گزارش پروژه نهایی معماری کامپیوتر

محمد بهرامی و نوید رئیس زاده - گروه 14

توضیحات جزئیات پیاده سازی:

برای پیاده سازی پردازنده Multi-cycle microprogrammed بزرگترین چالش نحوه پیاده سازی ماژول controller بود و در پیاده سازی datapath چالش خاصی نبود و به وضوح در مرجع هریس به این قضیه پرداخته شده بود.

ماژول controller :

در حالت عادی یک پردازنده مولتی سایکل توسط یک Finite State Machine کنترل میشود اما در اینجا باید به جای آن از روش micro-programmed استفاده کرد و هر اینستراکشن توسط تعدادی میکرواینستراکشن که در یک ROM ذخیره شده اند اجرا شود.

قالب کلی کلمه کنترلی ریزدستورالعمل

پیشنهاد می‌شود ریزدستورالعمل‌ها تحت قالب کلی ریزدستورالعمل‌های افقی (شکل 19.12 مرجع استالینگز) ساماندهی و در حافظه کنترلی ذخیره شوند. در صورت نیاز به تغییر، با ذکر دلیل انجام شود. با توجه به‌این‌که برای حافظه کنترلی ROM ۳۲ خط در نظر گرفته شده است، ۵ بیت برای آدرس ریزدستورالعمل بعدی اختصاص می‌یابد.

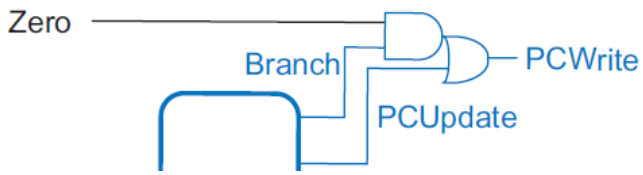
Control Signals										Branch Condition				Branch Target
pcwrite	regwrite	memwrite	irwrite	adrsrc	resultsrc [1:0]	alusrca [1:0]	alusrcb [1:0]	immsrc [1:0]	alucontrol [2:0]	OP	Funct3	Funct7b5	zero	*Micro-instruction* Address [4:0]

تغییرات نسبت به CW ای که در صورت پروژه مشاهده میشود:

به جای alucontrol مقدار aluop را ذخیره میکنیم چون مقدار alucontrol به ازای هر دستور R-type و I-type های محاسباتی فرق میکند و این موجب میشود که تعداد Control Word هایی که در ROM ذخیره می کنیم زیاد شود و همچنین هندل کردن next Instruction ها سخت می شود.

مقادیر Branch Condition نیز به عنوان ورودی ماژول داده میشوند و نیازی به ذخیره کردنشان نیست و اینکه مقدار zero بستگی به نوع اینستراکشن ندارد و به سورس رجیستر های آن اینستراکشن مربوط است.

مقدار pcwrite نیز گاهی به مقدار zero مربوط است (در beq):

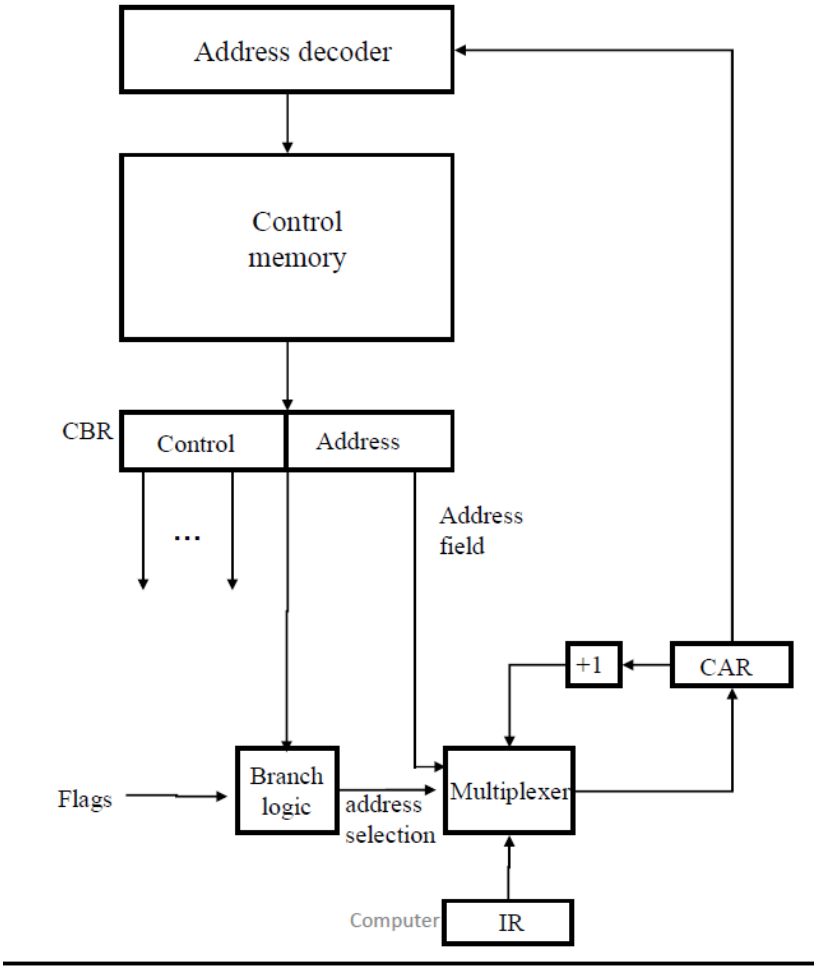


پس در CW مقادیر Branch و PCUpdate نگه داشته میشوند و بعد از خواندن مقدارشان از ROM وارد این مدار ساده میشوند تا مقدار PCWrite مشخص شود.

در نهایت CW ما 21 بیتی خواهد بود.

16 بیت سیگنال های کنترلی و 5 بیت آدرس میکرواینستراکشن بعدی.

ساختار تعیین next microinstruction:



در این ساختار نیز یک تغییر میدهیم

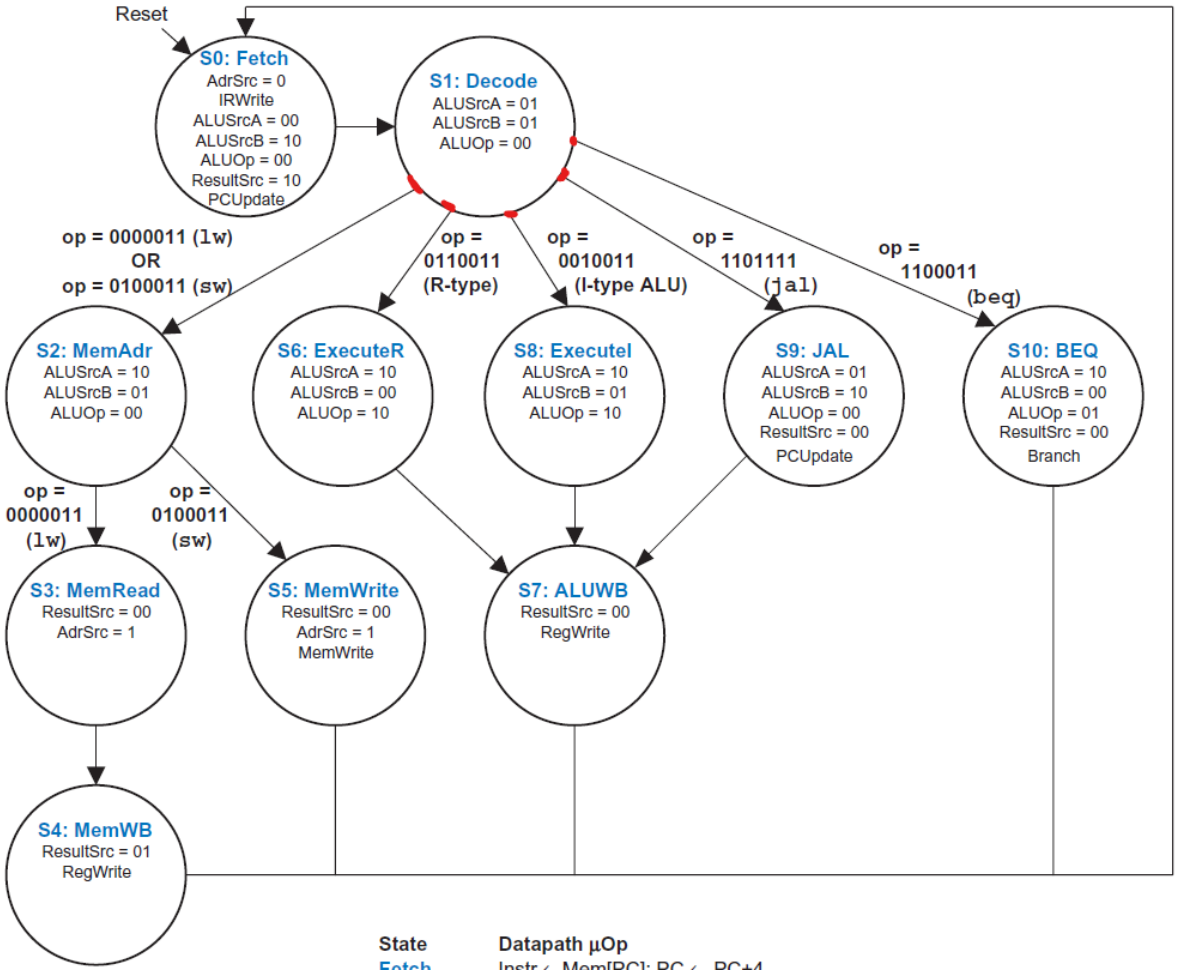
به جای به علاوه یک کردن آدرس یک ورودی دیگر به مالتیپلکسر میدهیم

برای هندل کردن استیت Decode که بعد از خودش به 5 استیت دیگر میرود اما با ساختار بالا از هر میکرواینستراکشن میتوان به دو حالت بعد رفت

راه حل:

در استیت دیکود باید بتوانیم قادر باشیم آدرس 5 استیت بعدیش رو مشخص کنیم

آدرس های ما 5 بیتی هستن



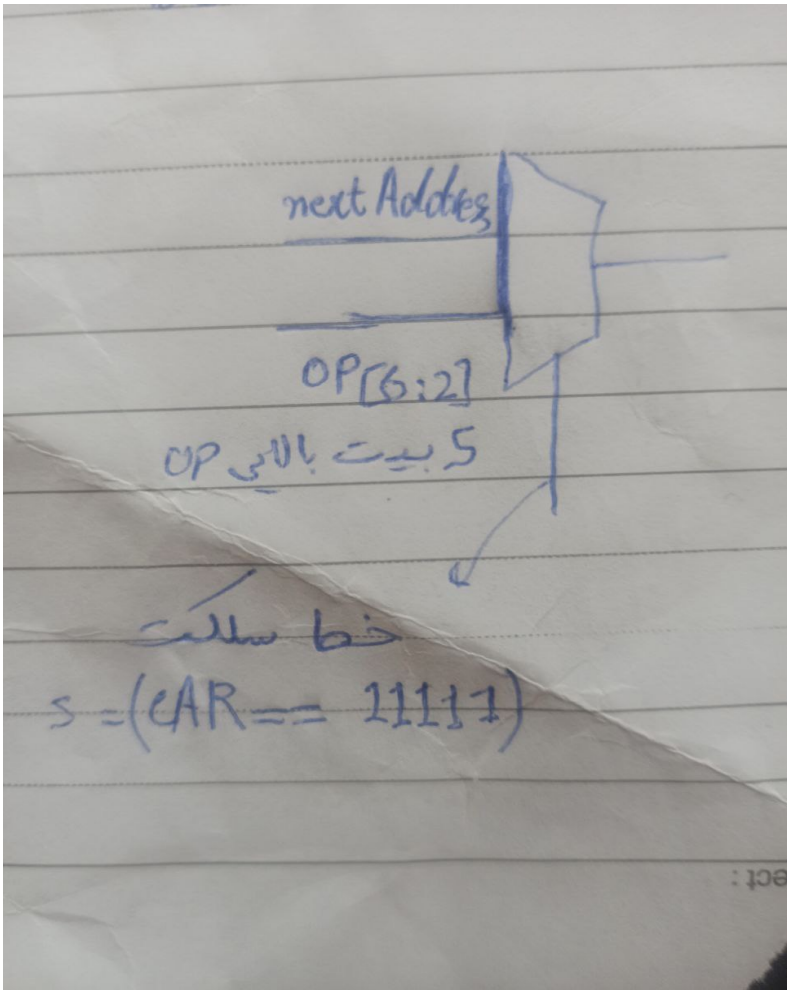
اگه به آپکد ها دقت کنیم که شرط ترنزیشن هستن متوجه میشویم که 5 بیت پر ارزش آن ها یونیک هستن و دو بیت کم ارزش همه 11 هستن

از این 5 بیت به عنوان آدرس میکرواینستراکشن بعدی استفاده میکنیم در بقیه استیت ها هم آدرس بعدی را در CW ذخیره میکنیم و حالا بین این دو مقدار با یک مالتیپلکسر انتخاب میکنیم که اگر در استیت Decode بودیم مقدار 5 بیت بالایی opcode انتخاب شود(مقدار نکست را برای CW خود این میکرو اینستراکشن برابر آدرس خودش قرار میدهیم که البته در هر صورت انتخاب نمیشود و اهمیتی ندارد چه باشد!) وگرنه نکست استیت.

نکته : برای MemAdr دو حالت پیش می آید که مقادیر سیگنال های کنترلی یکسان خواهند بود و فقط در ROM دو بار ذخیره میشود یک بار برای sw و بار دیگر برای lw .

مقدار اولیه Control Address Register برابر آدرس استیت Fetch خواهد بود.

سیگنال های کنترلی هر میکرواینستراکشن را بر اساس شکل های 32 34 36 38 40 42 فصل هفت به دست می آوریم و در رام ذخیره میکنیم.



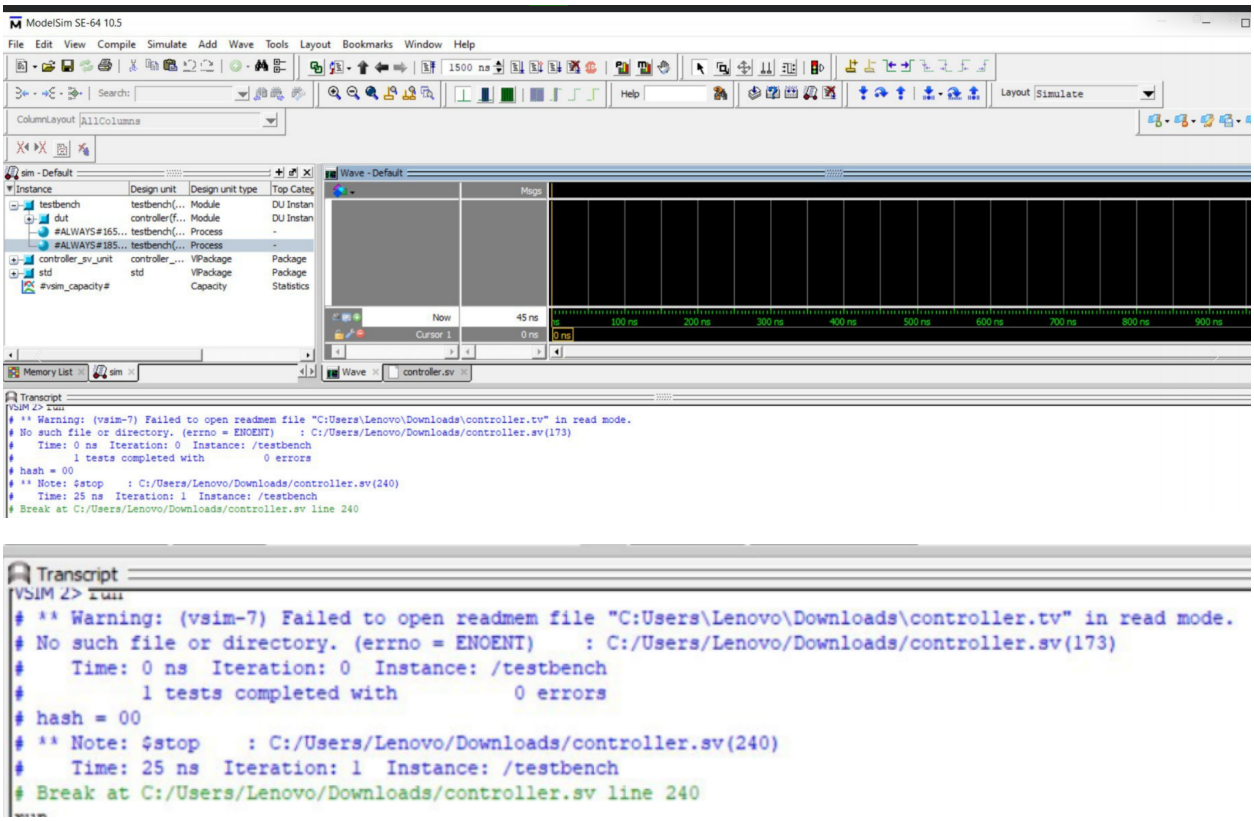

```
rom[30]  = 21'b0_0_0_0_1_0_00_00_00_00_00_11111; // Fetch     -> Decode
rom[31]  = 21'b0_0_0_0_0_X_XX_XX_XX_00_00_11111; // Decode    -> unkown
rom[0]   = 21'b0_0_0_0_0_X_XX_10_01_00_00_01110; // MemAdr for lw -> MemRead
rom[8]   = 21'b0_0_0_0_0_X_XX_10_01_00_00_11100; // MemAdr for sw -> MemWrite
rom[12]  = 21'b0_0_0_0_0_X_XX_10_00_XX_10_01010; // ExecuteR  -> ALUWB
rom[4]   = 21'b0_0_0_0_0_X_XX_10_01_XX_10_01010; // ExecuteI  -> ALUWB
rom[27]  = 21'b1_0_0_0_0_X_00_01_10_11_00_01010; // JAL       -> ALUWB
rom[24]  = 21'b0_1_0_0_0_X_00_10_00_10_01_11110; // BEQ       -> Fetch
rom[14]  = 21'b0_0_0_0_0_1_00_XX_XX_00_XX_01111; // MemRead   -> MemWB
rom[15]  = 21'b0_0_1_0_0_X_01_XX_XX_00_XX_11110; // MemWB     -> Fetch
rom[28]  = 21'b0_0_0_1_0_1_00_XX_XX_01_XX_11110; // MemWrite  -> Fetch
rom[10]  = 21'b0_0_1_0_0_X_00_XX_XX_XX_XX_11110; // ALUWB     -> Fetch
nd
```

در قسمت کامنت شده میکرواینستراکشن بعدی مشخص شده است.

با دادن آدرس ذخیره شده در CAR به رام CW را در بافر CBR ذخیره میکنیم و آدرس بعدی و سیگنال های کنترلی را به دست می آوریم

```
assign {PCUpdate , Branch, RegWrite, MemWrite, IRWrite, AdrSrc, ResultSrc, ALUSrcA, ALUSrcB, ImmSrc, ALUOp, nextAddress} = CBR;
```

حاصل شبیه سازی controller :



پیاده سازی datapath :

در این مرحله به چالش خاصی برنخوردیم

بر اساس شکل 27 فصل هفت طراحی رو انجام دادیم

روند کار بسیار مشابه طراحی دیتاپث در تمرین چهارم برای پردازنده single-cycle بود

و ماژول ها آماده بودن و کدشون رو در اختیار داشتیم.

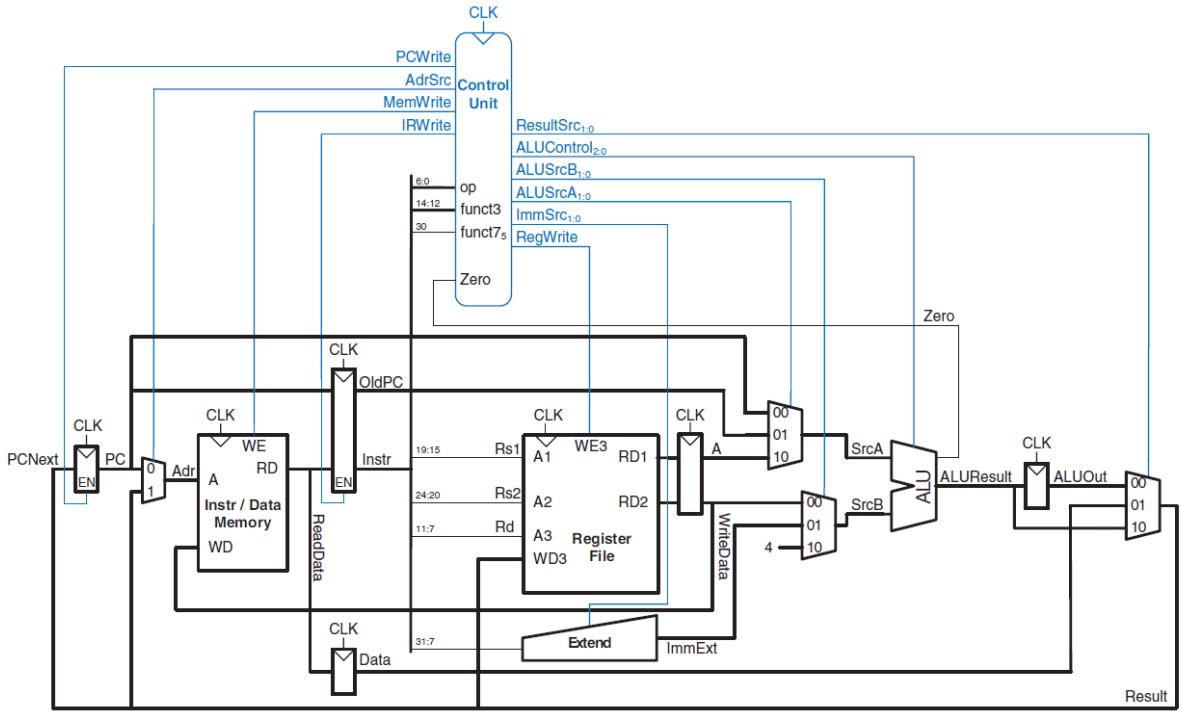
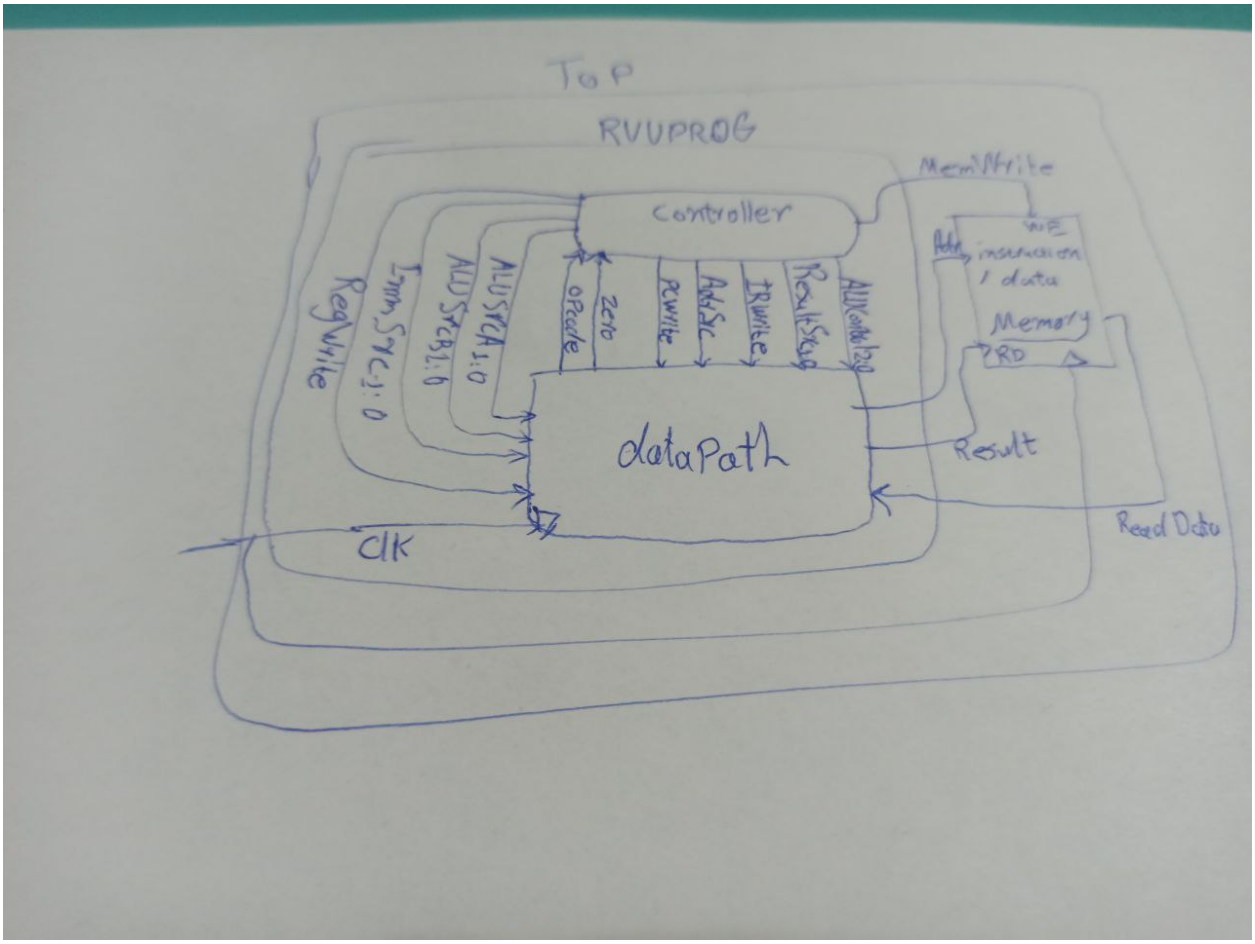


Figure 7.27 Complete multicycle processor

تفاوت بزرگ وجود تنها یک مموری هست که خروجی های دیتاپث به آن متصل میشوند و تعدادی رجیستر که مقادیر رو برای اجرای سایکل بعدی داخل خودشون نگه میدارن

تغییر یافته شکل 63 فصل 7 :


Top
RVUPROG
MemWrite
controller
WE
Adr
instruction
/ data
Memory
RD
ALUSrcA 1:0
ALUSrcB 1:0
Imm Src 1:0
RegWrite
Zero
oPcode
PcWrite
AdrSrc
IRWrite
ResultSrc
ALUControl 2:0
datapath
Result
Read Data
clk