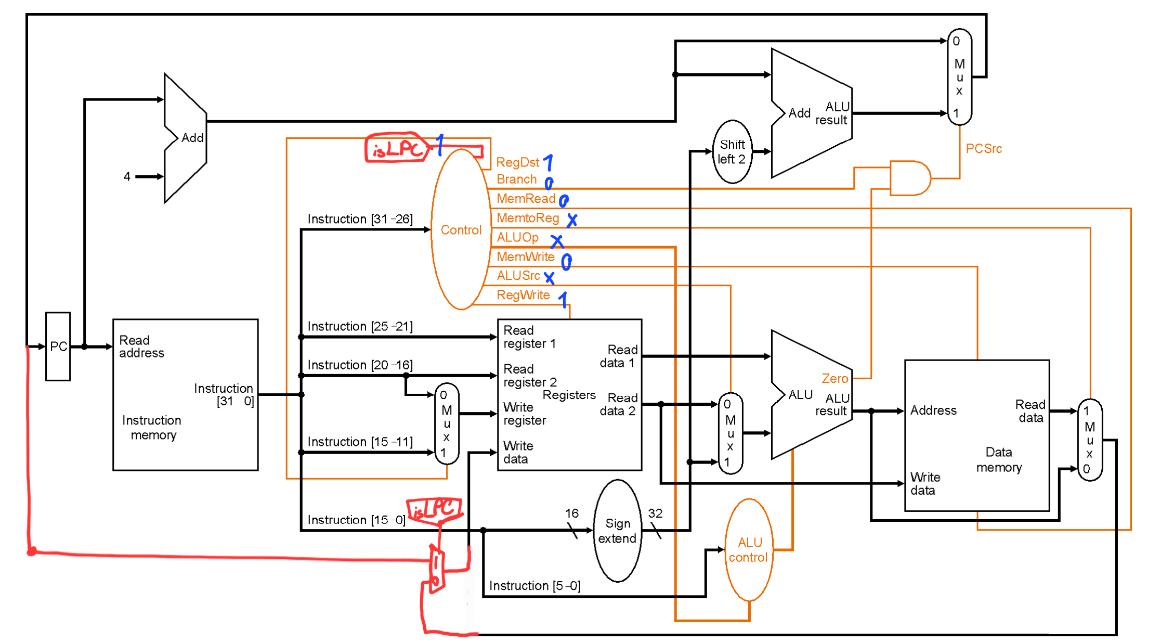
در غیر این صورت (اگر در توضیحات دستور جای rs,rt اشتباه شده باشد)، ما برای این دستور تنها نیازمند تنظیم سیگنال های کنترلی هستیم:

RegDst = 0, Branch = 0, MemRead = 1, MemtoReg = 1, ALUSrc = 1, ALU operation -> ADD, MemWrite = 1, RegWrite = 1

برای addnz، یک سیگنال جدید isAnz اضافه کردم که در صورت دیدن اپکد این دستور 1 میشود. یک مالتی پلکسر برای دادن rs به عنوان ثباتی که روی آن نوشته میشود اضافه کردم و همچنین سیگنال regWrite را با کمک تعدادی گیت و یک مقایسه کننده 32 بیتی کمی تغییر دادم( اگر isAns=1 و نتیجه comparator rt,0 بتساوی باشد، regwrite جدید فعال میشود.) . سیگنال های کنترلی و تغییرات را در شکل زیر میتوانید مشاهده کنید:



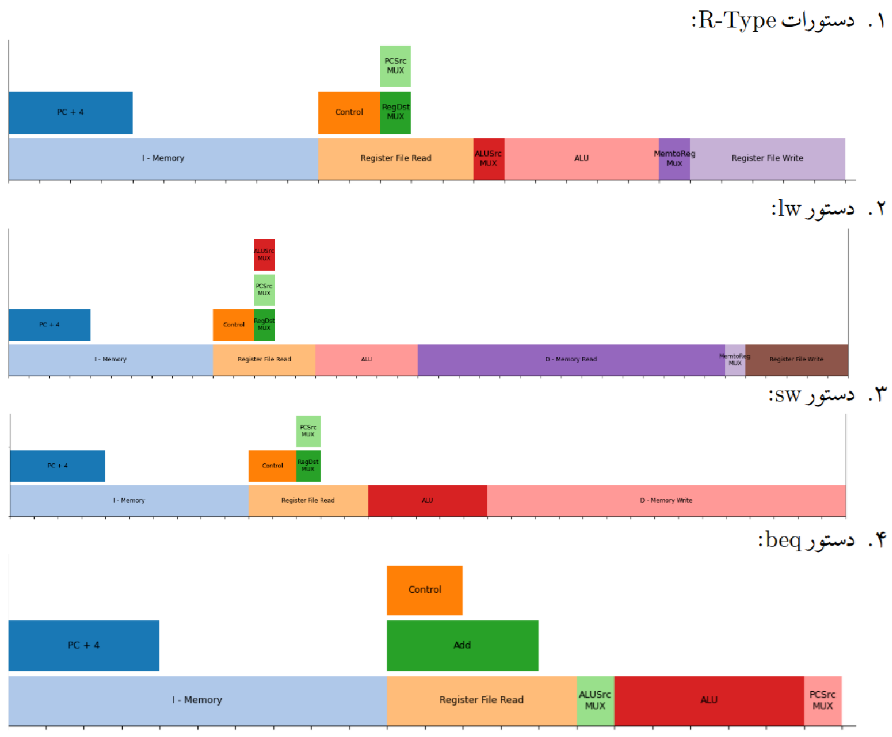
برای دستور loadPC نیاز به یک مالتی پلکسر کهPC+4 را به ورودی DataWrite متصل میکند و یک سیگنال کنترلی جدید که سلکت آن است، داریم. تغییرات و سیگنال هارا در شکل زیر میتوانید مشاهده کنید:



برای آخرین دستور یا همان brumzهم تنها تعیین سیگنال های کنترلی کافیست. این دستور مشابه beq ای است که Alu در آن بجای تفریق، جمع میکند:

RegDst = X, Branch = 1, MemRead = 0, MemtoReg = X, ALUSrc = 0, ALU operation -> ADD, MemWrite = 0, RegWrite = 0

**2. آ)** ابتدا مشابه تمرین قبل مسیر بحرانی هر نوع دستور را نوشته و با توجه به زمان های داده شده در صورت سوال، زمان هریک را محاسبه میکنیم:



R\_type: I-Mem, Regs, Mux, Alu, Mux, Regs = 500+220+100+180+100+220 = 1320ps

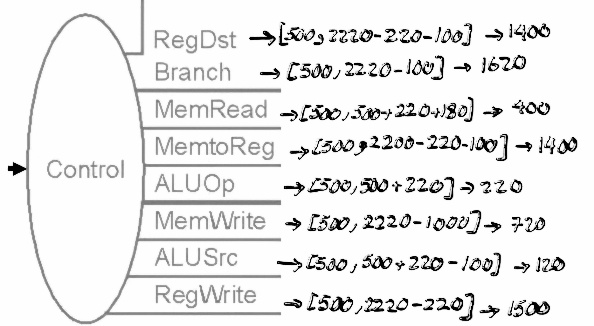
Lw: I-Mem, max(Regs, Sign-extend+Mux) , Alu, D-Mem, Mux, Regs = 500+220+180+1000+100+220 = 2220ps

Sw: I-Mem, Regs, Alu, D-Mem = 500+220+180+1000=1900ps

Beq: I-Mem, Regs, Mux, Alu, Mux = 00+220+100+180+100=1100ps

Jump: I-Mem , max(Control, Sign-extend+shift-let-2+Add), Mux >= 500+90+200+150+100=1040

بنابر این بیشترین زمان بین دستورات همانطور که میدانستیم برای lw و برابر 2220 پیکو ثانیه است. پس طول هر کلاک باید حداقل همین مقدار (برابر با 2.22 نانوثانیه) باشد.

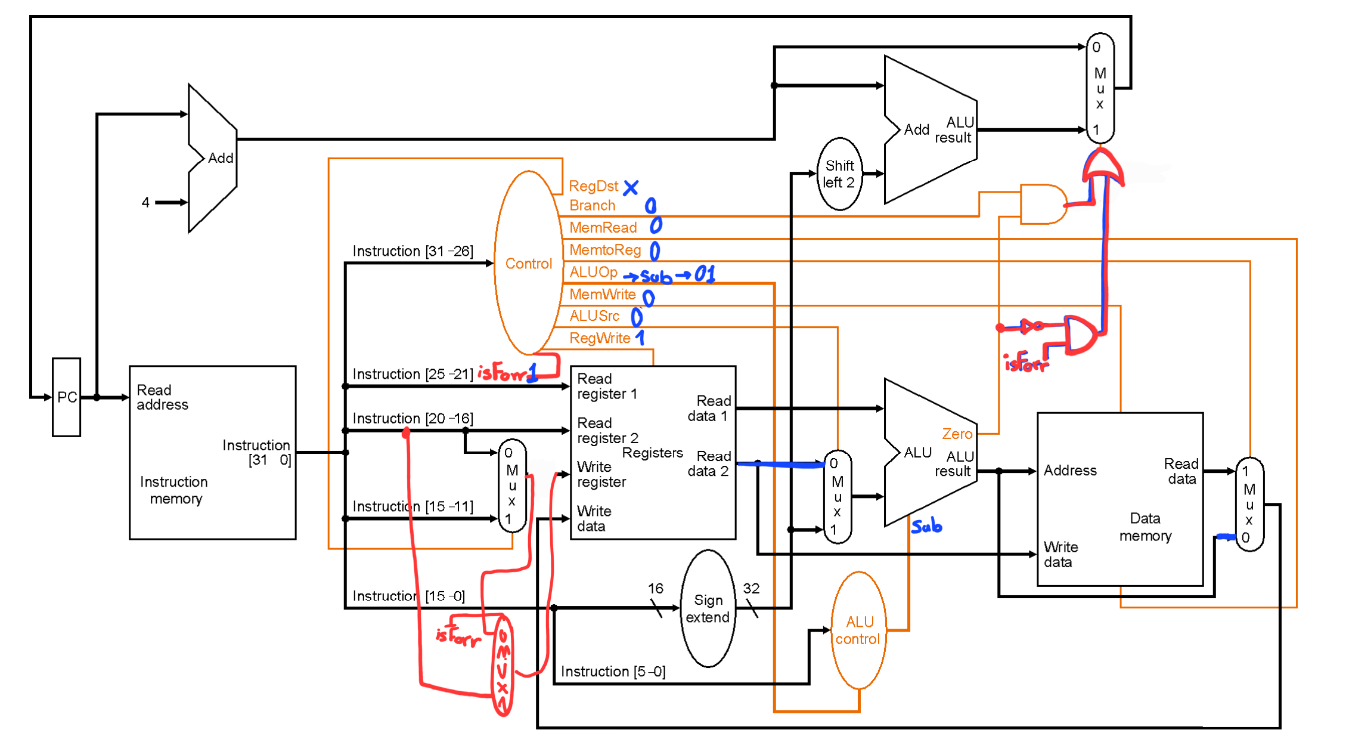
**ب)** برای آنکه سیگنال MemWrite بتواند ایجاد شود، نیاز دارد که ابتدا دستور کدگشایی شود که یعنی زمان شروع ایجاد این سیگنال از 500 پیکوثانیه بعد از لبه بالارونده کلاک است. همچنین بعد از ایجاد این سیگنال تازه روند نوشتن در حافظه که خودش 1000 پیکو ثانیه طول میکشد آغاز میشود و ما میخواهیم این 1000 پیکو ثانیه درون همان کلاک تمام شود پس این سیگنال حداکثر میتواند از بازه 500ps تا 2220-1000=1220ps را استفاده کند که 720 پیکو ثانیه است.

**ج)** من با همون ایده قسمت ب، تخمینم از بازه زمانی ای که هرکدوم از سیگنال ها میتونن برای تولید شدن استفاده کنن که در نهایت تمام دستورات در 2220 پیکو ثانیه تموم شن رو در شکل روبرو نوشتم. همانطور که مشاهده میکنید، سیگنال Branch با زمان 1620 پیکو ثانیه، بیشترین آزادی عمل را بین سیگنال ها دارد.

**د)** مشابه شکل قسمت ج، سیگنال ALUSrc باید در 120 پیکو ثانیه ایجاد شود که کمترین زمان بین زمان هاست. این زمان بخاطر آن است که این سیگنال در lw که طولانی ترین دستور است لازم میشود.

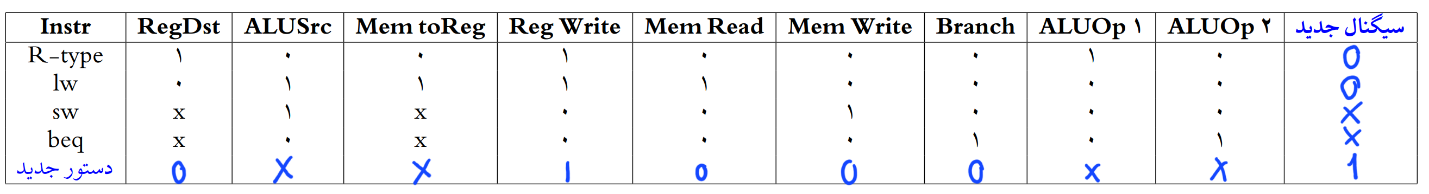
**3.آ)** چون این دستور rs, rt , immediate دارد، پس I-type است.

**ب)** تمام تغییرات لازم را در شکل زیر داده ام. توضیحات: ابتدا یک سیگنال کنترلی isForr ایجاد میکنیم که آن را به سلکت مالتی پلکسری که ثبات مقصد نوشتن را بین rs و خروجی مالتی پلکسرRegDst تعیین میکند، میدهیم. همچنین PCSrc را نیز با آن با استفاده از Not, And, Or برای حالتی که isForr=1 , zero =0 باشد فعال (برابر1) میکنیم. همچنین سیگنال های کنترلی این حالت را با رنگ ابی روی شکل نوشته ام :)



**4. آ)** این تغییرات قابلیت نوشتن یک immediate در رجیستر ها را اضافه کرده است. در صورت 1 بودن سیگنال کنترلی جدید، مقدار ورودی write data از کنار هم قرار گرفتن 16 بیت immediate و 16 بیت 0، ایجاد میشود. از آنجا که احساس میکنم آن 16 بیت 0، در نیمه کم ارزش ورودی قرار گرفته اند؛ پس این دستور برای پیاده سازی lui است: lui $rt, imm16

**ب)** تقدیم شما:



**5.** میخوام قشنگ خلاصه بنویسم.

|  |  |  |
| --- | --- | --- |
| سیگنال | Stuck at 0 | Stuck at 1 |
| RegDst | در دستورات R-type دیگه rd پر و سلام بر rt | در دستورات I-type، خدافظ rt و سلام بر مقصد رندوم |
| Jump | خداحافظ Jump | در همه دستورات بجز Jump، سلام بر PC رندوم |
| Branch | خداحافظ Branch | در همه دستورات جز Branch که zero هم 1 شده، سلام بر PC رندوم |
| MemRead | خداحافظ خواندن از حافظه (مثل lw و...) | خدافظ نوشتن برحافظه(اگر همزمان ممکن نباشن)(مثل sw) |
| MemtoReg | نوشتن از حافظه بر ثبات ها پر (مثل lw و...) | نوشتن خروجی Alu بر ثبات ها پر و سلام بر داده ی چرت و پرت روی ثبات ها (مثل R-typeها) |
| MemWrite | خدافظ نوشتن بر حافظه(اگر همزمان ممکن نباشن) (مثل sw) | خدافظ خواندن از حافظه (اگر همزمان ممکن نباشن) (مثل lw) و سلام بر داده رندوم روی حافظه! |
| ALUSrc | دیگه ورودی immediate به ALU پر و مثلا lw و sw (I-type)به مشکل میخورن | دیگه rt به ALU ورودی داده نمیشه و R-type ها غالبا به مشکل میخورن |
| RegWrite | در دستورات R-type یا lw و موارد مشابه، خداحافظ نوشتن روی ثبات ها | سلام بر داده چرتو پرت روی ثبات ها |