

同济大学计算机系

计算机组成原理实验报告



学 号 2152809

姓 名 曾崇然

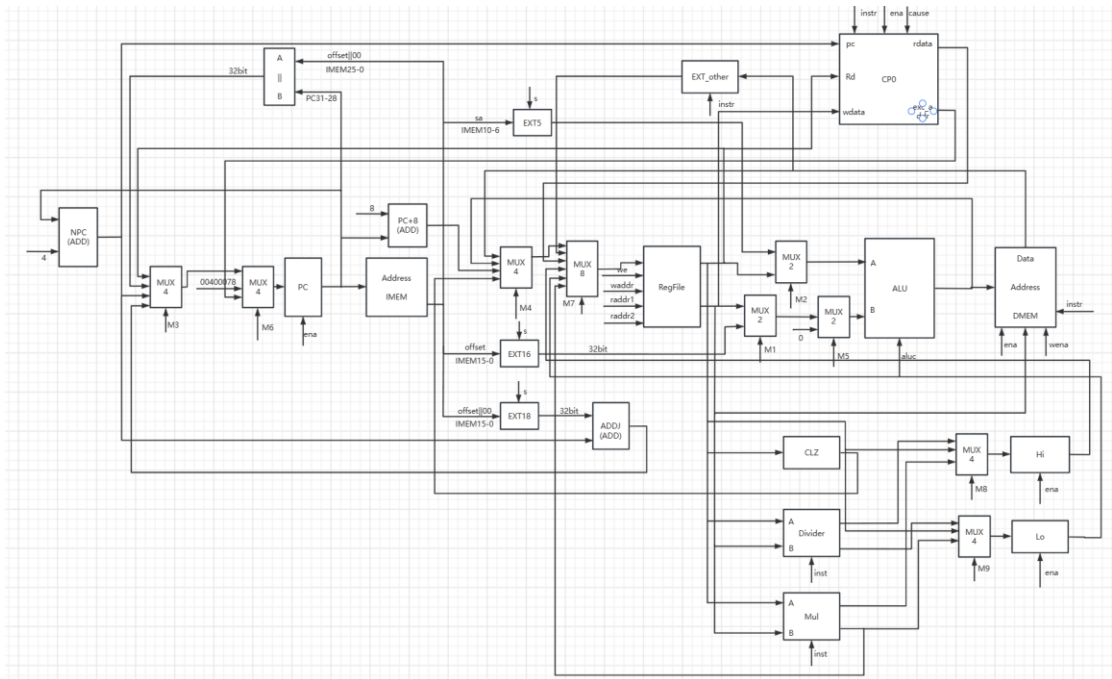
专 业 计算机科学与技术

授课老师 张冬冬老师

一、实验内容

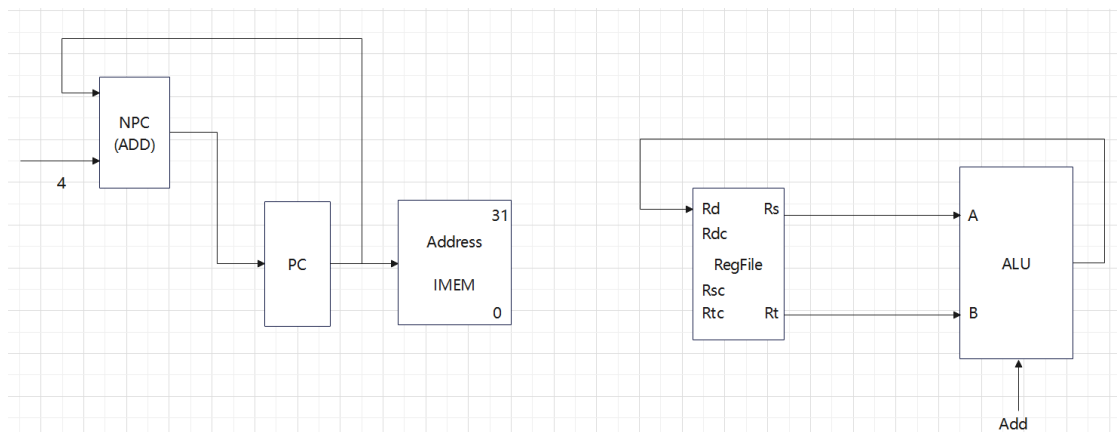
使用 Verilog HDL 语言实现 54 条 MIPS 指令的 CPU 的设计和 仿真

二、数据通路构建

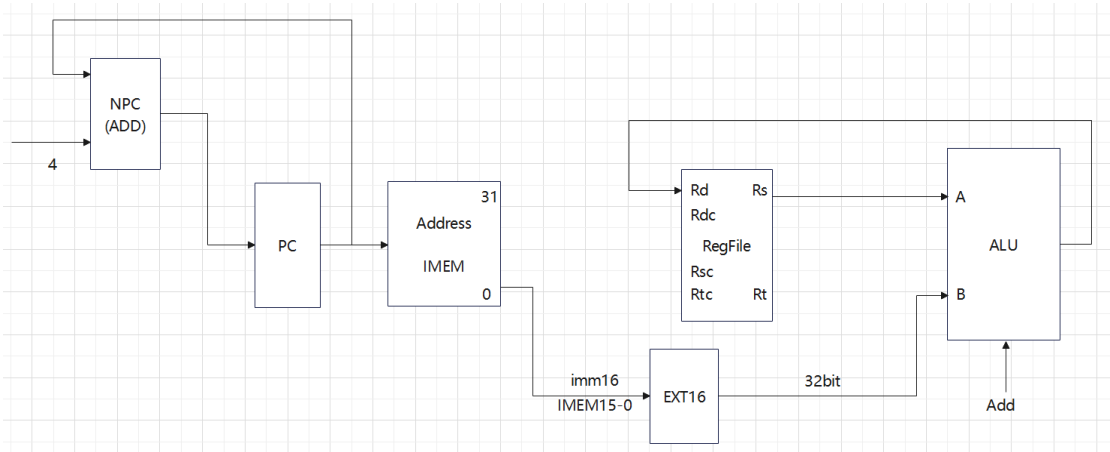


三、数据通路构建

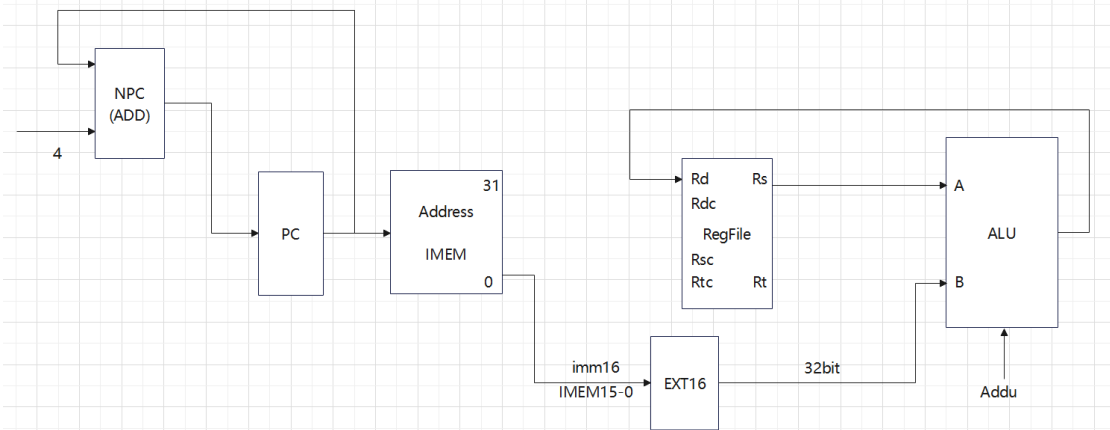
1. add:PC 的值送入指令存储器, PC+4 的结果送入 PC; 寄存器堆中两个寄存器的值送入 ALU, 计算结果送入指定的寄存器



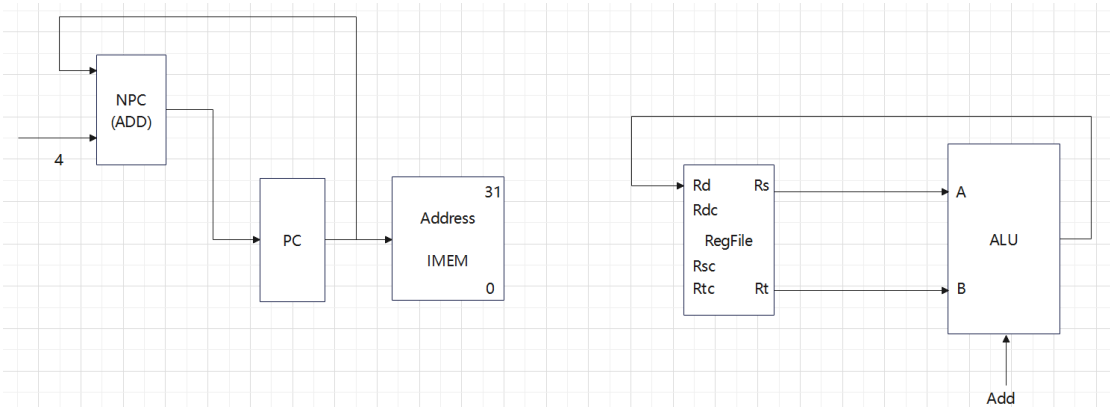
2. **addi**:PC 的值送入指令存储器, PC+4 的结果送入 PC; 指令存储器选出值的 15 位到 0 位经位扩展送入 ALU 的一端, 寄存器堆中的一个寄存器的值送入 ALU 的另一端, 计算结果送入指定的寄存器



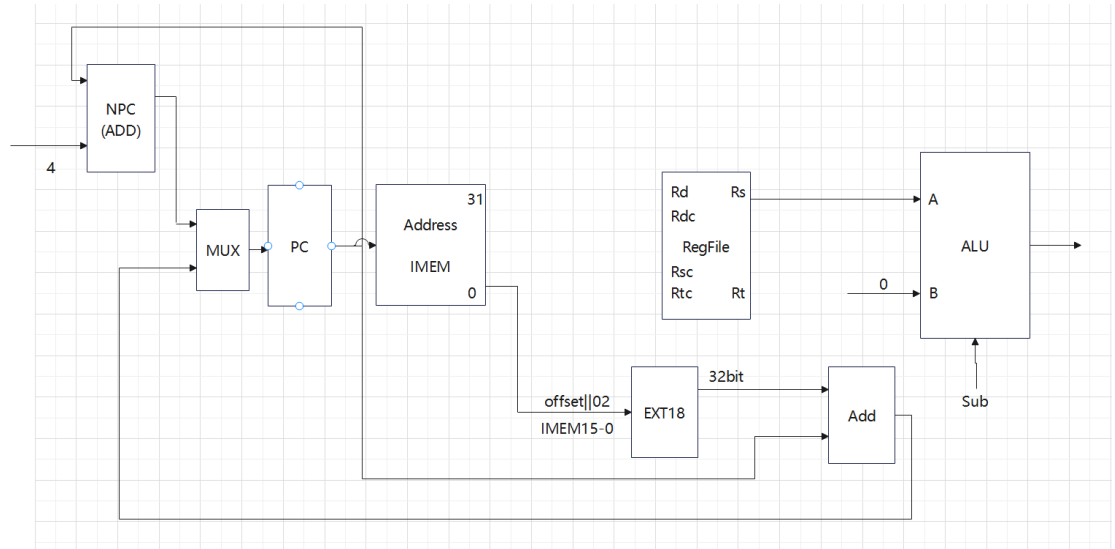
3. `addiu`: PC 的值送入指令存储器，PC+4 的结果送入 PC；指令存储器选出值的 15 位到 0 位经位扩展送入 ALU 的一端，寄存器堆中的一个寄存器的值送入 ALU 的另一端，计算结果送入指定的寄存器



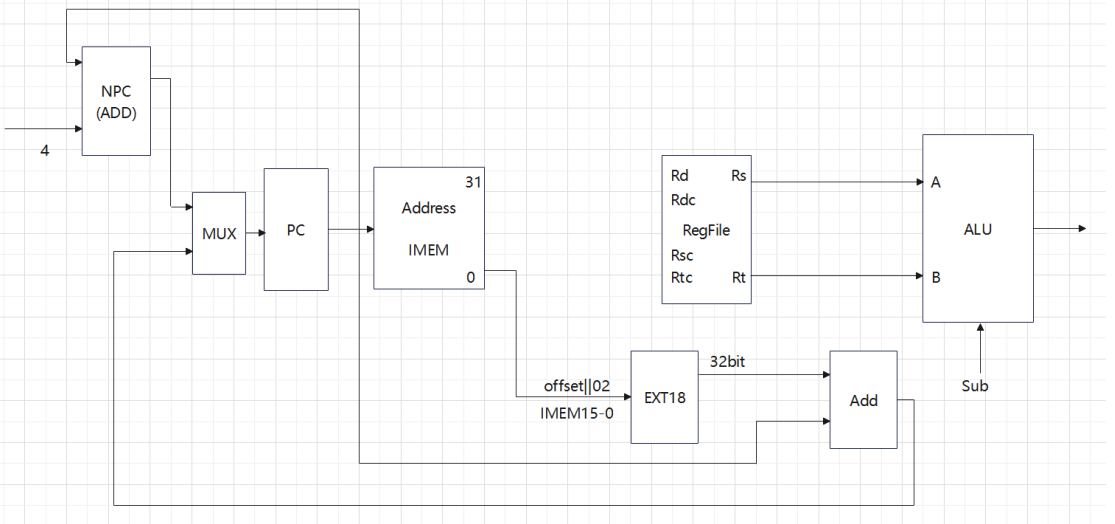
4. `addu`: PC 的值送入指令存储器，PC+4 的结果送入 PC；寄存器堆中两个寄存器的值送入 ALU，计算结果送入指定的寄存器



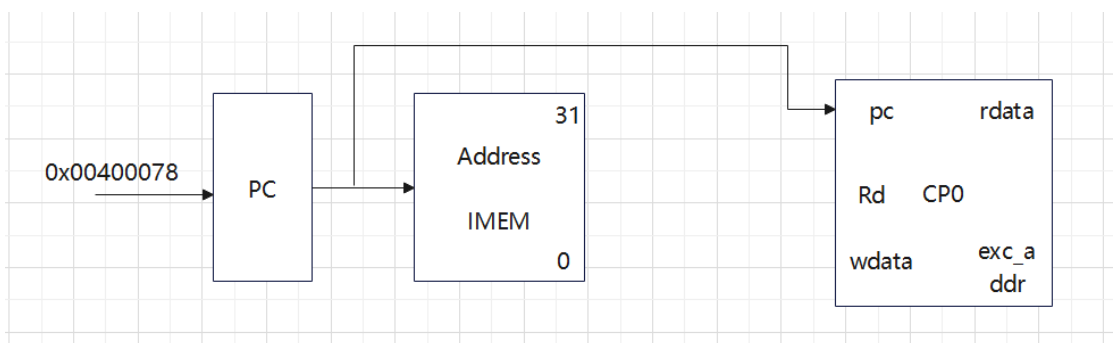
5. `and`: PC 的值送入指令存储器，PC+4 的结果送入 PC；寄存器堆中两个寄存器的值送入 ALU，计算结果送入指定的寄存器



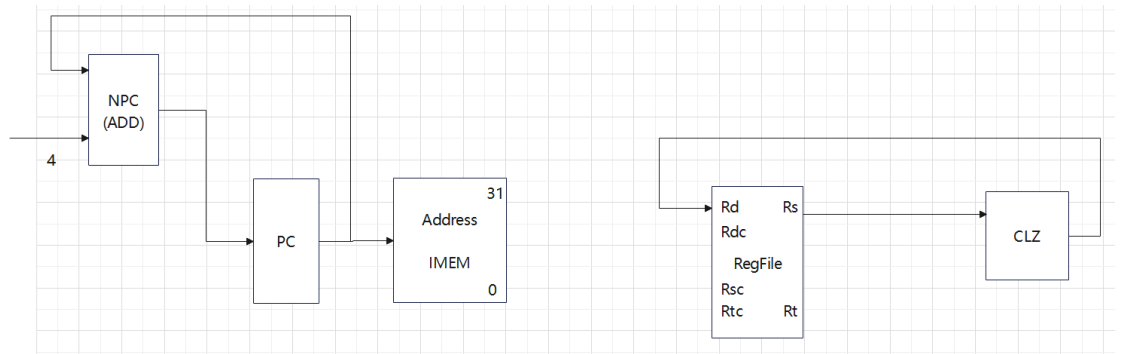
9. bne: PC 的值送到指令存储器，寄存器堆中两个寄存器的内容送到 ALU 运算，根据结果选择将 PC+4 还是指令的 15-0 位扩展结果送到 PC



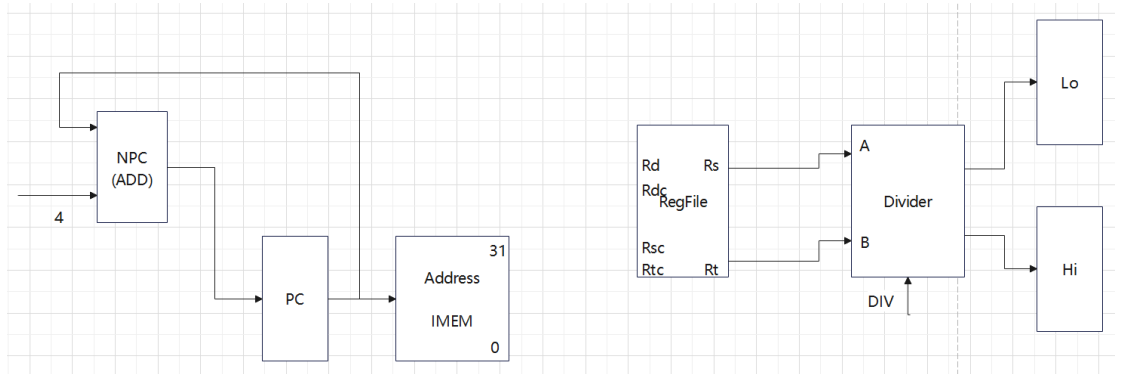
10. break: PC 送入数据存储器 and cp0 中存储 pc 值的寄存器，固定值送入 PC



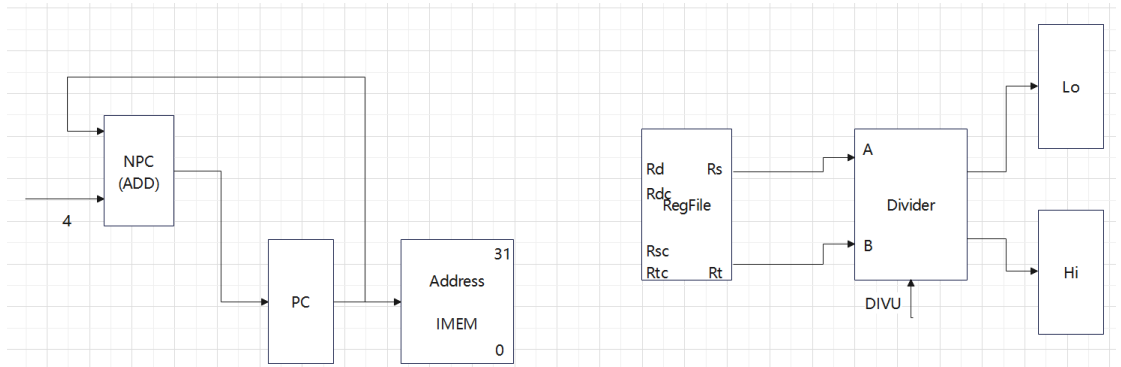
11. clz: PC 送入数据存储器，PC+4 的值送入 PC，寄存器堆中的一个寄存器的值送入到 CLZ 中，CLZ 的值送入到寄存器堆的指定寄存器



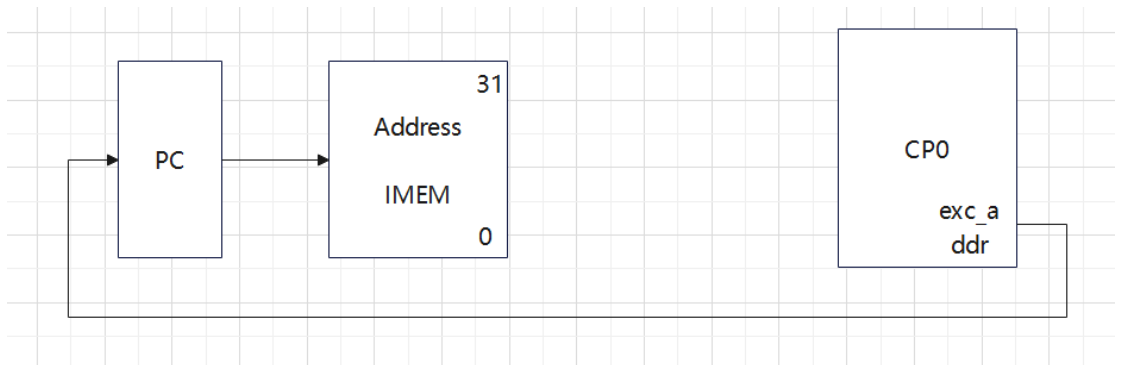
12. div: PC 送入数据存储器，PC+4 的值送入 PC，寄存器堆中的两个寄存器的值分别送入到 Divider 的两端，Divider 的结果分别送入 Lo 和 Hi 寄存器



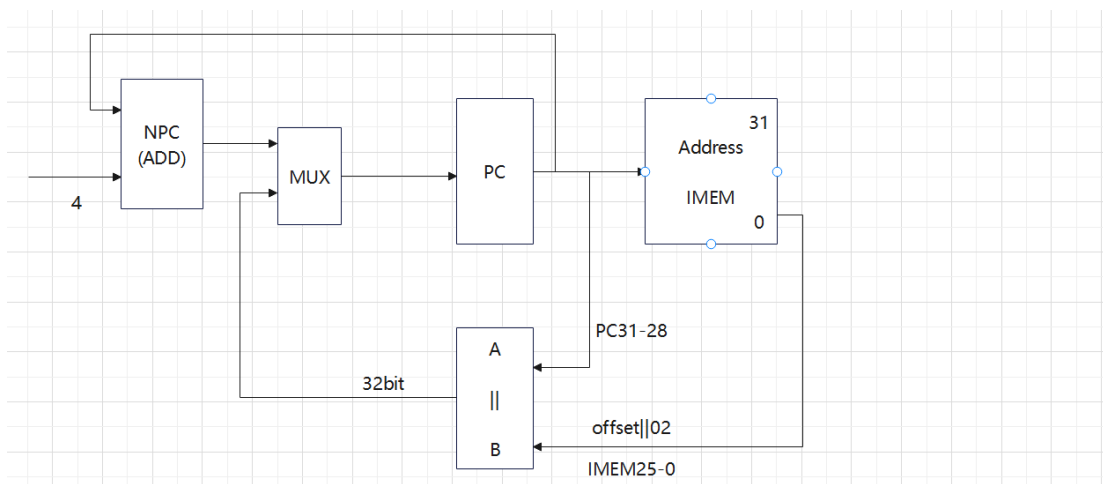
13. divu: PC 送入数据存储器，PC+4 的值送入 PC，寄存器堆中的两个寄存器的值分别送入到 Divider 的两端，Divider 的结果分别送入 Lo 和 Hi 寄存器



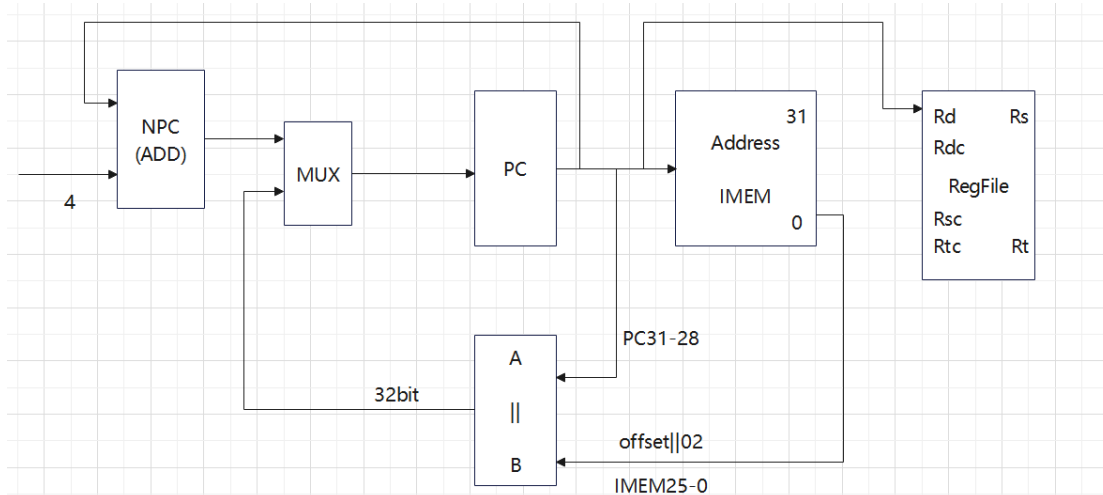
14. eret: PC 的值送入数据存储器中，cp0 中存储返回地址寄存器的值送入到 PC 中



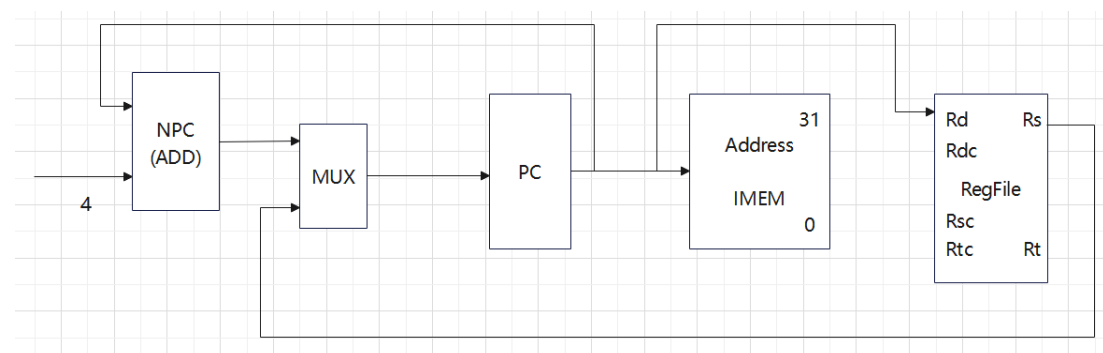
15. j:PC 的值送到指令存储器，指令存储器值的 25-0 位左移两位和 PC 高 4 位拼接送到 PC 寄存器



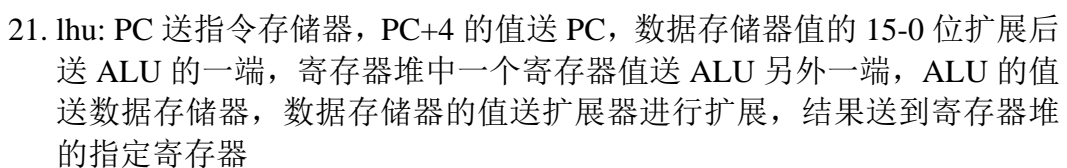
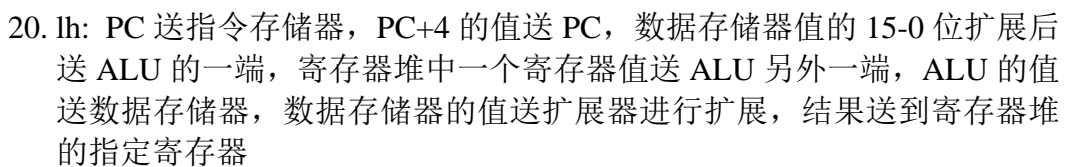
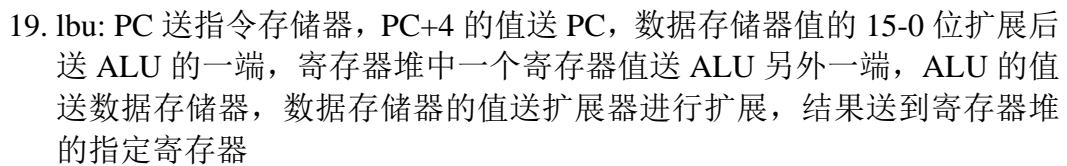
16. jal: PC 的值送到指令存储器和寄存器堆指定寄存器，指令存储器值的 25-0 位左移两位和 PC 高 4 位拼接送到 PC 寄存器

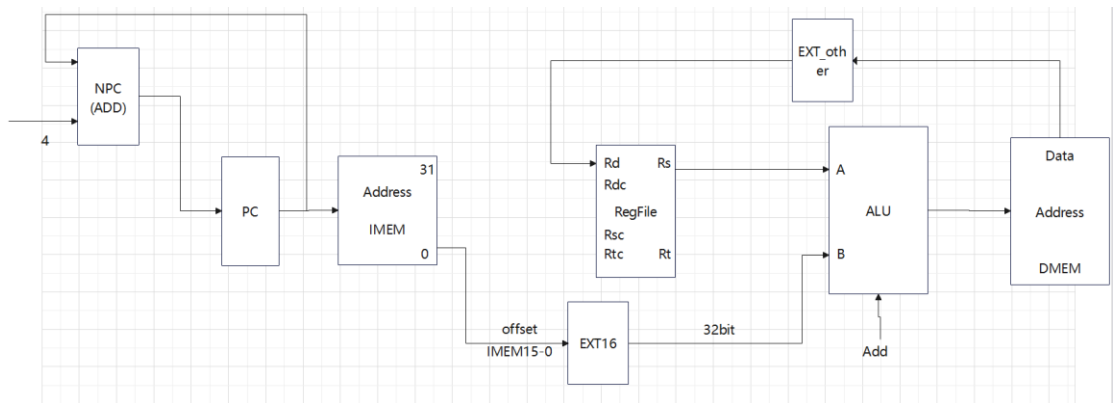


17. jalr: PC 送指令存储器和寄存器堆中指定寄存器，寄存器堆中一个寄存器的值送到 PC

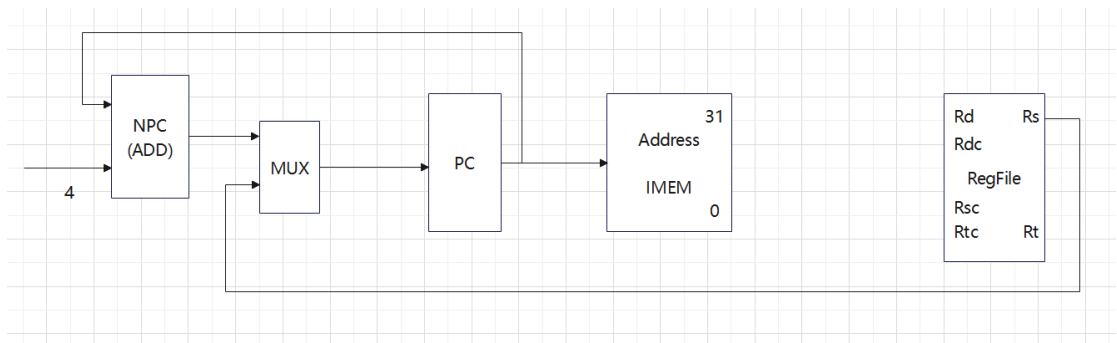


18. lb: PC 送指令存储器，PC+4 的值送 PC，数据存储器值的 15-0 位扩展后送 ALU 的一端，寄存器堆中一个寄存器值送 ALU 另外一端，ALU 的值送数据存储器，数据存储器的值送扩展器进行扩展，结果送到寄存器堆的指定寄存器

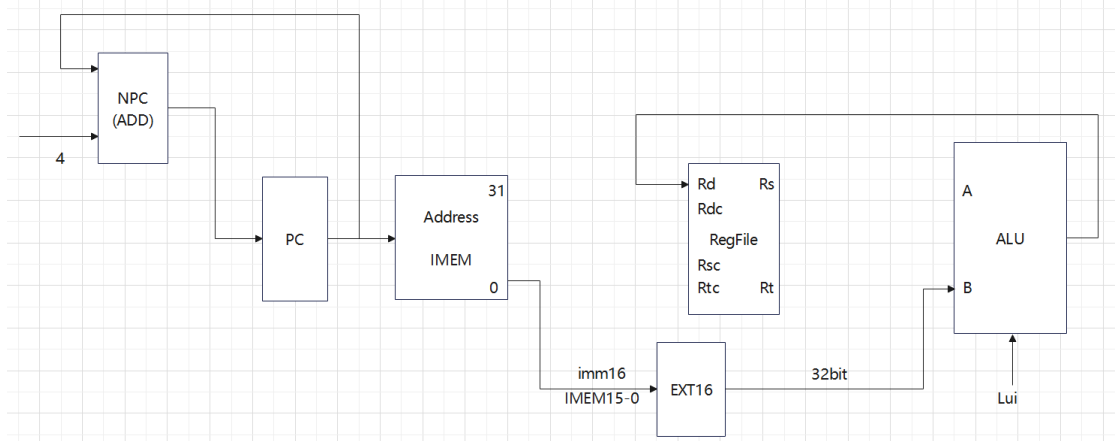




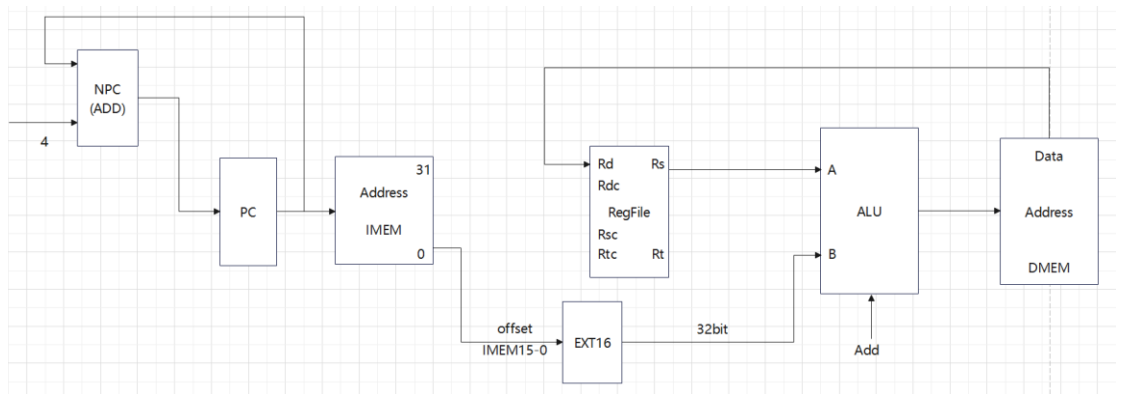
22. jr:PC 的值送到指令存储器，寄存器堆中指定的寄存器送到 PC



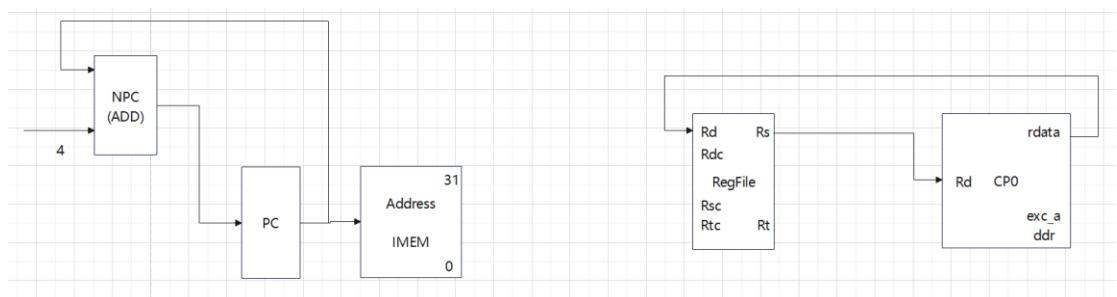
23. lui: PC 的值送入指令存储器，PC+4 的结果送入 PC；指令存储器选出值的 15 位到 0 位经位扩展送入 ALU 的一端，寄存器堆中的一个寄存器的值送入 ALU 的另一端，计算结果送入指定的寄存器



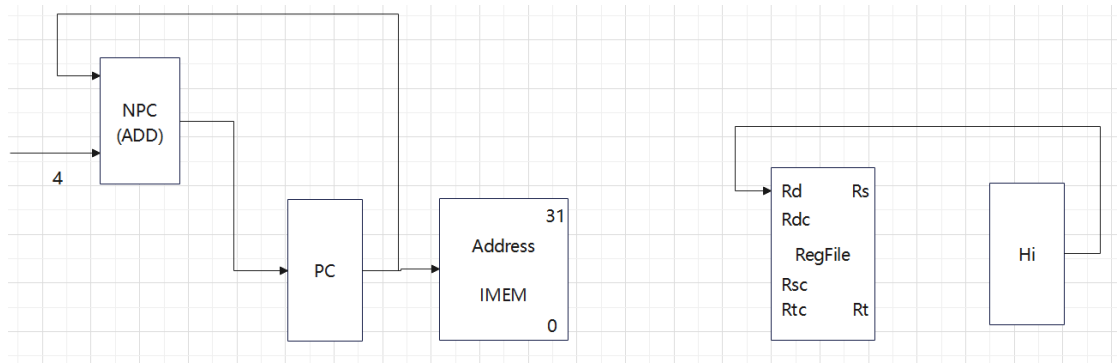
24. lw:PC 的值送指令存储器，PC+4 的值送 PC，指令存储器的值的 15-0 位扩展后送 ALU 一端，寄存器堆中一个寄存器的值送 ALU 另一端，ALU 结果送数据存储器，数据存储器的值送寄存器堆中的指定寄存器



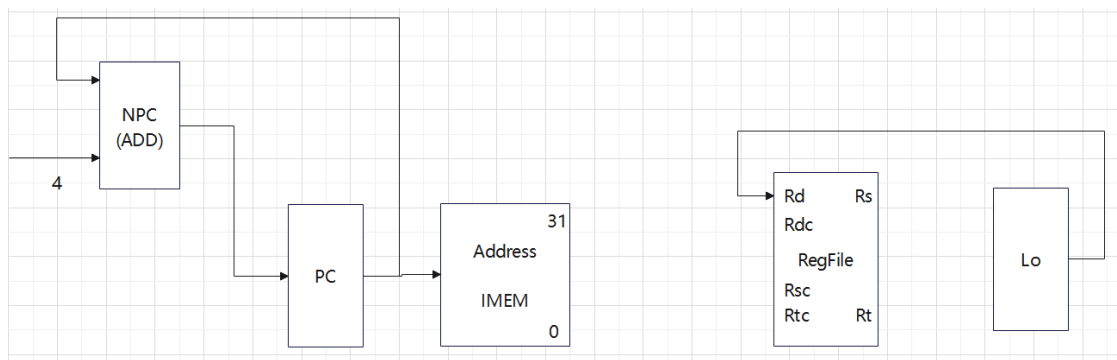
25. mfc0: PC 的值送指令寄存器, PC+4 的值送 PC, 寄存器堆中一个寄存器的值送 cp0, cp0 的数据值送寄存器堆中的指定寄存器



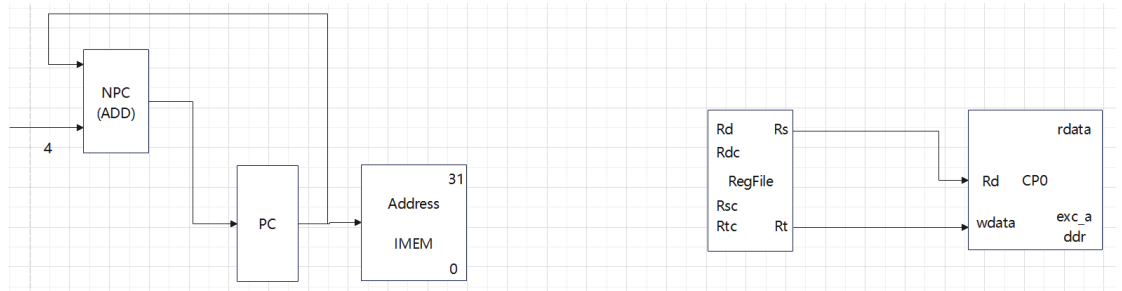
26. mfhi: PC 的值送指令寄存器, PC+4 的值送 PC, hi 寄存器值送寄存器堆中的指定寄存器



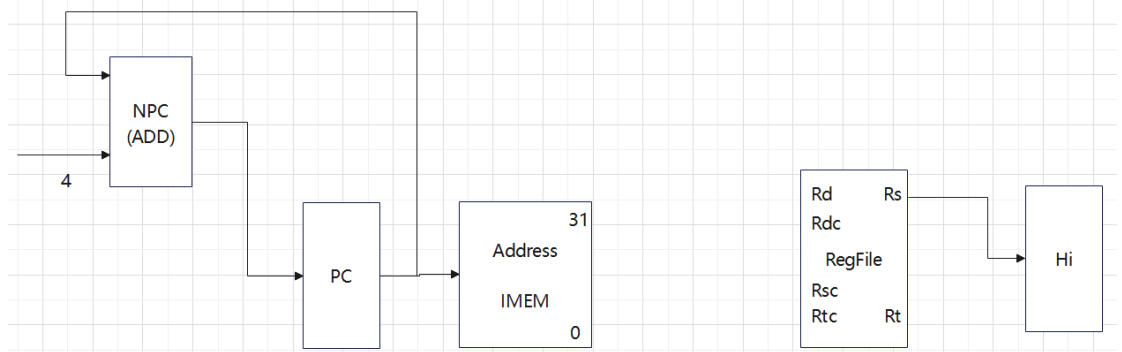
27. mflo: PC 的值送指令寄存器, PC+4 的值送 PC, lo 寄存器值送寄存器堆中的指定寄存器



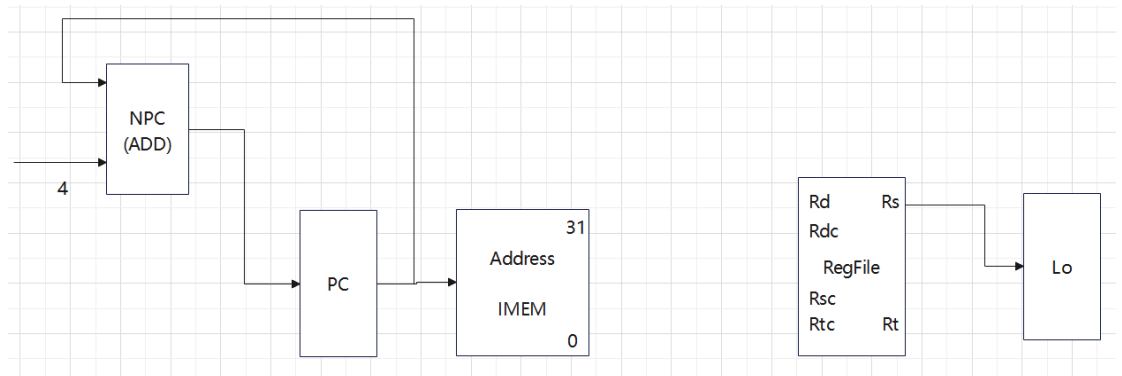
28. mtc0: PC 的值送指令寄存器, PC+4 的值送 PC, 寄存器堆中一个寄存器的值送 cp0 表地址, 另一个寄存器值送 cp0 表要存储的数据



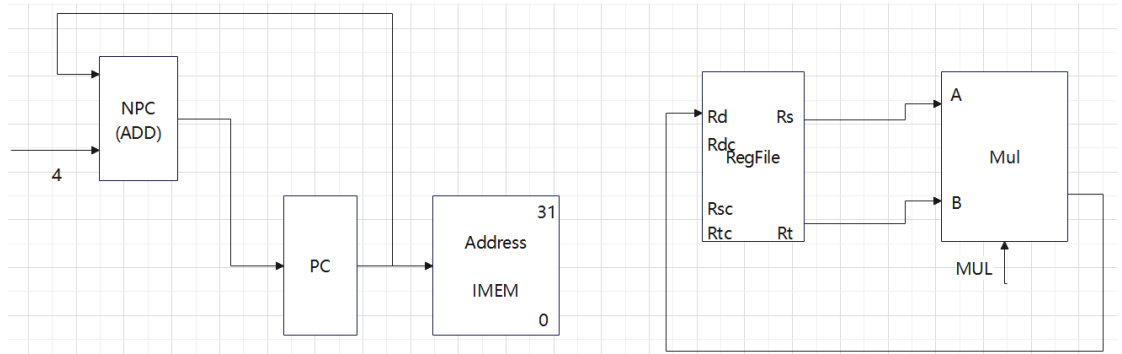
29. mthi: PC 的值送指令寄存器, PC+4 的值送 PC, 寄存器堆中的指定寄存器值送 hi 寄存器



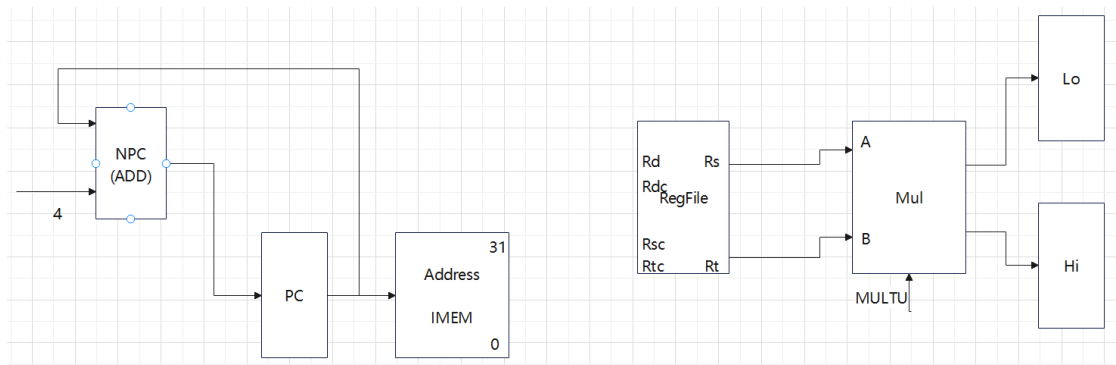
30. mtlo: PC 的值送指令寄存器, PC+4 的值送 PC, 寄存器堆中的指定寄存器值送 lo 寄存器



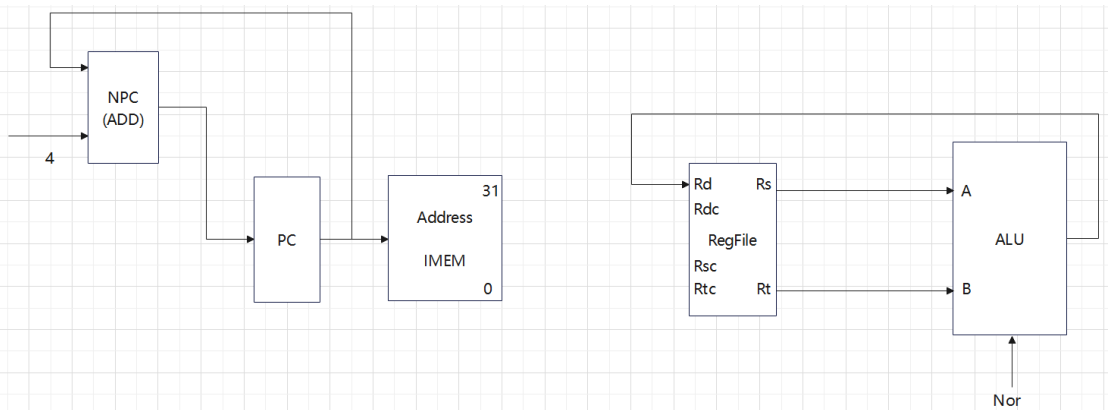
31. mul: PC 的值送指令寄存器, PC+4 的值送 PC, 寄存器堆中的两个寄存器值分别送乘法器两端, 乘法器结果送寄存器堆指定寄存器



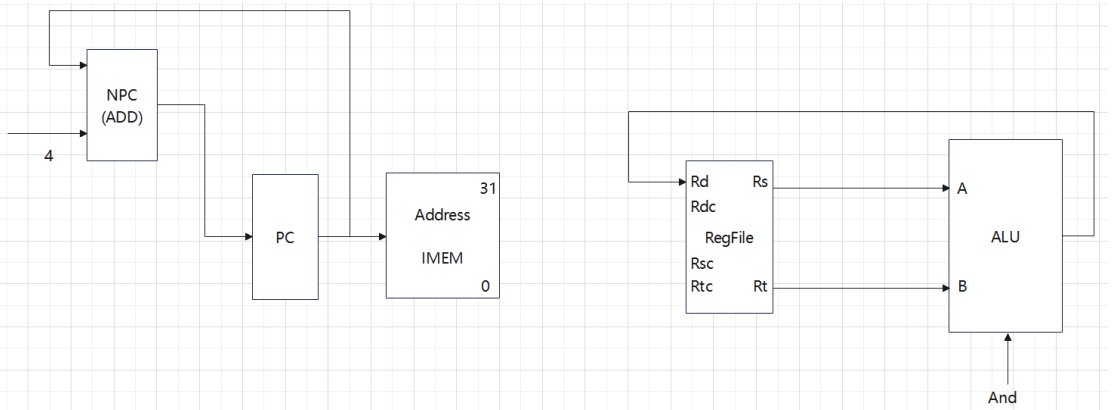
32. multu: PC 的值送指令寄存器, PC+4 的值送 PC, 寄存器堆中的两个寄存器值分别送乘法器两端, 乘法器结果分别送 lo 寄存器和 hi 寄存器



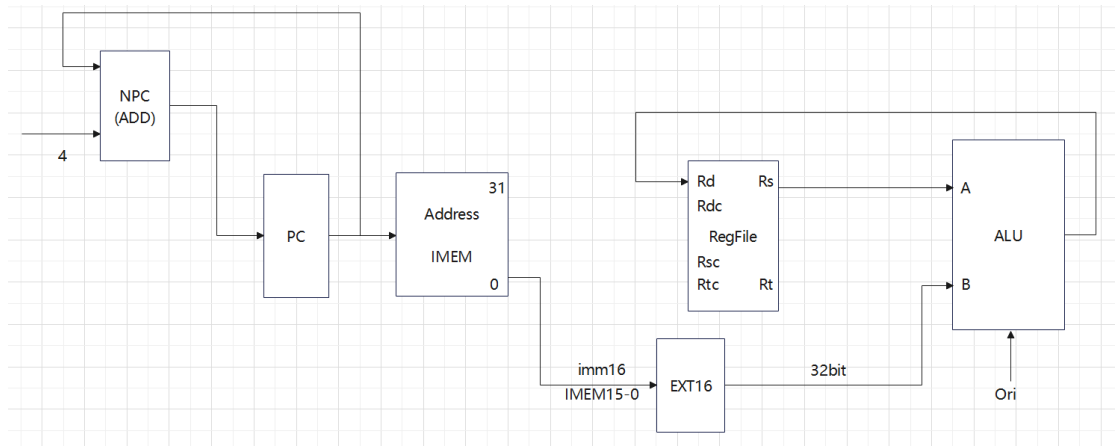
33. nor: PC 的值送入指令存储器，PC+4 的结果送入 PC；寄存器堆中两个寄存器的值送入 ALU，计算结果送入指定的寄存器



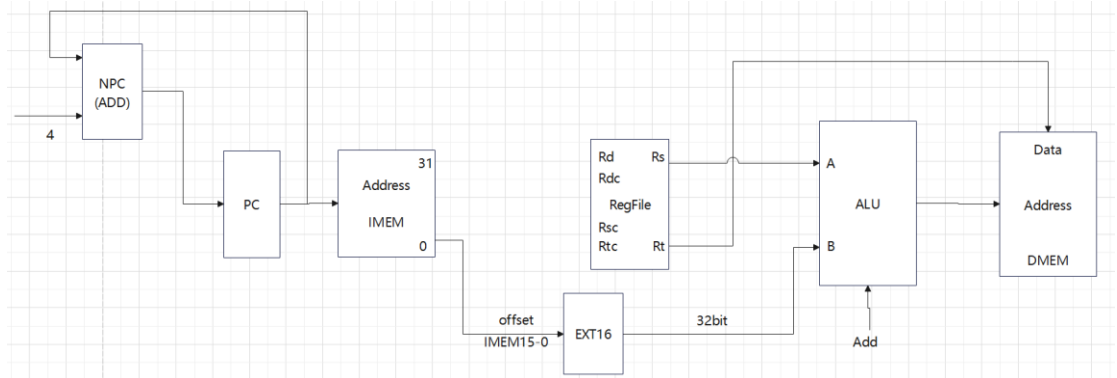
34. or: PC 的值送入指令存储器，PC+4 的结果送入 PC；寄存器堆中两个寄存器的值送入 ALU，计算结果送入指定的寄存器



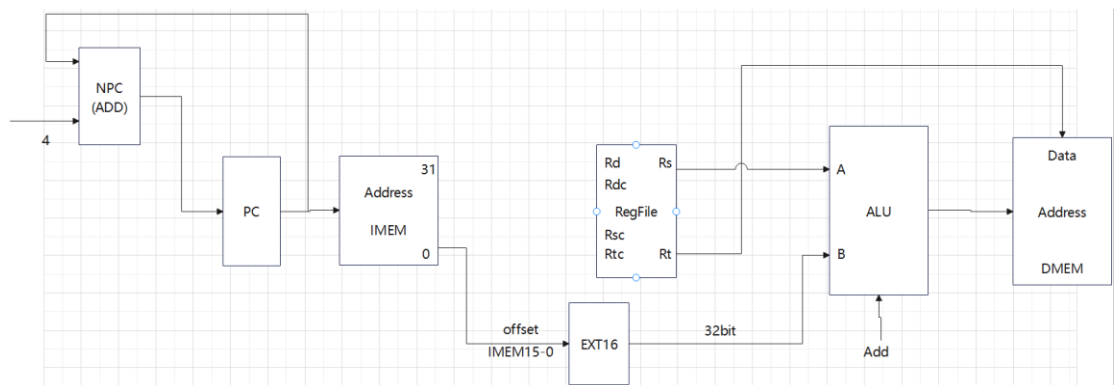
35. ori: PC 的值送入指令存储器，PC+4 的结果送入 PC；指令存储器选出值的 15 位到 0 位经位扩展送入 ALU 的一端，寄存器堆中的一个寄存器的值送入 ALU 的另一端，计算结果送入指定的寄存器



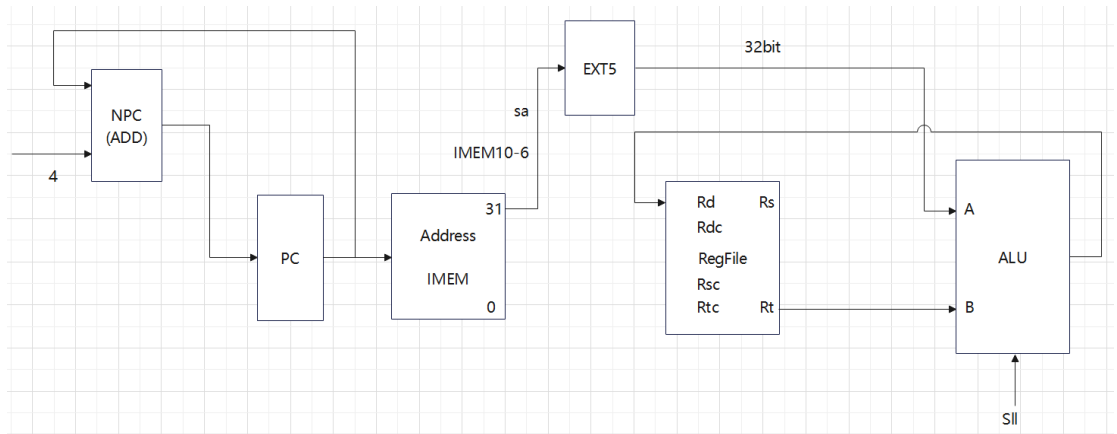
36. sb: PC 寄存器的值送到指令存储器，PC+4 的值送到 PC 寄存器，指令存储器值的 15-0 位扩展后送到 ALU 的一端，寄存器堆中一个寄存器的值送到 ALU 的另一端，ALU 的结果送到数据存储器，寄存器堆中一个寄存器的值送到数据存储器



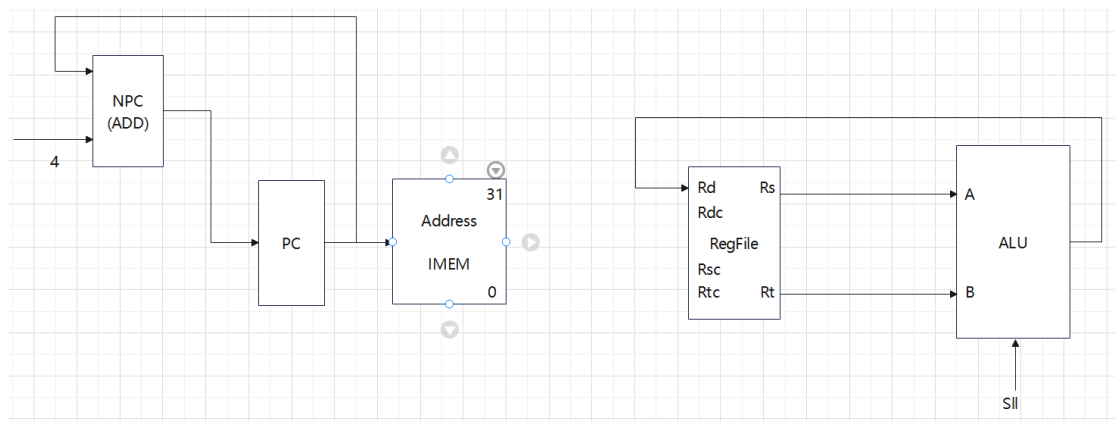
37. sh: PC 寄存器的值送到指令存储器，PC+4 的值送到 PC 寄存器，指令存储器值的 15-0 位扩展后送到 ALU 的一端，寄存器堆中一个寄存器的值送到 ALU 的另一端，ALU 的结果送到数据存储器，寄存器堆中一个寄存器的值送到数据存储器



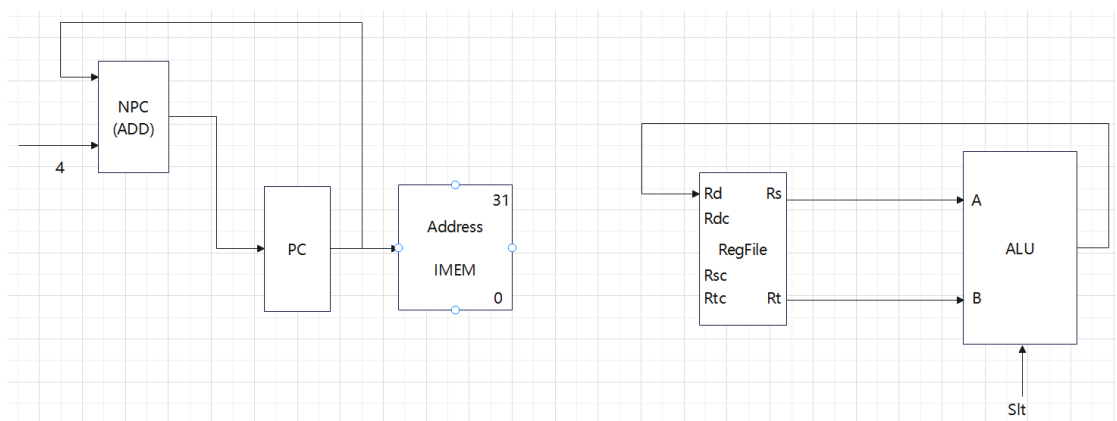
38. sll: PC 的值送指令存储器，PC+4 的值送 PC，指令存储器值的 10-6 位扩展后送 ALU 一端，寄存器堆中一个寄存器的值送 ALU 另一端，ALU 结果送寄存器堆中指定寄存器



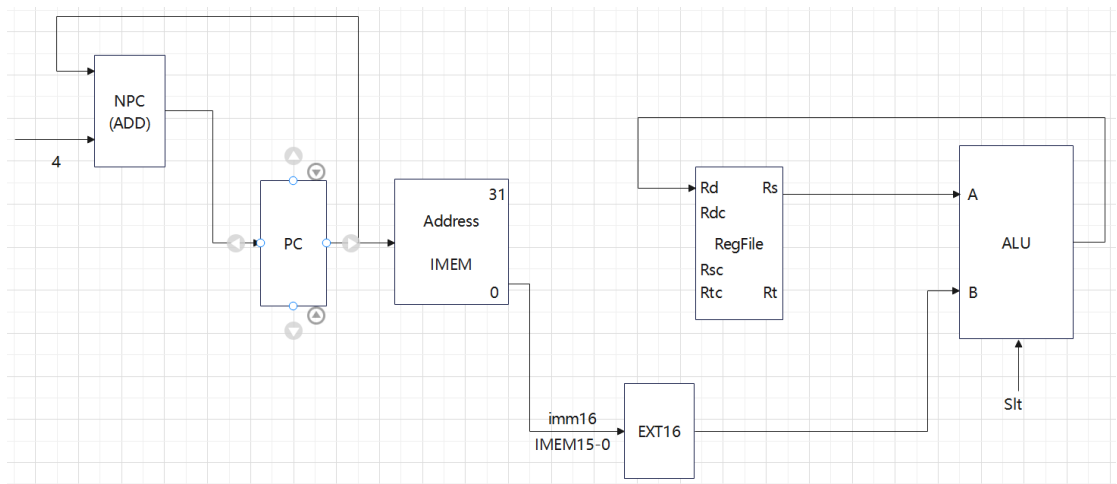
39. sllv: PC 的值送入指令存储器，PC+4 的结果送入 PC；寄存器堆中两个寄存器的值送入 ALU，计算结果送入指定的寄存器



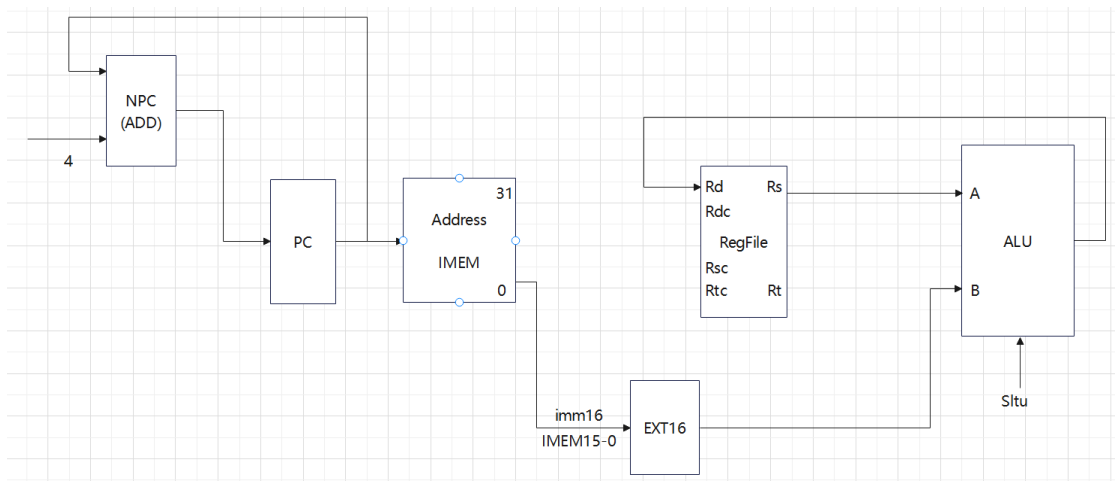
40. slt: PC 的值送入指令存储器，PC+4 的结果送入 PC；寄存器堆中两个寄存器的值送入 ALU，计算结果送入指定的寄存器



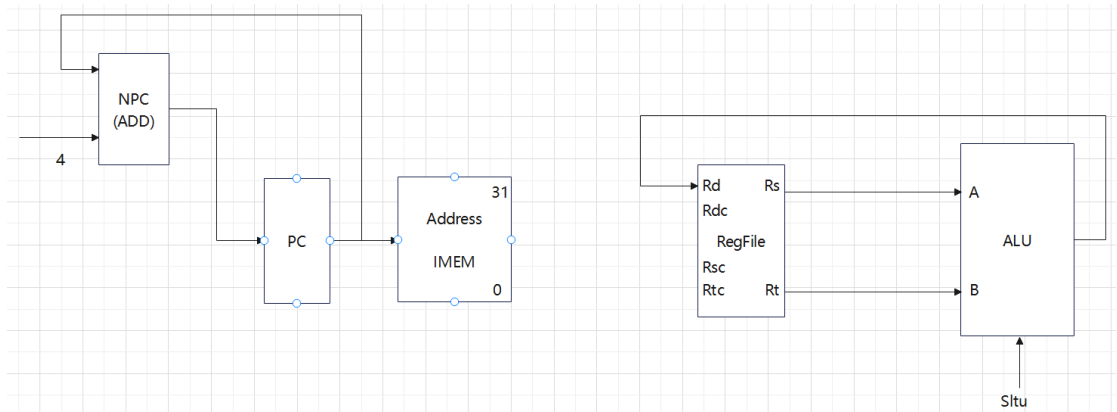
41. slti: PC 的值送入指令存储器，PC+4 的结果送入 PC；指令存储器选出值的 15 位到 0 位经位扩展送入 ALU 的一端，寄存器堆中的一个寄存器的值送入 ALU 的另一端，计算结果送入指定的寄存器



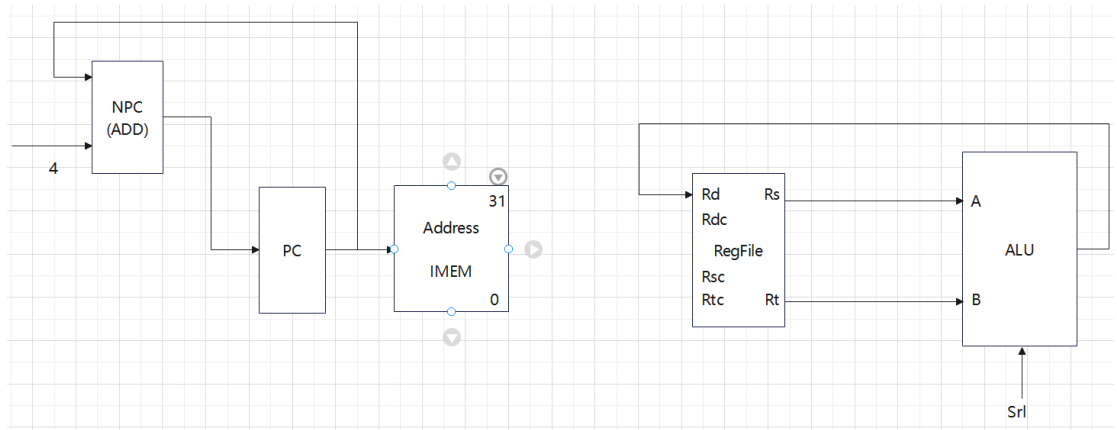
42. `sltiu`: PC 的值送入指令存储器，PC+4 的结果送入 PC；指令存储器选出值的 15 位到 0 位经位扩展送入 ALU 的一端，寄存器堆中的一个寄存器的值送入 ALU 的另一端，计算结果送入指定的寄存器



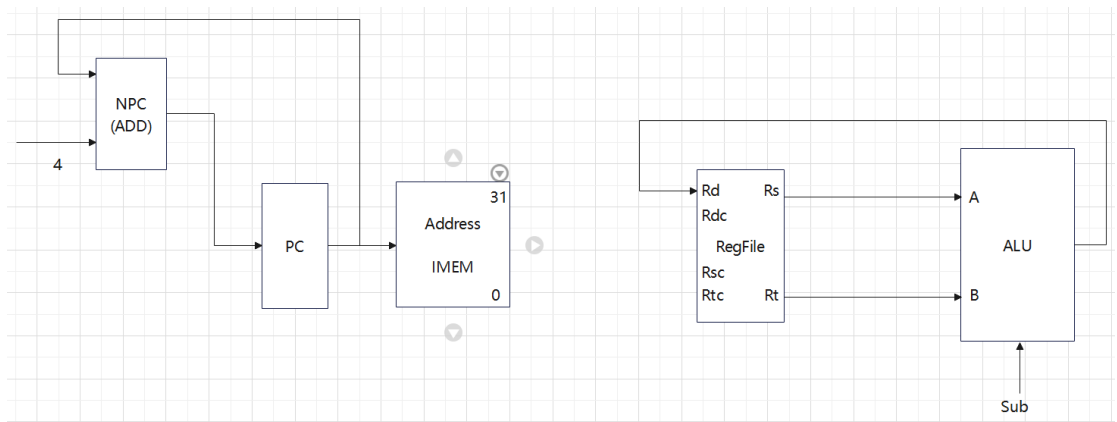
43. `sltu` PC 的值送入指令存储器，PC+4 的结果送入 PC；寄存器堆中两个寄存器的值送入 ALU，计算结果送入指定的寄存器



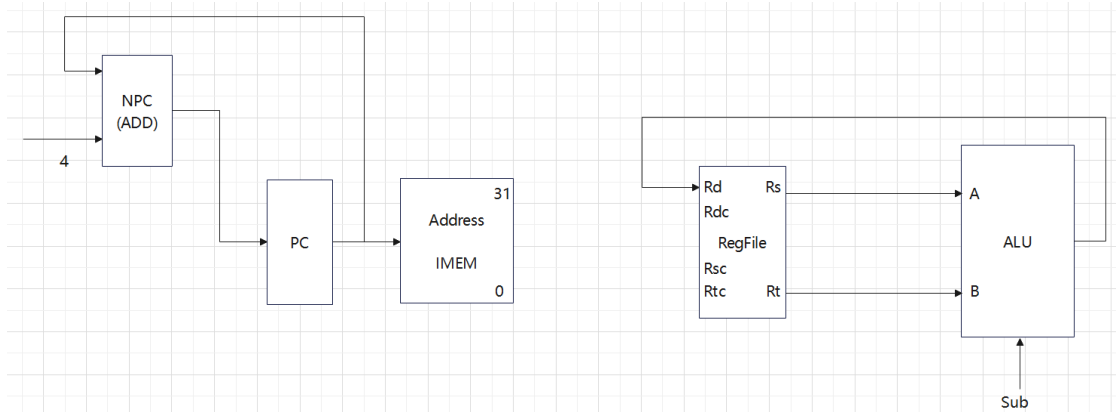
44. `sra`: PC 的值送指令存储器，PC+4 的值送 PC，指令存储器值的 10-6 位扩展后送 ALU 一端，寄存器堆中一个寄存器的值送 ALU 另一端，ALU 结果送寄存器堆中指定寄存器



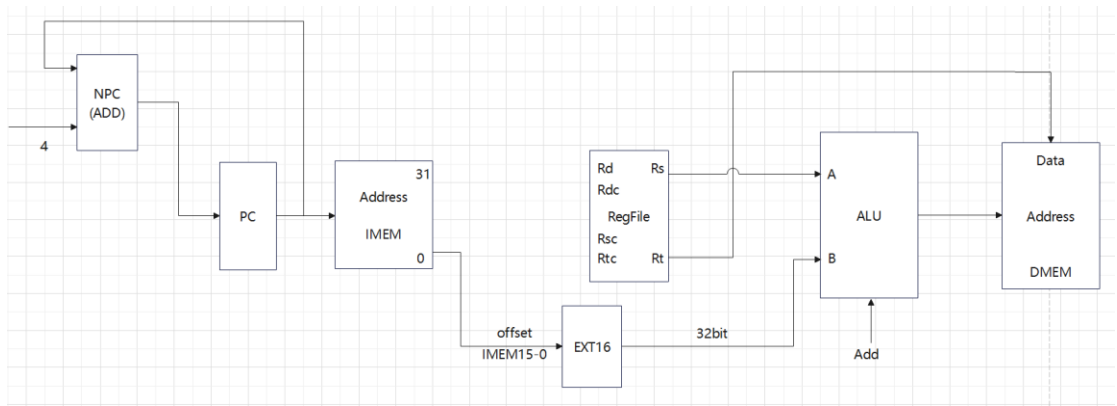
48. `sub`: PC 的值送入指令存储器，PC+4 的结果送入 PC；寄存器堆中两个寄存器的值送入 ALU，计算结果送入指定的寄存器



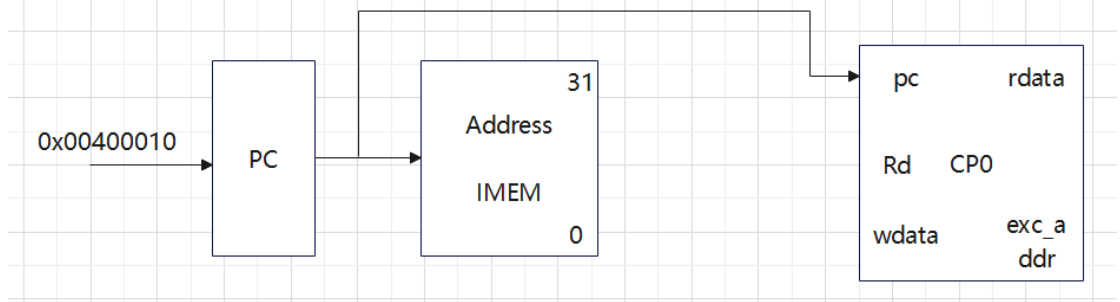
49. `subu`: PC 的值送入指令存储器，PC+4 的结果送入 PC；寄存器堆中两个寄存器的值送入 ALU，计算结果送入指定的寄存器



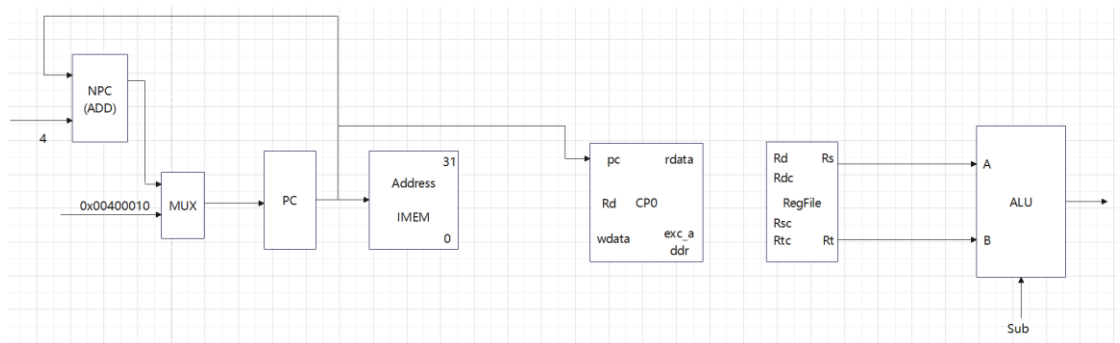
50. `sw`: PC 寄存器的值送到指令存储器，PC+4 的值送到 PC 寄存器，指令存储器值的 15-0 位扩展后送到 ALU 的一端，寄存器堆中一个寄存器的值送到 ALU 的另一端，ALU 的结果送到数据存储器，寄存器堆中一个寄存器的值送到数据存储器



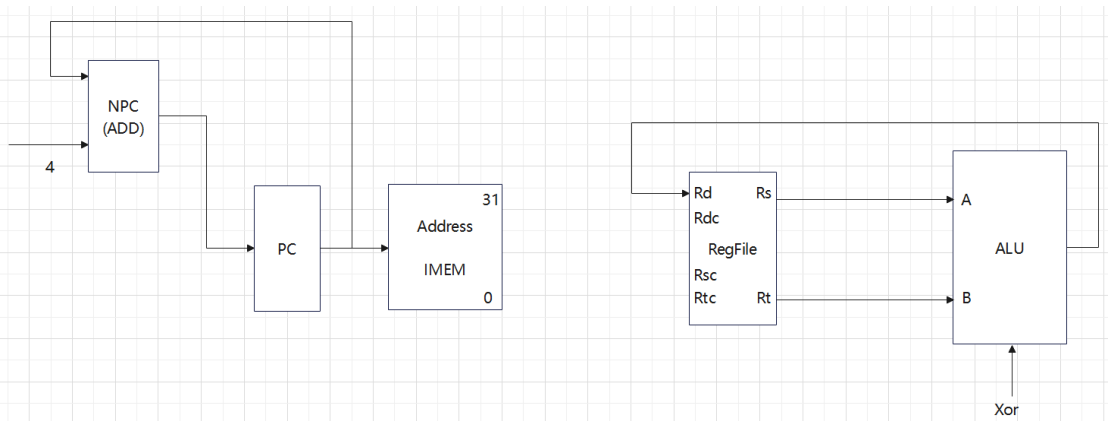
51. syscall: PC 值送指令存储器和 cp0 中中存储地址的寄存器，固定值送 PC



52. teq: PC 值送 cp0 中的存储地址的寄存器，寄存器队中的两个寄存器的值送到 ALU 两端，根据 ALU 的结果判断是 PC+4 还是一个固定值送入 PC

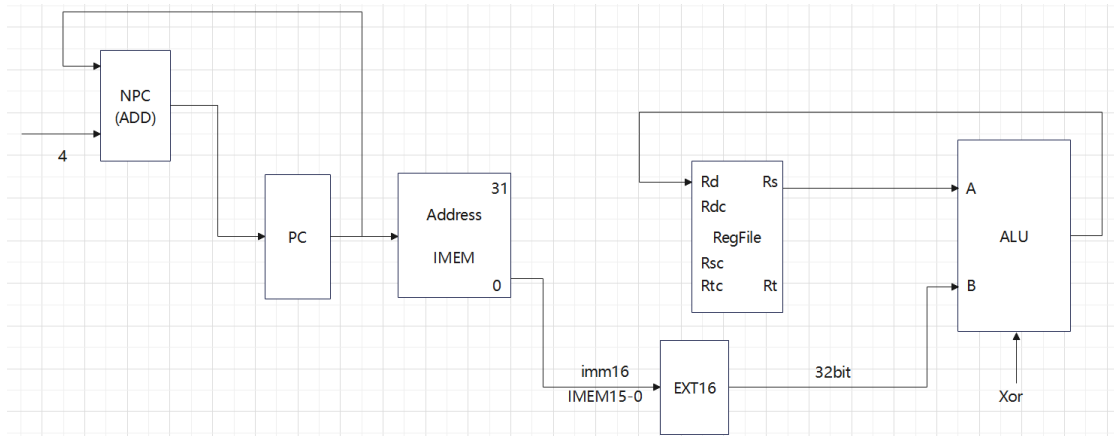


53. xor: PC 的值送入指令存储器，PC+4 的结果送入 PC；寄存器堆中两个寄存器的值送入 ALU，计算结果送入指定的寄存器



54. xori: PC 的值送入指令存储器，PC+4 的结果送入 PC；指令存储器选出值

的 15 位到 0 位经位扩展送入 ALU 的一端，寄存器堆中的一个寄存器的值送入 ALU 的另一端，计算结果送入指定的寄存器



四. 控制信号

	add	addu	sub	subu	and	or	xor	nor	sll	slltu	sll	srl	sra	sllv	srlv	sra	jr
pcreg_ena	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
d_ram_wena	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
d_ram_ena	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ext5_s	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ext16_s	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ext18_s	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
rf_wve	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
rf_wadd	imem15-11	imem15-11	imem15-11	imem15-11	imem15-11	imem15-11	imem15-11	imem15-11	imem15-11	imem15-11	imem15-11	imem15-11	imem15-11	imem15-11	imem15-11	imem15-11	imem15-11
rf_radd1	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21
rf_radd2	imem20-16	imem20-16	imem20-16	imem20-16	imem20-16	imem20-16	imem20-16	imem20-16	imem20-16	imem20-16	imem20-16	imem20-16	imem20-16	imem20-16	imem20-16	imem20-16	imem20-16
alu_aluc	4'b0010	4'b0000	4'b0011	4'b0001	4'b0100	4'b0101	4'b0110	4'b0111	4'b0111	4'b0101	4'b1110	4'b1101	4'b1100	4'b1110	4'b1101	4'b1100	0
M1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M2	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0
M3	2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b00
M4	2'b01	2'b01	2'b01	2'b01	2'b01	2'b01	2'b01	2'b01	2'b01	2'b01	2'b01	2'b01	2'b01	2'b01	2'b01	2'b01	0
M5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M6	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00
M7	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000
M8	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00
M9	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00
eo_instr	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00
cp0_instr	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000
cp0_ena	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
cp0_cause	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000
div_instr	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
mul_instr	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
hi_ena	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
lo_ena	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
d_ram_instr	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00
指令编号	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

addi	addiu	andi	ori	xori	lw	sw	beq	bne	sllti	slltiu	lui	j	jal	clz	divu
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	1	1	0	0	1	1	0	0	0	0	0
0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	0
imem20-16	imem20-16	imem20-16	imem20-16	imem20-16	imem20-16	imem20-16	0	0	imem20-16	imem20-16	imem20-16	0	5'b11111	imem15-11	0
imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	0	imem25-21	imem25-21	imem25-21
0	0	0	0	0	0	0	imem20-16	imem20-16	imem20-16	imem20-16	0	0	0	0	imem20-16
4'b0010	4'b0000	4'b0100	4'b0101	4'b0110	4'b0000	4'b0000	4'b0011	4'b0011	4'b0101	4'b1010	4'b1000	0	0	0	0
1	1	1	1	1	1	1	0	0	1	1	1	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b1[zero]	2'b1[~zero]	2'b10	2'b10	2'b10	2'b01	2'b01	2'b10	2'b10
2'b01	2'b01	2'b01	2'b01	2'b01	2'b00	0	0	0	2'b01	2'b01	2'b01	0	2'b10	2'b11	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00
3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000
2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00
2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00
3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31+	32+

eret	jahr	lb	lbu	lhu	sb	sh	lh	mfc0	mfhi	mflo	mtc0	mti	mtlo	mul	multu	syscall	teq	bgez
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	1	1	1	1	0	0	1	1	1	1	0	0	0	1	0	0	0	0
0	imem15-11	imem20-16	imem20-16	imem20-16	0	0	imem20-16	imem20-16	imem15-11	imem15-11	0	0	0	imem15-11	0	0	0	0
0	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem25-21	imem15-11	0	0	imem20-16	imem25-21	imem25-21	imem25-21	imem25-21	0	imem25-21	imem25
0	0	0	0	0	imem20-16	imem20-16	0	0	0	0	imem15-11	0	0	imem20-16	imem20-16	0	imem20-16	imem20
0	0	4'b0000	4'b0000	4'b0000	4'b0000	4'b0000	4'b0000	0	0	0	0	0	0	0	0	0	4'b0011	4'b0011
0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1
2'b00	2'b00	2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b10	2'b00	2'b10	2'b1(-n
0	2'b10	2'b00	2'b00	2'b00	0	0	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
2'b10	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b01	2'b0(zero)	2'b00
3'b000	3'b000	3'b001	3'b001	3'b001	3'b000	3'b000	3'b001	3'b010	3'b011	3'b100	3'b000	3'b000	3'b000	3'b101	3'b000	3'b000	3'b000	3'b000
2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b01	2'b00	2'b00	2'b10	2'b00	2'b00	2'b00
2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b01	2'b00	2'b10	2'b00	2'b00
2'b00	2'b00	2'b00	2'b01	2'b11	2'b00	2'b00	2'b10	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00
3'b001	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b100	3'b000	3'b000	3'b010	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000	3'b000
1	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0
5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b00000	5'b01000	5'b01101	5'b0000
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
2'b00	2'b00	2'b00	2'b00	2'b00	2'b01	2'b10	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00	2'b00
33*	34*	35*	36*	37*	38*	39*	40*	41*	42*	43*	44*	45*	46*	47*	48*	49*	50*	51*

break	div
1	1
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	imem25-21
0	imem20-16
0	0
0	0
0	0
0	0
2'b00	2'b10
2'b00	0
0	0
2'b01	2'b00
3'b000	3'b000
2'b00	2'b00
2'b00	2'b00
2'b00	2'b00
3'b000	3'b000
1	0
5'b01001	5'b00000
0	0
0	0
0	1
0	1
2'b00	2'b00
52*	53*

五. 所用到的部件建模

1. PC 寄存器:

```

module pcreg(
    input clk,
    input rst,
    input ena,
    input [31 : 0] data_in,
    output reg [31 : 0] data_out
);
initial
begin
    data_out = 32'h00400000;
end
always @(negedge clk)
begin
    if (rst == 1)
        begin
            data_out <= 32'h00400000;
        end
    else
        begin
            if (ena == 1)
                begin
                    data_out <= data_in;
                end
            end
        end
    endmodule

```

2. lo 和 hi 寄存器:

```

module single_reg2(
    input clk,
    input rst,
    input ena,
    input [31 : 0] data_in,
    output reg [31 : 0] data_out

```

```

);
initial
begin
    data_out = 32'h00400000;
end
always @ (posedge clk)
begin
    if (rst == 1)
    begin
        data_out <= 32'h00400000;
    end
end

```

```

end
else
begin
    if (ena == 1)
    begin
        data_out <= data_in;
    end
end
end
endmodule

```

3. 行为级 ALU:

```

module alu(
    input [31 : 0] a,
    input [31 : 0] b,
    input [3 : 0] aluc,
    output reg [31 : 0] r,
    output reg zero,
    output reg carry,
    output reg negative,
    output reg overflow
);
reg [31 : 0] results [13 : 0];
reg zeros [13 : 0];
reg carrys [13 : 0];
reg negatives [13 : 0];
reg overflows [13 : 0];
always @ (*)
begin
    //无符号加法
    results[0] = a + b;
    if (results[0] == 32'h00000000)
    begin
        zeros[0] = 1;
    end
    else
    begin
        zeros[0] = 0;
    end
    if ($unsigned(results[0]) <
$unsigned(a) || $unsigned(results[0]) <
$unsigned(b))
    begin
        carrys[0] = 1;
    end
end

```

```

else
begin
    carrys[0] = 0;
end
if (results[0][31] == 1)
begin
    negatives[0] = 1;
end
else
begin
    negatives[0] = 0;
end
//有符号加法
results[1] = a + b;
if (results[1] == 32'h00000000)
begin
    zeros[1] = 1;
end
else
begin
    zeros[1] = 0;
end
if (results[1][31] == 1)
begin
    negatives[1] = 1;
end
else
begin
    negatives[1] = 0;
end
if (a[31] == 1 && b[31] == 1
&& results[1][31] == 0)
begin

```

```

        overflows[1] = 1;
    end
    else if (a[31] == 0 && b[31] ==
0 && results[1][31] == 1)
        begin
            overflows[1] = 1;
        end
    else
        begin
            overflows[1] = 0;
        end
    //无符号减法
    results[2] = a - b;
    if (results[2] == 32'h00000000)
        begin
            zeros[2] = 1;
        end
    else
        begin
            zeros[2] = 0;
        end
    if ($unsigned(a) <
$unsigned(b))
        begin
            carries[2] = 1;
        end
    else
        begin
            carries[2] = 0;
        end
    if (results[2][31] == 1)
        begin
            negatives[2] = 1;
        end
    else
        begin
            negatives[2] = 0;
        end
    //有符号减法
    results[3] = a - b;
    if (results[3] == 32'h00000000)
        begin
            zeros[3] = 1;
        end

```

```

    else
        begin
            zeros[3] = 0;
        end
    if (results[3][31] == 1)
        begin
            negatives[3] = 1;
        end
    else
        begin
            negatives[3] = 0;
        end
    if (a[31] == 0 && b[31] == 1
&& results[3][31] == 1)
        begin
            overflows[3] = 1;
        end
    else if (a[31] == 1 && b[31] ==
0 && results[3][31] == 0)
        begin
            overflows[3] = 1;
        end
    else
        begin
            overflows[3] = 0;
        end
    //与运算
    results[4] = a & b;
    if (results[4] == 32'h00000000)
        begin
            zeros[4] = 1;
        end
    else
        begin
            zeros[4] = 0;
        end
    if (results[4][31] == 1)
        begin
            negatives[4] = 1;
        end
    else
        begin
            negatives[4] = 0;
        end

```

```

//或运算
results[5] = a | b;
if (results[5] == 32'h00000000)
begin
    zeros[5] = 1;
end
else
begin
    zeros[5] = 0;
end
if (results[5][31] == 1)
begin
    negatives[5] = 1;
end
else
begin
    negatives[5] = 0;
end
//异或运算
results[6] = a ^ b;
if (results[6] == 32'h00000000)
begin
    zeros[6] = 1;
end
else
begin
    zeros[6] = 0;
end
if (results[6][31] == 1)
begin
    negatives[6] = 1;
end
else
begin
    negatives[6] = 0;
end
//或非运算
results[7] = ~(a | b);
if (results[7] == 32'h00000000)
begin
    zeros[7] = 1;
end
else
begin

```

```

    zeros[7] = 0;
end
if (results[7][31] == 1)
begin
    negatives[7] = 1;
end
else
begin
    negatives[7] = 0;
end
//lui 运算
results[8] = {b[15 : 0], 16'b0};
if (results[8] == 32'h00000000)
begin
    zeros[8] = 1;
end
else
begin
    zeros[8] = 0;
end
carrys[8] = b[16];
if (results[8][31] == 1)
begin
    negatives[8] = 1;
end
else
begin
    negatives[8] = 0;
end
//有符号比较
results[9] = ($signed(a) <
$signed(b)) ? 1 : 0;
if (results[9] == 32'h00000000)
begin
    zeros[9] = 1;
end
else
begin
    zeros[9] = 0;
end
negatives[9] = 0;
//无符号比较
if ($unsigned(a) <
$unsigned(b))

```

```

begin
    results[10] = 1;
end
else
begin
    results[10] = 0;
end
if (results[10] ==
32'h00000000)
begin
    zeros[10] = 1;
end
else
begin
    zeros[10] = 0;
end
negatives[10] = 0;
//算术右移
results[11] = $signed(b) >>> a;
if (results[11] == 0)
begin
    zeros[11] = 1;
end
else
begin
    zeros[11] = 0;
end
if (a != 0)
begin
    if ($signed(a) <= 32)
    begin
        carries[11] = b[a - 1];
    end
    else
    begin
        carries[11] = b[31];
    end
end
if (results[11][31] == 1)
begin
    negatives[11] = 1;
end
else
begin
    negatives[11] = 0;
end
if (a != 0)
begin
    negatives[11] = 0;
end
//逻辑左移/算术左移
results[12] = b << a;
if (results[12] ==
32'h00000000)
begin
    zeros[12] = 1;
end
else
begin
    zeros[12] = 0;
end
if (a != 0)
begin
    if ($signed(a) <= 32)
    begin
        carries[12] = b[32 - a];
    end
    else
    begin
        carries[12] = 0;
    end
end
if (results[12][31] == 1)
begin
    negatives[12] = 1;
end
else
begin
    negatives[12] = 0;
end
//逻辑右移
results[13] = b >> a;
if (results[13] ==
32'h00000000)
begin
    zeros[13] = 1;
end
else
begin
    zeros[13] = 0;
end
if (a != 0)

```



```

begin
    if ($signed(a) <= 32)
        begin
            carrys[13] = b[a - 1];
        end
    else
        begin
            carrys[13] = 0;
        end
    end
    if (results[13][31] == 1)
    begin
        negatives[13] = 1;
    end
    else
    begin
        negatives[13] = 0;
    end
    if (aluc == 4'b0000)//无符号加
    begin
        r = results[0];
        zero = zeros[0];
        carry = carrys[0];
        negative = negatives[0];
        overflow = overflows[0];
    end
    else if (aluc == 4'b0010)//有符号加
    begin
        r = results[1];
        zero = zeros[1];
        carry = carrys[1];
        negative = negatives[1];
        overflow = overflows[1];
    end
    else if (aluc == 4'b0001)//无符号减
    begin
        r = results[2];
        zero = zeros[2];
        carry = carrys[2];
        negative = negatives[2];
        overflow = overflows[2];
    end

```

```

    else if (aluc == 4'b0011)//有符号减
    begin
        r = results[3];
        zero = zeros[3];
        carry = carrys[3];
        negative = negatives[3];
        overflow = overflows[3];
    end
    else if (aluc == 4'b0100)//and
    begin
        r = results[4];
        zero = zeros[4];
        carry = carrys[4];
        negative = negatives[4];
        overflow = overflows[4];
    end
    else if (aluc == 4'b0101)//or
    begin
        r = results[5];
        zero = zeros[5];
        carry = carrys[5];
        negative = negatives[5];
        overflow = overflows[5];
    end
    else if (aluc == 4'b0110)//xor
    begin
        r = results[6];
        zero = zeros[6];
        carry = carrys[6];
        negative = negatives[6];
        overflow = overflows[6];
    end
    else if (aluc == 4'b0111)//nor
    begin
        r = results[7];
        zero = zeros[7];
        carry = carrys[7];
        negative = negatives[7];
        overflow = overflows[7];
    end
    else if (aluc == 4'b1000 || aluc == 4'b1001)//lui
    begin

```

```

        r = results[8];
        zero = zeros[8];
        carry = carrys[8];
        negative = negatives[8];
        overflow = overflows[8];
    end
    else if (aluc == 4'b1011)//slt
    begin
        r = results[9];
        zero = zeros[9];
        carry = carrys[9];
        negative = negatives[9];
        overflow = overflows[9];
    end
    else if (aluc == 4'b1010)//sltu
    begin
        r = results[10];
        zero = zeros[10];
        carry = carrys[10];
        negative = negatives[10];
        overflow = overflows[10];
    end
    else if (aluc == 4'b1100)//sra
    begin
        r = results[11];

```

```

        zero = zeros[11];
        carry = carrys[11];
        negative = negatives[11];
        overflow = overflows[11];
    end
    else if (aluc == 4'b1110 || aluc
    == 4'b1111)//sll
    begin
        r = results[12];
        zero = zeros[12];
        carry = carrys[12];
        negative = negatives[12];
        overflow = overflows[12];
    end
    else//srl
    begin
        r = results[13];
        zero = zeros[13];
        carry = carrys[13];
        negative = negatives[13];
        overflow = overflows[13];
    end
    end
endmodule

```

4. 寄存器堆:

```

module regfile(
    input clk,
    input rst,
    input we,
    input [4:0] raddr1,
    input [4:0] raddr2,
    input [4:0] waddr,
    input [31:0] wdata,
    output [31:0] rdata1,
    output [31:0] rdata2
);
    reg [31 : 0] array_reg [31 : 0];
    assign rdata1 = array_reg[raddr1];
    assign rdata2 = array_reg[raddr2];
    initial
    begin
        array_reg[0] = 0;
        array_reg[1] = 0;

```

```

        array_reg[2] = 0;
        array_reg[3] = 0;
        array_reg[4] = 0;
        array_reg[5] = 0;
        array_reg[6] = 0;
        array_reg[7] = 0;
        array_reg[8] = 0;
        array_reg[9] = 0;
        array_reg[10] = 0;
        array_reg[11] = 0;
        array_reg[12] = 0;
        array_reg[13] = 0;
        array_reg[14] = 0;
        array_reg[15] = 0;
        array_reg[16] = 0;
        array_reg[17] = 0;
        array_reg[18] = 0;
        array_reg[19] = 0;

```

```

        array_reg[20] = 0;
        array_reg[21] = 0;
        array_reg[22] = 0;
        array_reg[23] = 0;
        array_reg[24] = 0;
        array_reg[25] = 0;
        array_reg[26] = 0;
        array_reg[27] = 0;
        array_reg[28] = 0;
        array_reg[29] = 0;
        array_reg[30] = 0;
        array_reg[31] = 0;
    end
    always @ (posedge clk)
    begin
        if (rst == 1)
            begin
                array_reg[0] = 0;
                array_reg[1] = 0;
                array_reg[2] = 0;
                array_reg[3] = 0;
                array_reg[4] = 0;
                array_reg[5] = 0;
                array_reg[6] = 0;
                array_reg[7] = 0;
                array_reg[8] = 0;
                array_reg[9] = 0;
                array_reg[10] = 0;
                array_reg[11] = 0;
                array_reg[12] = 0;
                array_reg[13] = 0;
                array_reg[14] = 0;
                array_reg[15] = 0;
                array_reg[16] = 0;
                array_reg[17] = 0;
                array_reg[18] = 0;
                array_reg[19] = 0;
                array_reg[20] = 0;
                array_reg[21] = 0;
                array_reg[22] = 0;
                array_reg[23] = 0;
                array_reg[24] = 0;
                array_reg[25] = 0;
                array_reg[26] = 0;
                array_reg[27] = 0;
                array_reg[28] = 0;
                array_reg[29] = 0;
                array_reg[30] = 0;
                array_reg[31] = 0;
            end
        else
            begin
                if (we == 1 && waddr !=
0)
                    begin
                        array_reg[waddr] =
wdata;
                    end
            end
        end
    end
endmodule

```

5. 乘法器

```

module mul(
    input [31 : 0] a,
    input [31 : 0] b,
    input instr,
    output reg [31 : 0] lo,
    output reg [31 : 0] hi
);
    reg [63 : 0] mid;
    always @ (*)
    begin
        if (instr)
            begin
                mid = a * b;
            end
        else
            begin
                mid = $signed(a) *
$signed(b);
            end
        hi = mid[63 : 32];
        lo = mid[31 : 0];
    end
end

```

```
endmodule
```

6. 除法器

```
module divider(  
    input [31 : 0] a,  
    input [31 : 0] b,  
    input instr,  
    output reg [31 : 0] lo,  
    output reg [31 : 0] hi  
);  
always @ (*)  
begin  
    if (instr)  
        begin  
            lo = a / b;
```

```
            hi = a % b;  
        end  
    else  
        begin  
            lo = $signed(a) /  
$signed(b);  
            hi = $signed(a) %  
$signed(b);  
        end  
    end  
endmodule
```

7. CLZ

```
module clz(  
    input [31 : 0] a,  
    output reg [31 : 0] r  
);  
integer i;  
reg [31 : 0] temp;  
reg found;  
always @ (*)  
begin  
    temp = 32;  
    found = 0;  
    for (i = 31; i >= 0 && found ==
```

```
0; i = i - 1)  
    begin  
        if (a[i] == 1)  
            begin  
                temp = 31 - i;  
                found = 1;  
            end  
        end  
        r = temp;  
    end  
endmodule
```

8. 将几位数拼接为位数更多的部件:

```
module concatenator(  
    input [25 : 0] offset,  
    input [3 : 0] pc,  
    output [31 : 0] r  
);  
assign r = { {pc}, {offset}, {2'b00} };  
endmodule
```

9. 加法器:

```
module add(  
    input [31 : 0] a,  
    input [31 : 0] b,  
    output [31 : 0] r  
);  
assign r = a + b;  
endmodule
```

10. 位数扩展器:

```

module ext #(parameter WIDTH = 16)(
    input [WIDTH - 1 : 0] a,
    input sext,
    output [31 : 0] b
);
    assign b = {sext ? {32 - WIDTH{a[WIDTH - 1]}} : {32 - WIDTH{1'b0}}, a[WIDTH -
1 : 0]};
endmodule

```

11. 特殊扩展器

```

module ext_other(
    input [31 : 0] data_in,
    input [1 : 0] instr,
    output reg [31 : 0] data_out
);
    always @ (*)
    begin
        if (instr == 0)
            begin
                data_out =
                {{24{data_in[7]}}, data_in[7 : 0]};
            end
        else if (instr == 1)
            begin
                data_out = {{24{1'b0}}},
                data_in[15 : 0]};
            end
        else if (instr == 2)
            begin
                data_out = {{16{1'b0}},
                data_in[15 : 0]};
            end
    end
endmodule

```

12. 二选一数据选择器:

```

module mux2(
    input [31 : 0] a,
    input [31 : 0] b,
    input s,
    output [31 : 0] r
);
    assign r = s ? b : a;
endmodule

```

13. 四选一数据选择器:

```

module mux4(
    input [31 : 0] a,
    input [31 : 0] b,
    input [31 : 0] c,
    input [31 : 0] d,
    input [1 : 0] s,
    output reg [31 : 0] r
);
    always @ (*)
    begin
        if (s == 0)
            begin
                r = a;
            end
        else if (s == 1)
            begin
                r = b;
            end
        else if (s == 2)
            begin
                r = c;
            end
        else if (s == 3)
            begin
                r = d;
            end
    end
endmodule

```



```

begin
    if (ram_ena == 1 && wena == 1)
    begin
        if (instr == 0)
        begin
            data[addr] = data_in;
        end
        else if (instr == 1)
        begin
            data[addr][7 : 0] = data_in[7 : 0];
        end
        else if (instr == 2)
        begin
            data[addr][15 : 0] = data_in[15 : 0];
        end
    end
end
endmodule

```

六. 控制模块建模以及 CPU 建模

1. 控制模块建模:

```

module controller(
    input [31 : 0] instruction,
    input zero,
    input negative,
    output reg pcreg_ena,
    output reg d_ram_wena,
    output reg d_ram_ena,
    output reg ext5_s,
    output reg ext16_s,
    output reg ext18_s,
    output reg rf_we,
    output reg [4 : 0] rf_waddr,
    output reg [4 : 0] rf_raddr1,
    output reg [4 : 0] rf_raddr2,
    output reg [3 : 0] aluc,
    output reg M1,
    output reg M2,
    output [1 : 0] M3,
    output reg [1 : 0] M4,
    output reg M5,
    output [1 : 0] M6,
    output reg [2 : 0] M7,

```

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

```
else if (instruction[31 : 26] == 6'b001110)
begin
    op = 54'b0000000000000000000000000000000000000000000000000000000;
    aluc = 4'b0110;
end
else if (instruction[31 : 26] == 6'b100011)
begin
    op = 54'b0000000000000000000000000000000000000000000000000000000;
    aluc = 4'b0000;
end
else if (instruction[31 : 26] == 6'b101011)
begin
    op = 54'b0000000000000000000000000000000000000000000000000000000;
    aluc = 4'b0000;
end
else if (instruction[31 : 26] == 6'b000100)
begin
    op = 54'b0000000000000000000000000000000000000000000000000000000;
    aluc = 4'b0011;
end
else if (instruction[31 : 26] == 6'b000101)
begin
    op = 54'b0000000000000000000000000000000000000000000000000000000;
    aluc = 4'b0011;
end
else if (instruction[31 : 26] == 6'b001010)
begin
    op = 54'b0000000000000000000000000000000000000000000000000000000;
    aluc = 4'b1011;
end
else if (instruction[31 : 26] == 6'b001011)
begin
    op = 54'b0000000000000000000000000000000000000000000000000000000;
    aluc = 4'b1010;
end
else if (instruction[31 : 26] == 6'b001111)
begin
    op = 54'b0000000000000000000000000000000000000000000000000000000;
    aluc = 4'b1000;
end
else if (instruction[31 : 26] == 6'b011100)
begin
    if (instruction[5 : 0] == 6'b100000)
        begin
```

```

        op
54'b0000000000000000000000001000000000000000000000000000000;
        aluc = 0;
    end
    else if (instruction[5 : 0] == 6'b000010)
    begin
        op
54'b0000000100000000000000000000000000000000000000000000000;
        aluc = 0;
    end
end
else if (instruction[31 : 26] == 6'b010000)
begin
    if (instruction[5 : 0] == 6'b011000)
    begin
        op
54'b0000000000000000000000001000000000000000000000000000000;
        aluc = 0;
    end
    else if (instruction[5 : 0] == 6'b000000)
    begin
        if (instruction[31 : 20] == 12'b010000000000)
        begin
            op
54'b0000000000000010000000000000000000000000000000000000000;
            aluc = 0;
        end
        else if (instruction[31 : 20] == 12'b010000001000)
        begin
            op
54'b0000000000100000000000000000000000000000000000000000000;
            aluc = 0;
        end
    end
end
end
else if (instruction[31 : 26] == 6'b100000)
begin
    op = 54'b0000000000000000000000001000000000000000000000000000000;
    aluc = 4'b0000;
end
else if (instruction[31 : 26] == 6'b100100)
begin
    op = 54'b0000000000000000000000001000000000000000000000000000000;
    aluc = 4'b0000;

```

```

end
else if (instruction[31 : 26] == 6'b100101)
begin
    op = 54'b00000000000000001000000000000000000000000000000;
    aluc = 4'b0000;
end
else if (instruction[31 : 26] == 6'b101000)
begin
    op = 54'b00000000000000001000000000000000000000000000000;
    aluc = 4'b0000;
end
else if (instruction[31 : 26] == 6'b101001)
begin
    op = 54'b00000000000000001000000000000000000000000000000;
    aluc = 4'b0000;
end
else if (instruction[31 : 26] == 6'b100001)
begin
    op = 54'b00000000000000001000000000000000000000000000000;
    aluc = 4'b0000;
end
else if (instruction[31 : 26] == 6'b000001)
begin
    op = 54'b00100000000000000000000000000000000000000000000;
    aluc = 4'b0011;
end
pcreg_ena = 1;//
d_ram_wena = op[23] || op[38] || op[39];//
d_ram_ena = op[22] || op[23] || op[35] || op[36] || op[37] || op[38] || op[39] ||
op[40];//
ext5_s = 0;//
ext16_s = op[17] || op[18] || op[22] || op[23] || op[26] || op[27] || op[35] || op[36] ||
op[37] || op[38] || op[39] || op[40];//
ext18_s = op[24] || op[25] || op[51];//
rf_we = ~(op[16] || op[23] || op[24] || op[25] || op[29] || op[32] || op[33] || op[38] ||
op[39] || op[44] || op[45] || op[46] || op[48] || op[49] || op[50] || op[51] || op[52] || op[53]);//
if (op[17] || op[18] || op[19] || op[20] || op[21] || op[22] || op[26] || op[27] || op[28] ||
op[35] || op[36] || op[37] || op[40] || op[41])
begin
    rf_waddr = instruction[20 : 16];
end
else if (op[16] || op[23] || op[24] || op[25] || op[29] || op[32] || op[33] || op[38] ||
op[39] || op[44] || op[45] || op[46] || op[48] || op[49] || op[50] || op[51] || op[52] || op[53])
begin

```

```

        rf_waddr = 0;
    end
    else if (op[30])
    begin
        rf_waddr = 5'b11111;
    end
    else
    begin
        rf_waddr = instruction[15 : 11];
    end//
    if (op[10] || op[11] || op[12] || op[28] || op[29] || op[30] || op[33] || op[42] || op[43] ||
op[49] || op[52])
    begin
        rf_raddr1 = 0;
    end
    else if(op[41])
    begin
        rf_raddr1 = instruction[15 : 11];
    end
    else if (op[44])
    begin
        rf_raddr1 = instruction[20 : 16];
    end
    else
    begin
        rf_raddr1 = instruction[25 : 21];
    end//
    if (op[16] || op[17] || op[18] || op[19] || op[20] || op[21] || op[22] || op[26] || op[27] ||
op[28] || op[29] || op[30] || op[31] || op[33] || op[34] || op[35] || op[36] || op[37] || op[40] ||
op[41] || op[42] || op[43] || op[45] || op[46] || op[49] || op[52])
    begin
        rf_raddr2 = 0;
    end
    else if (op[44])
    begin
        rf_raddr2 = instruction[15 : 11];
    end
    else
    begin
        rf_raddr2 = instruction[20 : 16];
    end//
    M1 = op[17] || op[18] || op[19] || op[20] || op[21] || op[22] || op[23] || op[26] || op[27]
|| op[28] || op[35] || op[36] || op[37] || op[38] || op[39] || op[40];//
    M2 = ~(op[10] || op[11] || op[12] || op[16] || op[28] || op[29] || op[30] || op[31] ||

```



```

op[32] || op[33] || op[34] || op[41] || op[42] || op[43] || op[44] || op[45] || op[46] || op[47] ||
op[48] || op[49] || op[52] || op[53]);//
    if (op[16] || op[22] || op[23] || op[24] || op[25] || op[29] || op[32] || op[33] || op[35] ||
op[36] || op[37] || op[38] || op[39] || op[40] || op[41] || op[42] || op[43] || op[44] || op[45] ||
op[46] || op[47] || op[48] || op[49] || op[50] || op[51] || op[52] || op[53])
        begin
            M4 = 2'b00;
        end
    else if (op[30] || op[34])
        begin
            M4 = 2'b10;
        end
    else if (op[31])
        begin
            M4 = 2'b11;
        end
    else
        begin
            M4 = 2'b01;
        end//
    M5 = op[51];//
    if (op[35] || op[36] || op[37] || op[40])
        begin
            M7 = 3'b001;
        end
    else if (op[41])
        begin
            M7 = 3'b010;
        end
    else if (op[42])
        begin
            M7 = 3'b011;
        end
    else if (op[43])
        begin
            M7 = 3'b100;
        end
    else if (op[47])
        begin
            M7 = 3'b101;
        end
    else
        begin
            M7 = 3'b000;
        end

```

```

end//
if (op[45])
begin
    M8 = 2'b01;
end
else if (op[48])
begin
    M8 = 2'b10;
end
else
begin
    M8 = 2'b00;
end//
if (op[46])
begin
    M9 = 2'b01;
end
else if (op[48])
begin
    M9 = 2'b10;
end
else
begin
    M9 = 2'b00;
end
if (op[36])
begin
    eo_instr = 2'b01;
end
else if (op[37])
begin
    eo_instr = 2'b11;
end
else if (op[40])
begin
    eo_instr = 2'b10;
end
else
begin
    eo_instr = 2'b00;
end//
if (op[33])
begin
    cp0_instr = 3'b001;

```

```

end
else if (op[41])
begin
    cp0_instr = 3'b100;
end
else if (op[44])
begin
    cp0_instr = 3'b010;
end
else
begin
    cp0_instr = 3'b000;
end//
cp0_ena = op[33] || op[41] || op[44] || op[49] || op[52];//
if (op[49])
begin
    cp0_cause = 5'b01000;
end
else if (op[50])
begin
    cp0_cause = 5'b01101;
end
else if (op[52])
begin
    cp0_cause = 5'b01001;
end
else
begin
    cp0_cause = 5'b00000;
end//
div_instr = op[32];//
mul_instr = op[48];//
hi_ena = op[32] || op[45] || op[48] || op[53];//
lo_ena = op[32] || op[46] || op[48] || op[53];//
if (op[38])
begin
    d_ram_instr = 2'b01;
end
else if (op[39])
begin
    d_ram_instr = 2'b10;
end
else
begin

```

```

        d_ram_instr = 2'b00;
    end//
    if (instruction == 32'hfffffff)
    begin
        pcreg_ena = 0;
        rf_we = 0;
    end
end
endmodule

```

2. **cpu** 建模

```

module cpu(
    input clk,
    input reset,
    input [31 : 0] inst,
    input [31 : 0] dmem_out,
    output [31 : 0] pc,
    output dmem_ena,
    output dmem_wena,
    output [31 : 0] dmem_addr,
    output [31 : 0] dmem_in,
    output [1 : 0] d_ram_instr
);
    wire [31 : 0] vpc;
    wire [31 : 0] valu;
    wire zero;
    wire carry;
    wire negative;
    wire overflow;
    wire [31 : 0] vreg1;
    wire [31 : 0] vreg2;

    wire [31 : 0] vconcat;
    wire [31 : 0] vaddj;
    wire [31 : 0] vnpc;
    wire [31 : 0] vpc_8;
    wire [31 : 0] vext18;
    wire [31 : 0] vext16;
    wire [31 : 0] vext5;
    wire [31 : 0] vclz;
    wire [31 : 0] vcp0_addr;
    wire [31 : 0] vcp0_sta;
    wire [31 : 0] vcp0_rdata;
    wire [31 : 0] vhi;
    wire [31 : 0] vlo;

```

```
wire [31 : 0] vmul1;  
wire [31 : 0] vmul2;  
wire [31 : 0] vdivider1;  
wire [31 : 0] vdivider2;  
wire [31 : 0] veo;  
wire [31 : 0] vmid;
```

```
wire [31 : 0] vmux_1;  
wire [31 : 0] vmux_2;  
wire [31 : 0] vmux_3;  
wire [31 : 0] vmux_4;  
wire [31 : 0] vmux_5;  
wire [31 : 0] vmux_6;  
wire [31 : 0] vmux_7;  
wire [31 : 0] vmux_8;  
wire [31 : 0] vmux_9;
```

```
wire pcreg_ena;  
wire ext5_s;  
wire ext16_s;  
wire ext18_s;  
wire reg_we;  
wire [4 : 0] reg_waddr;  
wire [4 : 0] reg_raddr1;  
wire [4 : 0] reg_raddr2;  
wire [3 : 0] aluc;  
wire M1;  
wire M2;  
wire [1 : 0] M3;  
wire [1 : 0] M4;  
wire M5;  
wire [1 : 0] M6;  
wire [2 : 0] M7;  
wire [1 : 0] M8;  
wire [1 : 0] M9;  
wire [1 : 0] eo_instr;  
wire [2 : 0] cp0_instr;  
wire cp0_ena;  
wire [4 : 0] cp0_cause;  
wire div_instr;  
wire mul_instr;  
wire hi_ena;  
wire lo_ena;
```

```

assign pc = vpc;
assign dmem_addr = valu;
assign dmem_in = vreg2;

controller cpu54controller_inst(inst, zero, negative, pcreg_ena, dmem_wena, dmem_ena,
ext5_s, ext16_s, ext18_s, reg_we, reg_waddr, reg_raddr1, reg_raddr2, aluc,
M1, M2, M3, M4, M5, M6, M7, M8, M9,
eo_instr, cp0_instr, cp0_ena, cp0_cause, div_instr, mul_instr, hi_ena, lo_ena, d_ram_instr);

single_reg pcreg_inst(clk, reset, pcreg_ena, vmux_6, vpc);
single_reg2 mid(clk, reset, 1, vreg1, vmid);
single_reg2 hi(clk, reset, hi_ena, vmux_8, vhi);
single_reg2 lo(clk, reset, lo_ena, vmux_9, vlo);

alu alu_inst(vmux_2, vmux_5, aluc, valu, zero, carry, negative, overflow);
clz clz_inst(vreg1, vclz);
divider divider_inst(vreg1, vreg2, div_instr, vdivider2, vdivider1);
mul mul_inst(vreg1, vreg2, mul_instr, vmul2, vmul1);

regfile cpu_ref(clk, reset, reg_we, reg_raddr1, reg_raddr2, reg_waddr, vmux_7, vreg1,
vreg2);
cp0 cp0_inst(clk, rst, cp0_ena, cp0_instr[2], cp0_instr[1], cp0_instr[0], vnpc, vreg1[4 : 0],
cp0_cause, vreg2, vcp0_rdata, vcp0_sta, vcp0_addr);

concatenator concat(inst[25 : 0], vpc[31 : 28], vconcat);
add addj(vnpc, vext18, vaddj);
add npc(vpc, 4, vnpc);
add pc_8(vpc, 4, vpc_8);
ext #(.WIDTH(5)) ext5(inst[10 : 6], ext5_s, vext5);
ext #(.WIDTH(16)) ext16(inst[15 : 0], ext16_s, vext16);
ext #(.WIDTH(18)) ext18({ inst[15 : 0] }, { 2'b00 }, ext18_s, vext18);
ext_other ext_other_inst(dmem_out, eo_instr, veo);

mux2 mux_1(vreg2, vext16, M1, vmux_1);
mux2 mux_2(vext5, vreg1, M2, vmux_2);
mux4 mux_3(vmid, vconcat, vnpc, vaddj, M3, vmux_3);
mux4 mux_4(dmem_out, valu, vpc_8, vclz, M4, vmux_4);
mux2 mux_5(vmux_1, 0, M5, vmux_5);
mux4 mux_6(vmux_3, 32'h00400004, vcp0_addr, 0, M6, vmux_6);
mux8 mux_7(vmux_4, veo, vcp0_rdata, vhi, vlo, vmul2, 0, 0, M7, vmux_7);
mux4 mux_8(vdivider1, vreg1, vmul1, 0, M8, vmux_8);
mux4 mux_9(vdivider2, vreg1, vmul2, 0, M9, vmux_9);
endmodule

```

七. 顶层模块设计

```
module sccomp_dataflow(
    input clk_in,
    input reset,
    output [31:0] inst,
    output [31:0] pc
);
    wire [31 : 0] dmem_out;
    wire dmem_ena;
    wire dmem_wena;
    wire [31 : 0] dmem_addr;
    wire [31 : 0] dmem_in;
    wire [1 : 0] d_ram_instr;
    wire [31 : 0] instruction;
    reg [25:0] cnt;

    assign inst = instruction;

    cpu sccpu(clk_in, reset, instruction, dmem_out, pc, dmem_ena, dmem_wena,
dmem_addr, dmem_in, d_ram_instr);
    imem im(pc[12 : 2], instruction);
    ram dmem(clk_in, dmem_ena, dmem_wena, d_ram_instr, dmem_addr[12 : 0], dmem_in,
dmem_out);
endmodule
```

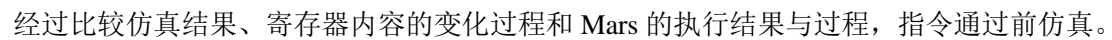
八. 前仿真测试

我在控制器中添加了当指令读取到 ffffffff 时关闭 PC 寄存器和寄存器堆的写入信号,并在 coe 文件的最后添加了一条 ffffffff 指令,因此 PC 寄存器和寄存器堆里的值能维持在最后一条指令执行后的状态,只需观察波形的最后即可判断前仿真结果(我也将每个周期的结果都打印出来进行对比,但是不便于放置在报告中,所以就只能展示几条代表性指令的波形)

测试文件如下:


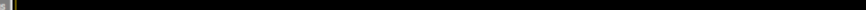




```
module top_tb();
    reg clk_in;
    reg reset;
    wire [31:0] inst;
    wire [31:0] pc;
    initial
    begin
        clk_in = 0;
        reset = 0;
    end
    always
```

使用 `mips_54_mars_simulate_student_ForWeb.coe` 测试的前仿真结果如下:



后仿真测试文件如下:

```
module after_tb();
    reg clk_in;
    reg reset;
    wire [31:0] inst;
    wire [31:0] pc;
    initial
    begin
        clk_in = 0;
        reset = 0;
    end
    always
        #50 clk_in = ~clk_in;
    sscomp_dataflow sscomp_dataflow_inst(clk_in, reset, inst, pc);
endmodule
```

Wave - User01		Magi
 /trap_thick_in	1to0	
 /trap_thinreset	32h00390004	
 /trap_thin	32h00400020	
 /trap_thick	32h00400000	
 /gbl_GSR	1to0	

由于下板需要在七段数码管显示结果，为了便于观察，我对 **cpu** 进行了分频，并添加了顶层模块，如下：

```
module cpu54top(
    input clk_in,
    input reset,
    output [7 : 0] o_seg,
    output [7 : 0] o_sel
);
    wire [31 : 0] inst;
```



```
wire [31 : 0] pc;  
reg [22 : 0] cnt;  
always @ (posedge clk_in, posedge reset)  
if (reset)  
    cnt <= 0;  
else  
    cnt <= cnt + 1'b1;  
wire clk_cpu = cnt[22];  
scomp_dataflow scomp_dataflow_inst(clk_cpu, reset, inst, pc);  
seg7x16(clk_in, reset, 1, pc, o_seg, o_sel);  
endmodule
```

下板结果如图所示：

