به نام خدا

موضوع: گزارش پروژه Verilog MIPS pipeline CPU **لینک پروژه:** https://git.io/fjaRD

روال کار پایپ لاین:

تفاوت پروژه pipelineبا single cycle :

در single cycle در هر کلاک یک دستور وارد میشود و اجرا میشود و پس از پایان آن در کلاک بعدی دستور بعد وارد میشود با دوره ی پالس طولانی در pipe line 5 مرحله در هر دستور موجود است . شامل :

Inst fetch–inst decode–execution–mem access–write back

که هر مرحله در یک کلاک اما برخلاف single cycle در دوره پالس کوتاه انجام میشود که هدف از این طراحی بهبود عملکرد نهایی پردازنده است.

بررسی کد:

4 رجیستر mem2wb—exe2mem—id2exe—if2id داریم هرکدام از این رجیسترها به همراه pc یک کلاک دارند و به لبه بالا رونده حساسند.

Reg file هم یک کلاک حساس به لبه پایین رونده دارد که عمل نوشتن درآن انجام میشود.

آدرس دستور از pc به instmem میرود.

pc با 4 جمع می شود و با خروجی instmem به if2id میروند در لبه بالا رونده کلاک خروجی ها به ماژول رجیستر میروند.

در ماژول رجیستر یک آرایه 32 بیتی داریم که با لبه بالا رونده ورودی های ماژول مقدار read reg1,read reg2 در خروجی های read data1 ,read data2 قرار داده میشود.

در لبه پایین رونده کلاک اگر دستور regwrite که از کنترلر می آید برابر1 بود مقدار write dataرا در write reg قرار میدهد.

Sign extended را در 4 ضرب میکنیم(دوتا به چپ شیفت میدهیم) بعد از آن با pc قبلی جمع میشود.

در مرحله بعد در مرحله مقدار دهی اولیه خروجی ها را 0 میدهیم سپس با لبه بالا رونده کلاک خروجی کلاک پایپ قبلی را به ماژول بعدی انتقال میدهیم.

این مرحله یک mux دارد که تعیین کننده ی ورودی alu است که از بین sign–extended و read data2 این ورودی را تعیین میکنیم.

برای تعیین انجام عمل sub, add, or, and که alu باید انجام دهد، alu controller با توجه به ورودی func که inst[0:6]میباشد و آرایه کنترلی aluop تصمیم میگیرد و نتیجه را به alu ارسال میکند.

در مرحله بعد در مرحله مقدار دهی اولیه خروجی ها را 0 میدهیم سپس با لبه بالا رونده کلاک خروجی کلاک پایپ قبلی را به ماژول بعدی انتقال میدهیم

در ماژول data mem دو خط کنترلی برای تصمیم بر خواندن و نوشتن وجود دارد (mem read , mem write)

در مرحله نیزمانند مراحل پیشین عمل میکنیم

در این مرحله یک mux وجود دارد برای تعیین Itype یا Rtype بودن با خط کنترلی mem to reg بین خروجی data memو alu تصمیم میگیرد

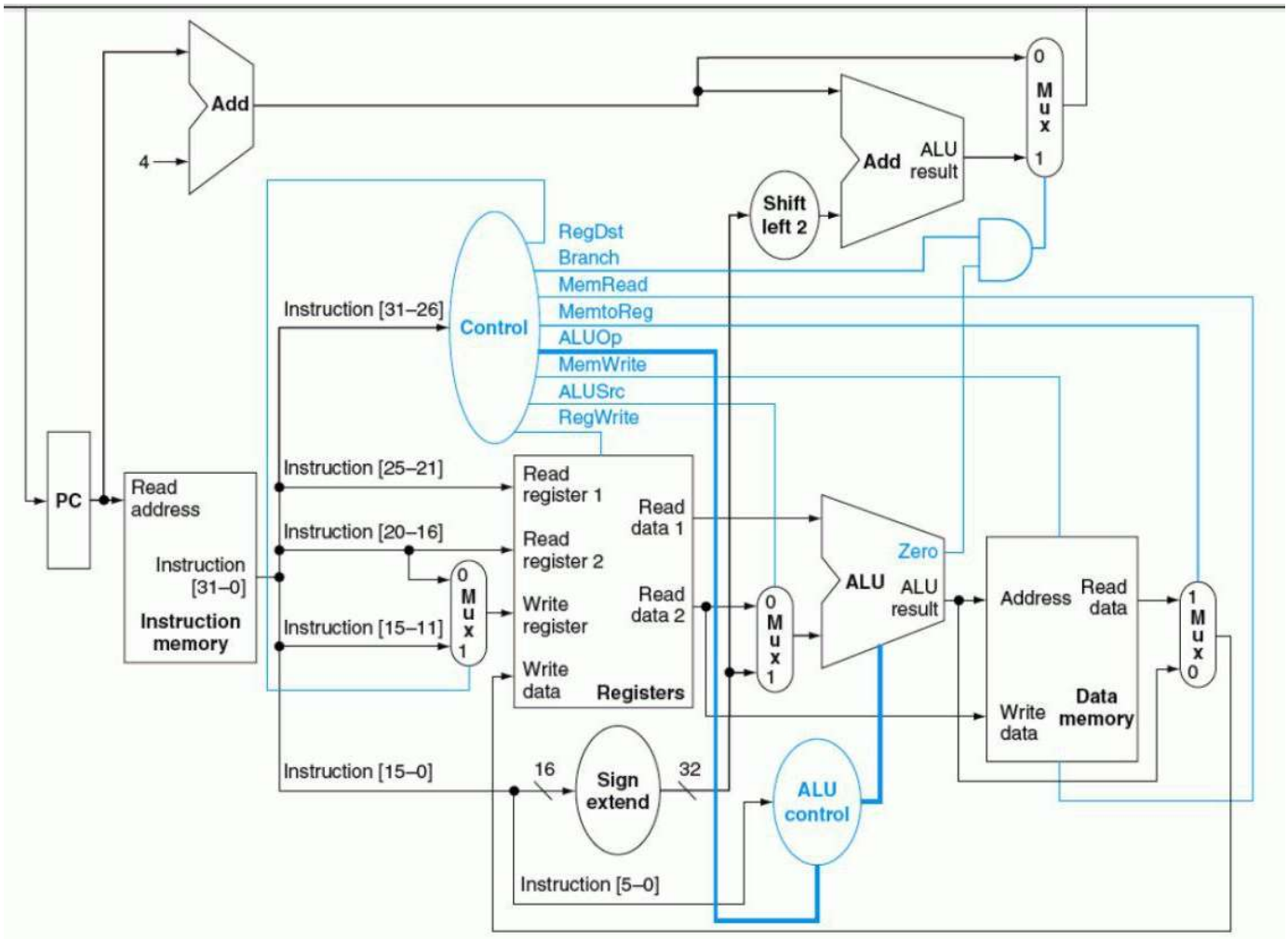
سپس خروجی را در write data واقع در ماژول رجیستر قرار میدهد .

بررسی مسیر داده ها ، دستور ها و کد های واقع در alucontroller و controller:

درصفحه های آتی ابتدا یک بررسی مختصر ی از single cycle داریم چون اول این پروژه را انجام دادیم و سپس با توجه به تفاوت هایی که ذکر شد آن را تغییر دادیم و به pipeline رسیدیم.

آدرس پروژه singlecycle: https://git.io/fjKne

تنها نکته ای که در تصویر موجود نیست کنترل هازارد beq است که در کد هست اما در تصویر اعمال نشده و این کنترل بدین صورت هست که مقادیر خروجی ماژول registers memory را در بخش decode مقایسه میکنیم و به گیت and در بخش mem میفرستد.

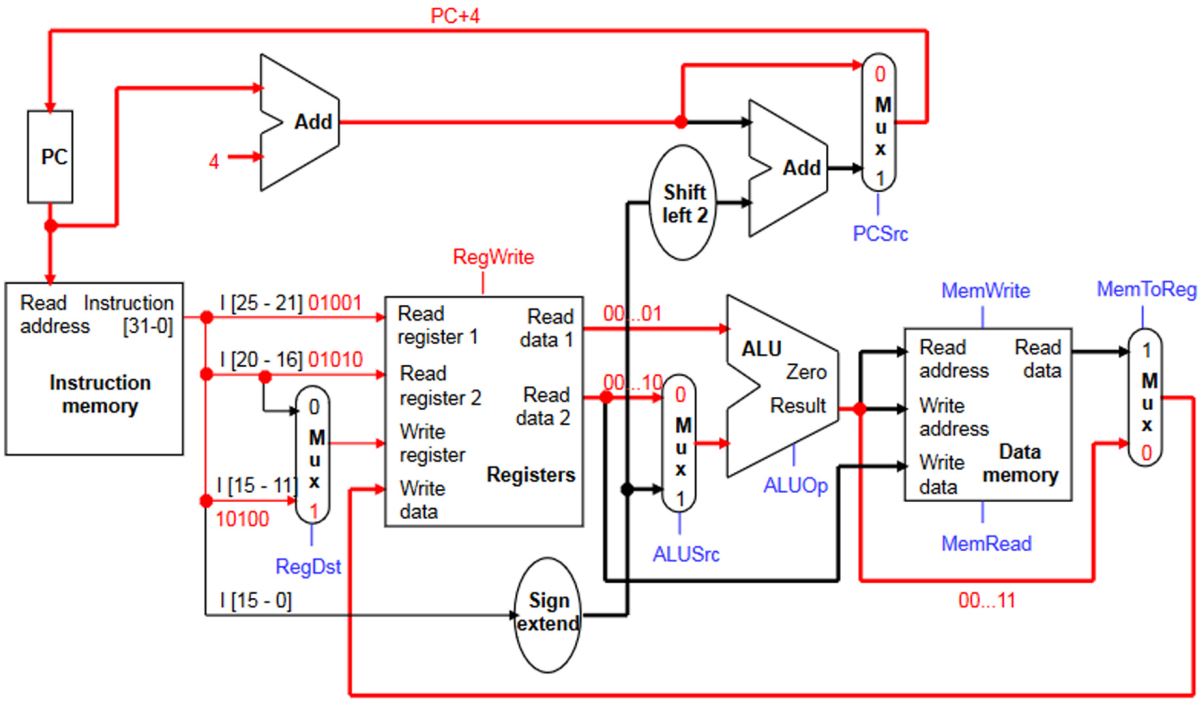


مسیر داده single cycle

The add moving through the datapath

Consider the instruction add $t1, $t1, $t2

000000	01001	01010	01001	00000	1000000
opcode	rs	rt	rd	shamt	func

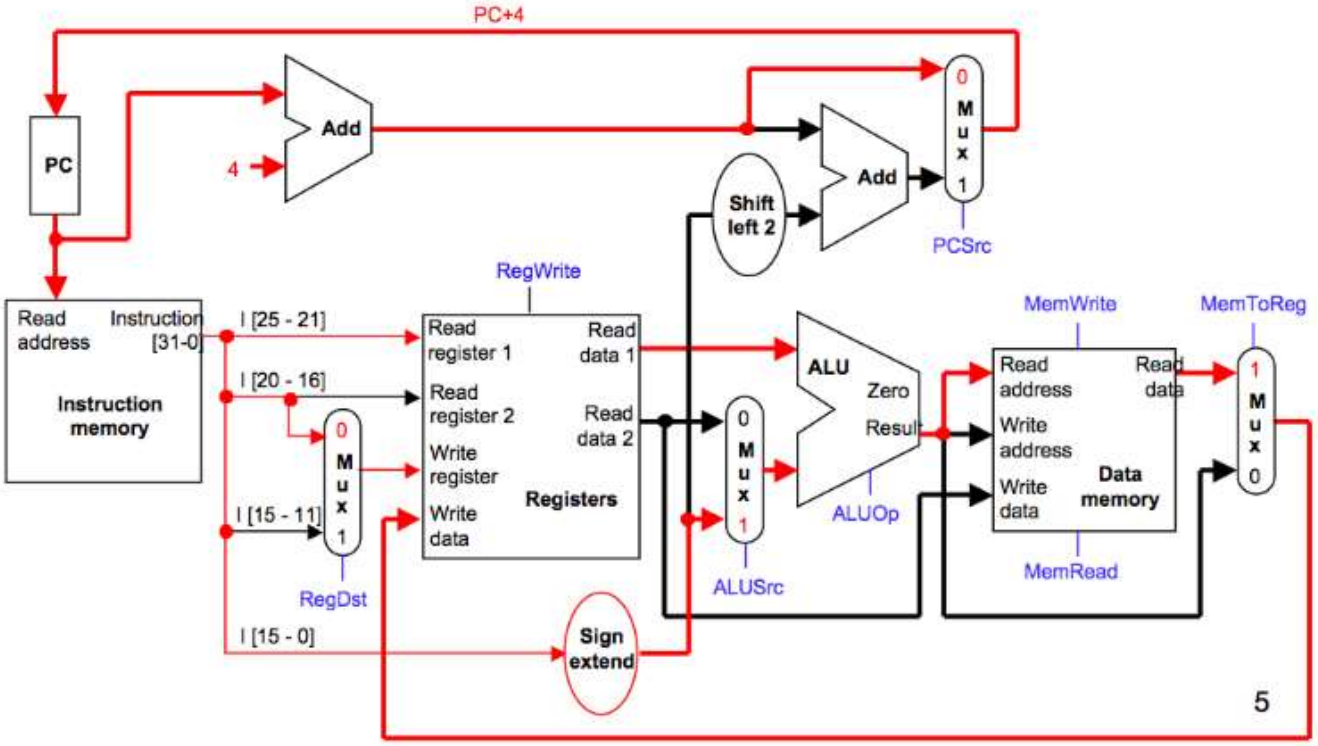


مسیر داده در دستورهای محاسباتی

The lw moves through the datapath

Consider lw $t0, –4($sp) execution

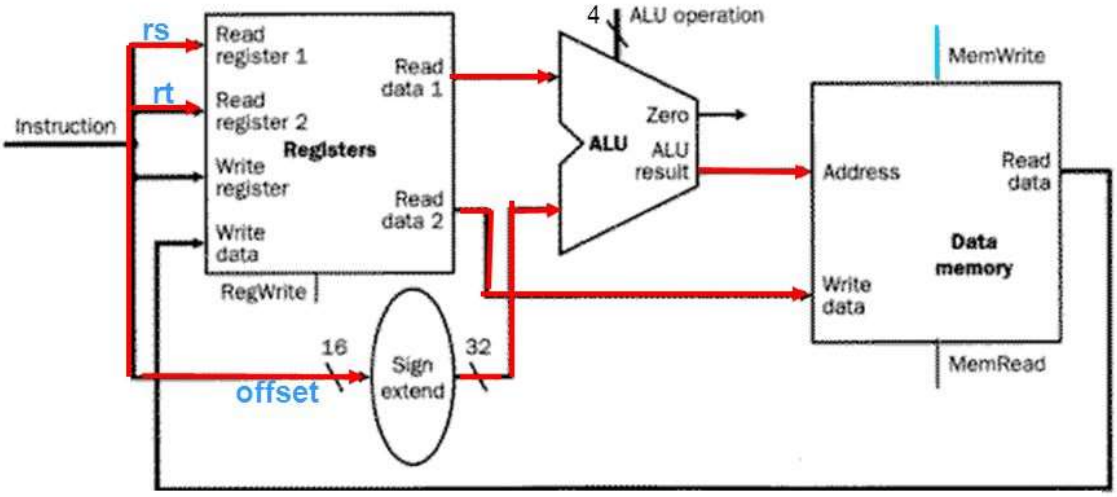
100011	11101	01000	1111 1111 1111 1100



مسیر داده LW

Store instruction datapath

sw $t0, offset($t1): Mem[$t1 + se(offset)]=$t0



I-format

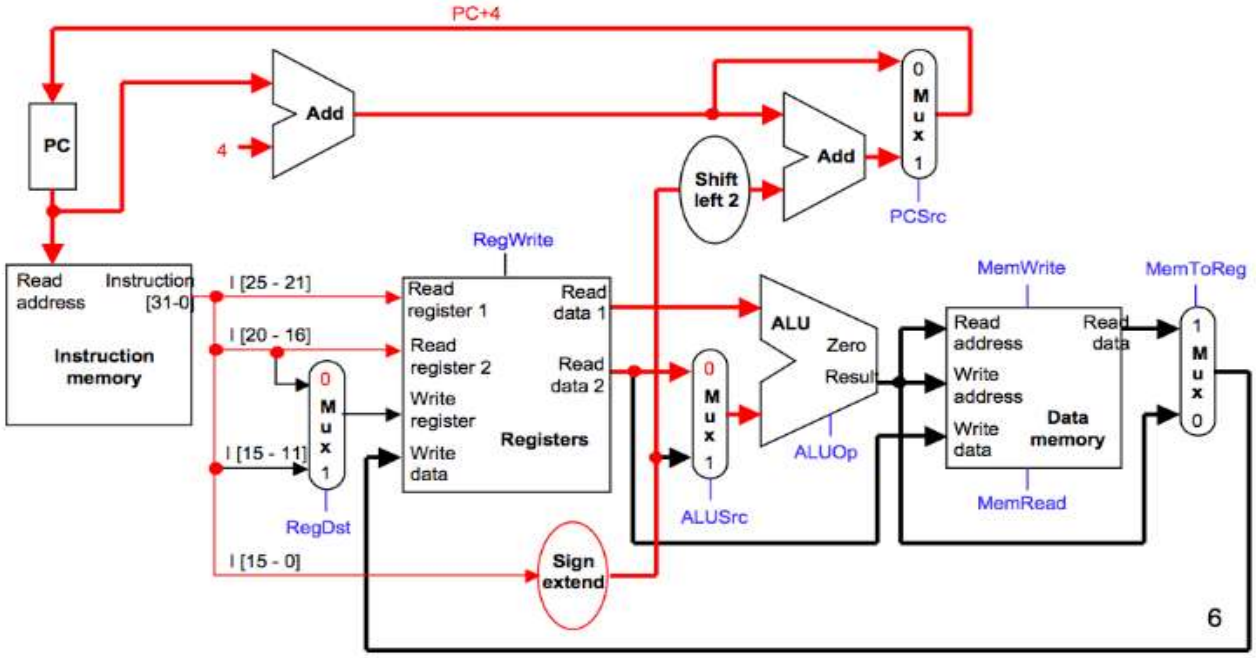
31-26	25-21	20-16	15-0
opcode	rs	rt	offset

مسیر داده SW

The beq moves through the datapath

Recall beq $at, $0, offset compute branch addr: PC + 4 + (offset x 4)

000100	00001	00000	0000 0000 0000 0011



مسیرداده beq

Inst	Reg-Dst	ALU-Src	Mem-toReg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp
R-	1	0	0	1	0	0	0	1 0
lw	0	1	1	1	1	0	0	0 0
sw	X	1	X	0	0	1	0	0 0
beq	X	0	X	0	0	0	1	0 1
sll								

Inst	Reg-Dst	**ALU-Src**	Mem-toReg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp
R-	1	0 0	0	1	0	0	0	1 0
lw	0	0 1	1	1	1	0	0	0 0
sw	X	0 1	X	0	0	1	0	0 0
beq	X	0 0	X	0	0	0	1	0 1
sll	**1**	**1 0**	**0**	**1**	**0**	**0**	**0**	**1 0**

```
Switch case (function)
6'b100000 add
6'b100100 and
6'b100101 or
6'b100010 sub
```