同济大学计算机系

计算机组成原理实验报告

学　　号 2050259

姓　　名 何征昊

专　　业 计算机科学与技术

授课老师 张冬冬

一、实验内容

本实验制作 54 指令的多周期 cpu，对于实验的指令有充分掌握，同时对于异常处理需要自己写 cp0 协处理器完成对于异常的处理。

二、硬件逻辑图

54 条操作硬件通路设计：

1）Addi

周期数：4

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 读出 rs 寄存器的内容以及指令低 16 位送拓展器；

c) 由 ALU 完成计算；

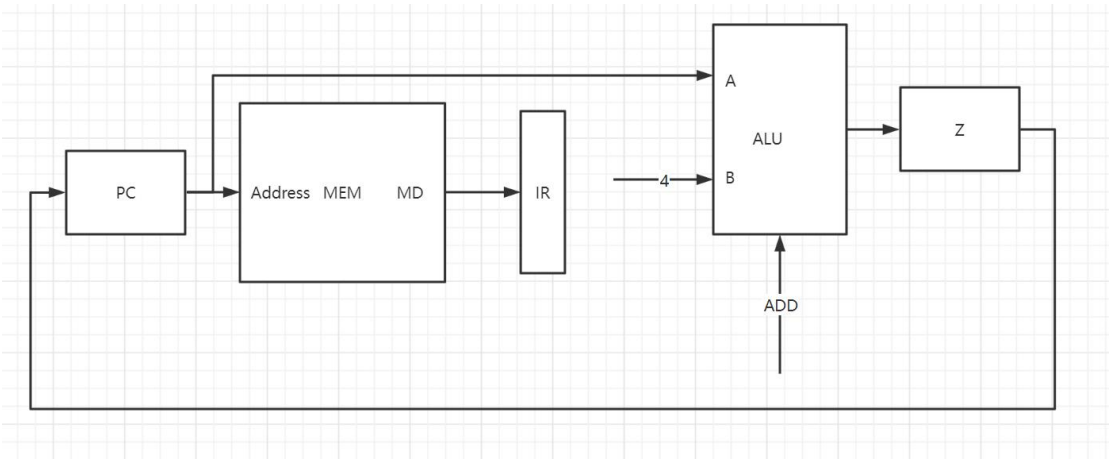
d) 把计算结果写入寄存器堆中的寄存器。

输入来源：

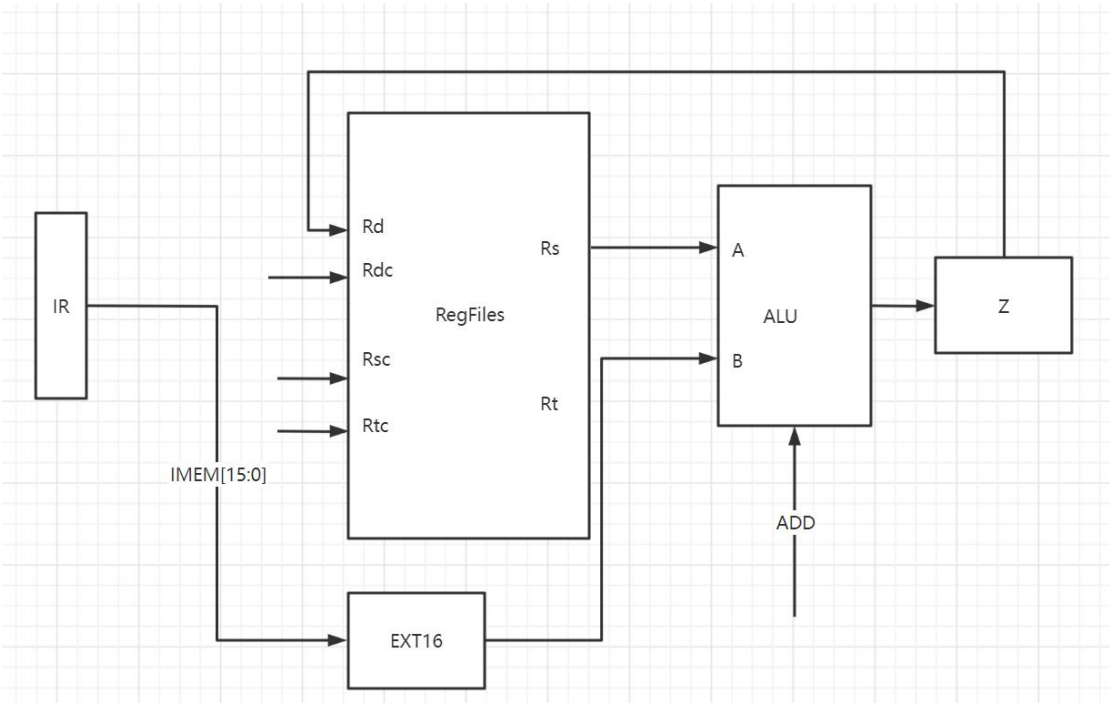
指令	PC	IR	MEM		RegFile	ALU		EXT16	Z
			Address	MD	Rd	A	B		
Fetch	Z	MD	PC			PC	4		ALU
ADDI					Z	Rs	EXT16	IR[15:0]	ALU

指令通路：

Fetch



ADDI:



2）addiu

周期数：4

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 读出 rs 寄存器的内容以及指令低 16 位送拓展器；

c) 由 ALU 完成计算；

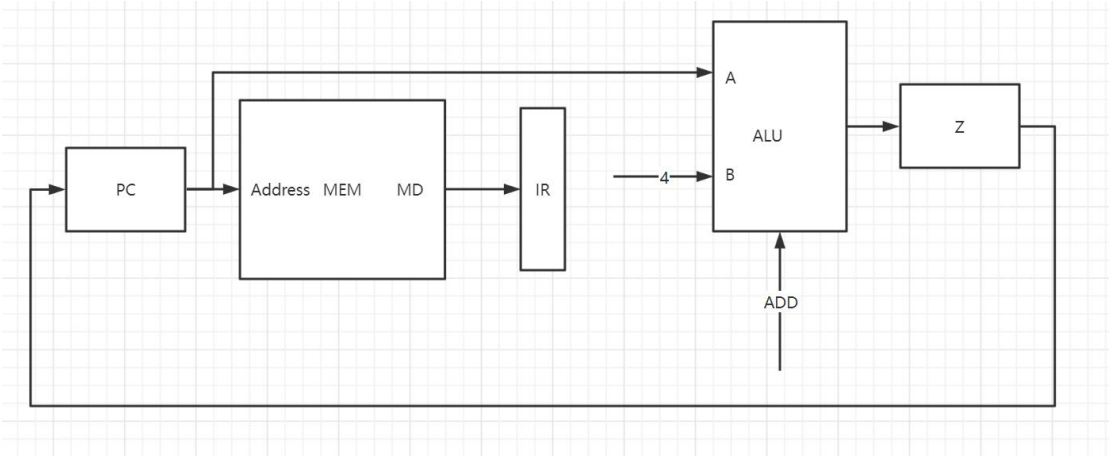
d) 把计算结果写入寄存器堆中的寄存器。

输入来源：

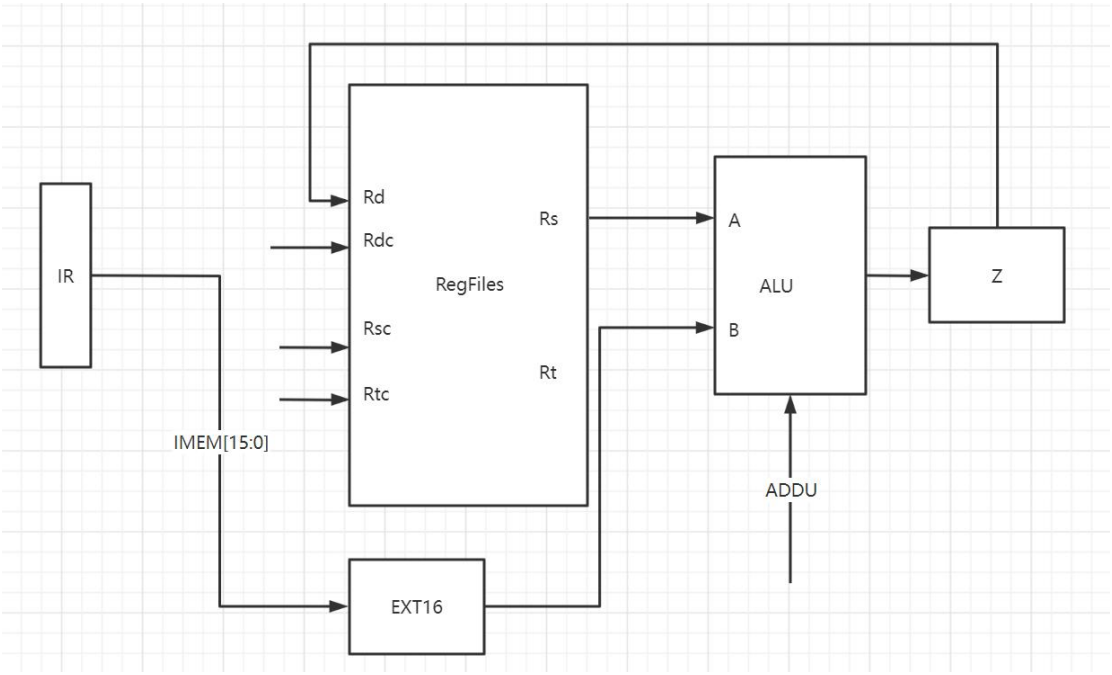
指令	PC	IR	MEM		RegFile	ALU		EXT16	Z
			Address	MD	Rd	A	B		
Fetch	Z	MD	PC			PC	4		ALU
ADDIU					Z	Rs	EXT16	IR[15:0]	ALU

指令通路：

Fetch



ADDIU:



3）andi

周期数：4

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 读出 rs 寄存器的内容以及指令低 16 位送拓展器（无符号）；

c) 由 ALU 完成计算；

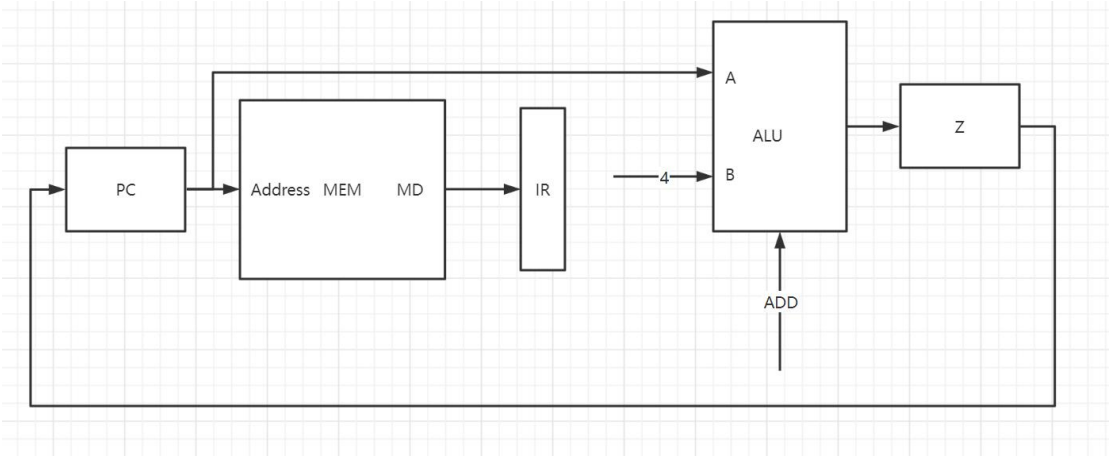
d) 把计算结果写入寄存器堆中的寄存器。

输入来源：

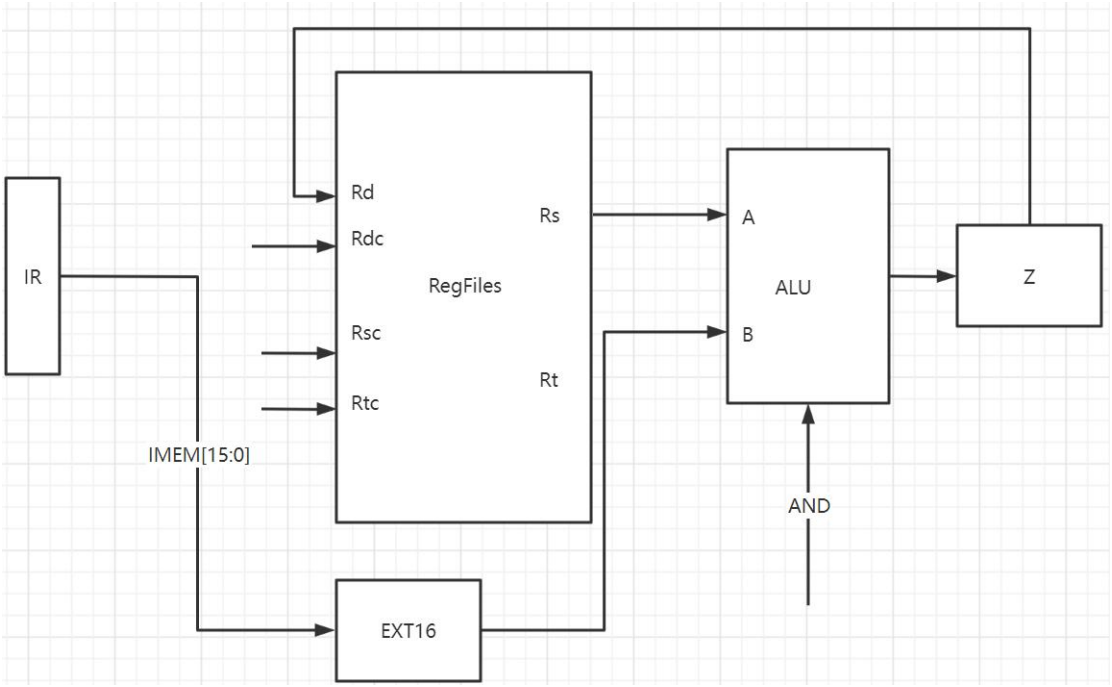
指令	PC	IR	MEM		RegFile	ALU		EXT16	Z
			Address	MD	Rd	A	B		
Fetch	Z	MD	PC			PC	4		ALU
ANDI					Z	Rs	EXT16	IR[15:0]	ALU

指令通路：

Fetch



ANDI:



4）ori

周期数：4

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 读出 rs 寄存器的内容以及指令低 16 位送拓展器（无符号）；

c) 由 ALU 完成计算；

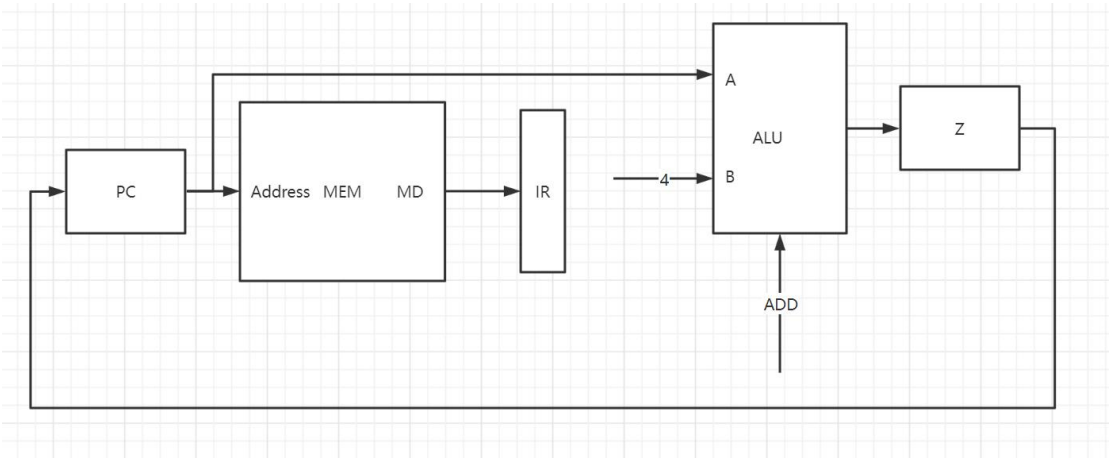
d) 把计算结果写入寄存器堆中的寄存器。

输入来源：

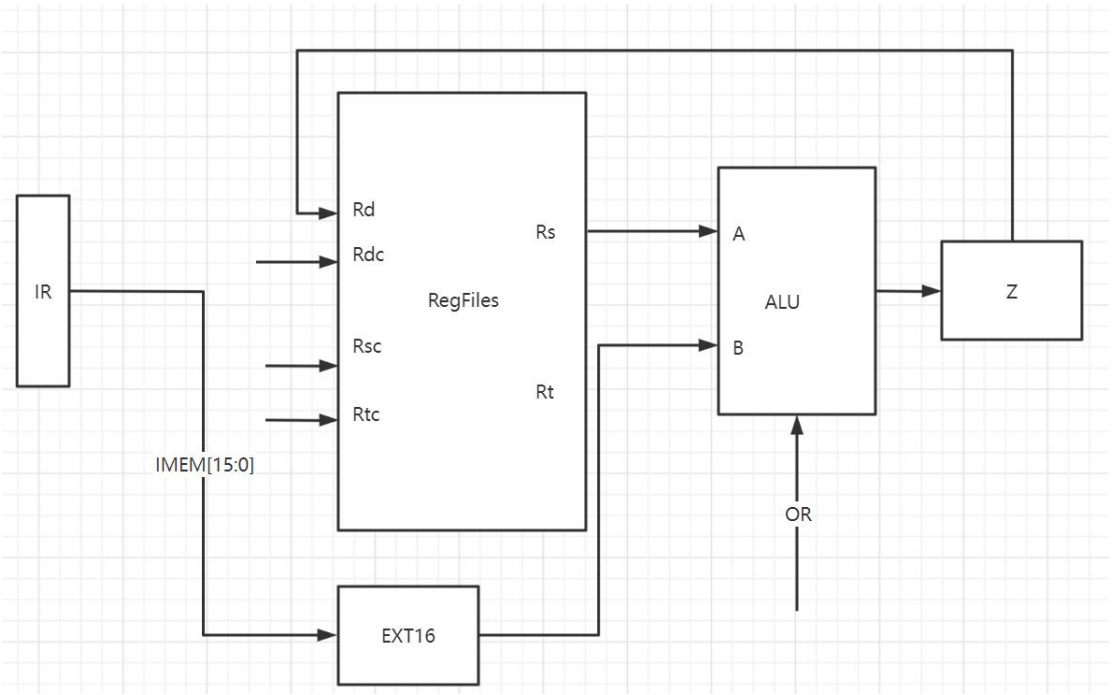
指令	PC	IR	MEM		RegFile	ALU		EXT16	Z
			Address	MD	Rd	A	B		
Fetch	Z	MD	PC			PC	4		ALU
ORI					Z	Rs	EXT16	IR[15:0]	ALU

指令通路：

Fetch



ORI:



5）Sltiu

周期数：4

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 读出 rs 寄存器的内容以及指令低 16 位送拓展器（有符号拓展）；

c) 由 ALU 完成计算；

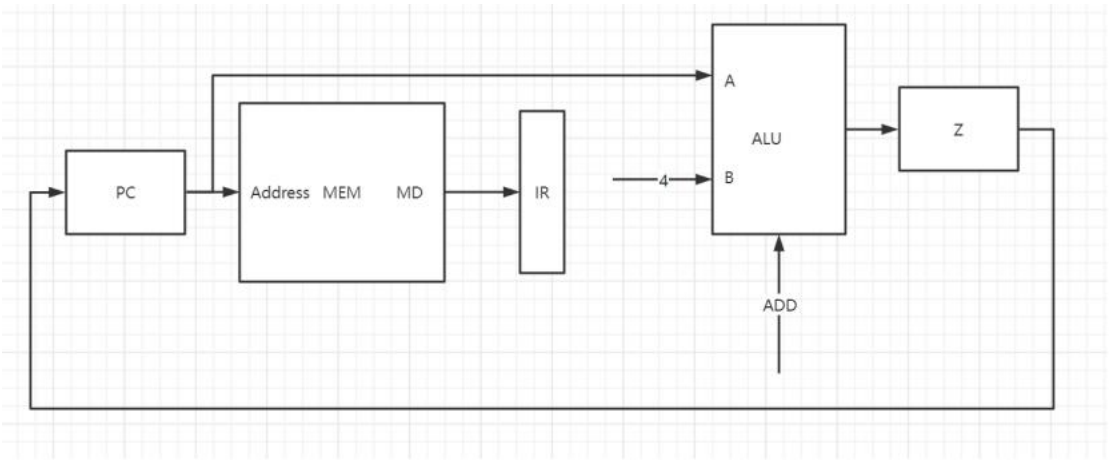
d) 把计算结果拓展后写入寄存器堆中的寄存器。

输入来源：

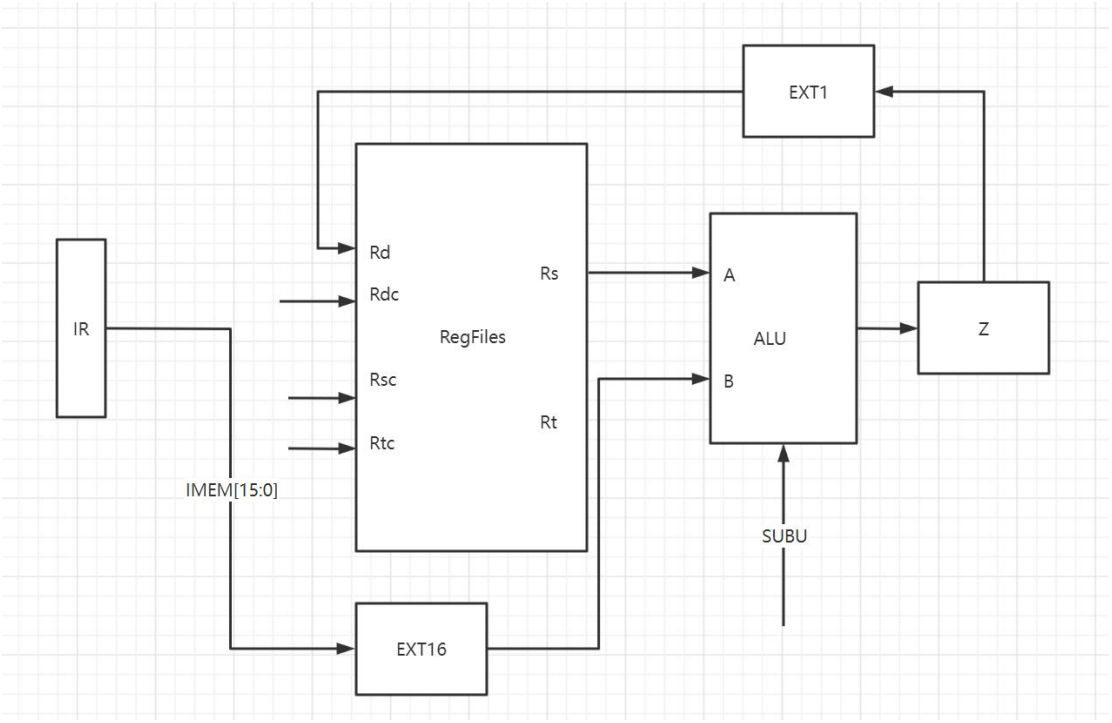
指令	PC	IR	MEM		RegFile	ALU		EXT16	Z
			Address	MD	Rd	A	B		
Fetch	Z	MD	PC			PC	4		ALU
SLTIU					Z	Rs	EXT16	IR[15:0]	ALU

指令通路：

Fetch



SLTIU:



6）Lui

周期数：4

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 读出 rs 寄存器的内容以及指令低 16 位送拓展器（有符号拓展）；

c) 由 ALU 完成计算；

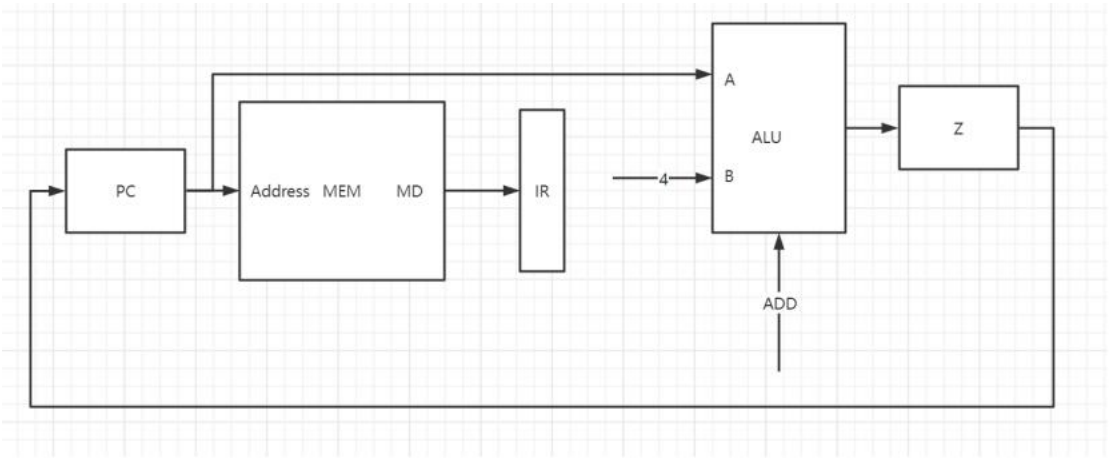
d) 把计算结果拓展后写入寄存器堆中的寄存器。

输入来源：

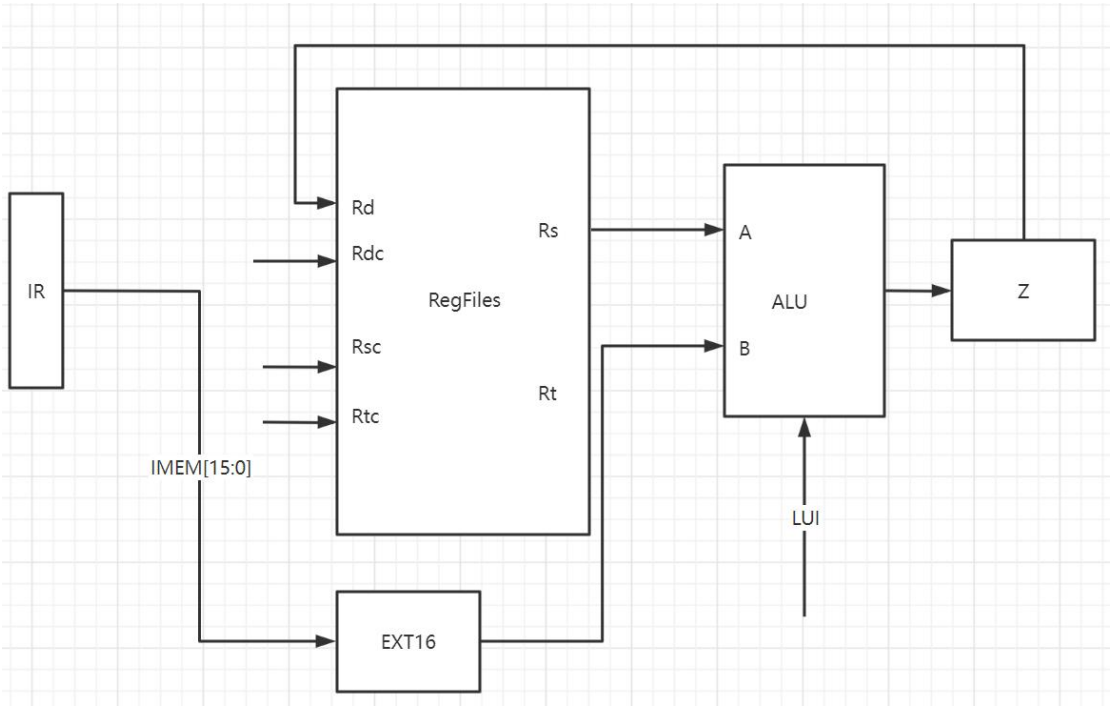
指令	PC	IR	MEM		RegFile	ALU		EXT16	Z
			Address	MD	Rd	A	B		
Fetch	Z	MD	PC			PC	4		ALU
LUI					Z	Rs	EXT16	IR[15:0]	ALU

指令通路：

Fetch



LUI:



7）xori

周期数：4

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 读出 rs 寄存器的内容以及指令低 16 位送拓展器（无符号拓展）；

c) 由 ALU 完成计算；

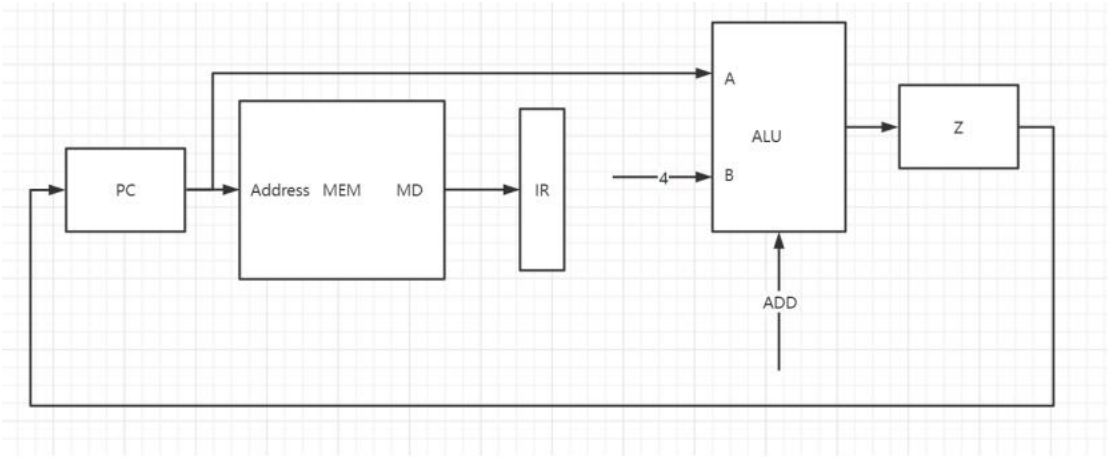
d) 把计算结果拓展后写入寄存器堆中的寄存器。

输入来源：

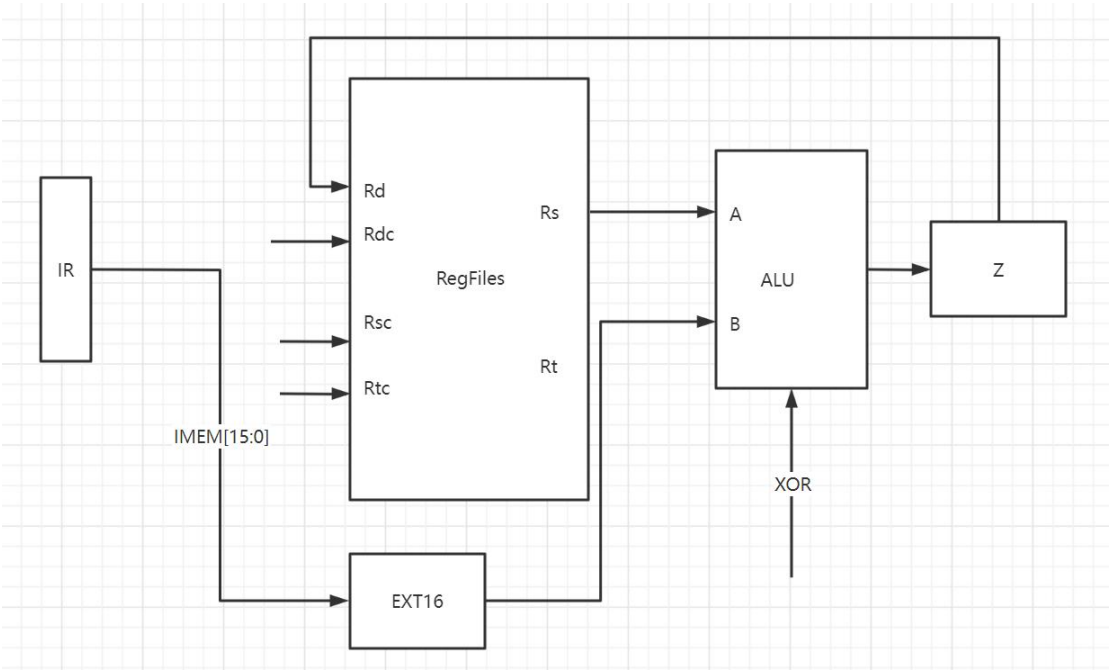
指令	PC	IR	MEM		RegFile	ALU		EXT16	Z
			Address	MD	Rd	A	B		
Fetch	Z	MD	PC			PC	4		ALU
XORI					Z	Rs	EXT16	IR[15:0]	ALU

指令通路：

Fetch



XORI:



8）slti

周期数：4

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 读出 rs 寄存器的内容以及指令低 16 位送拓展器（有符号拓展）；

c) 由 ALU 完成计算；

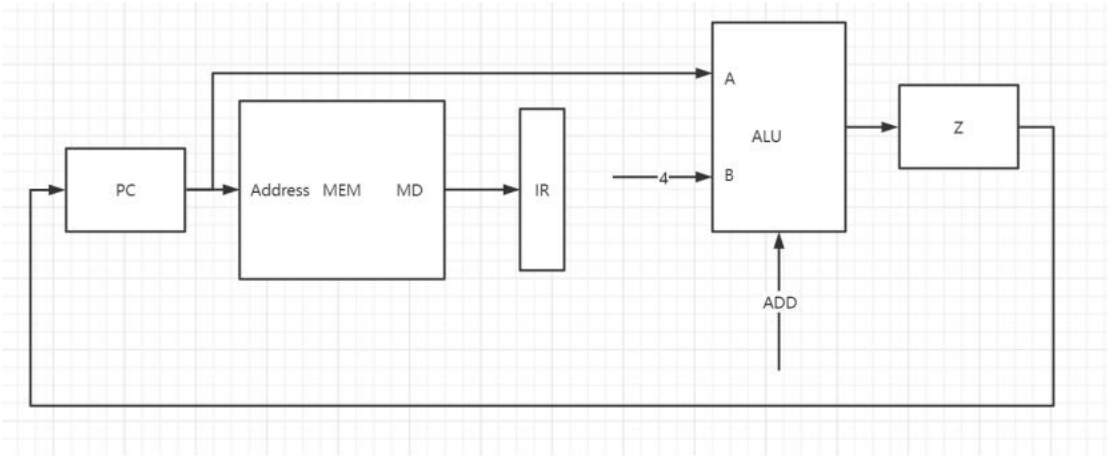
d) 把计算结果拓展后写入寄存器堆中的寄存器。

输入来源：

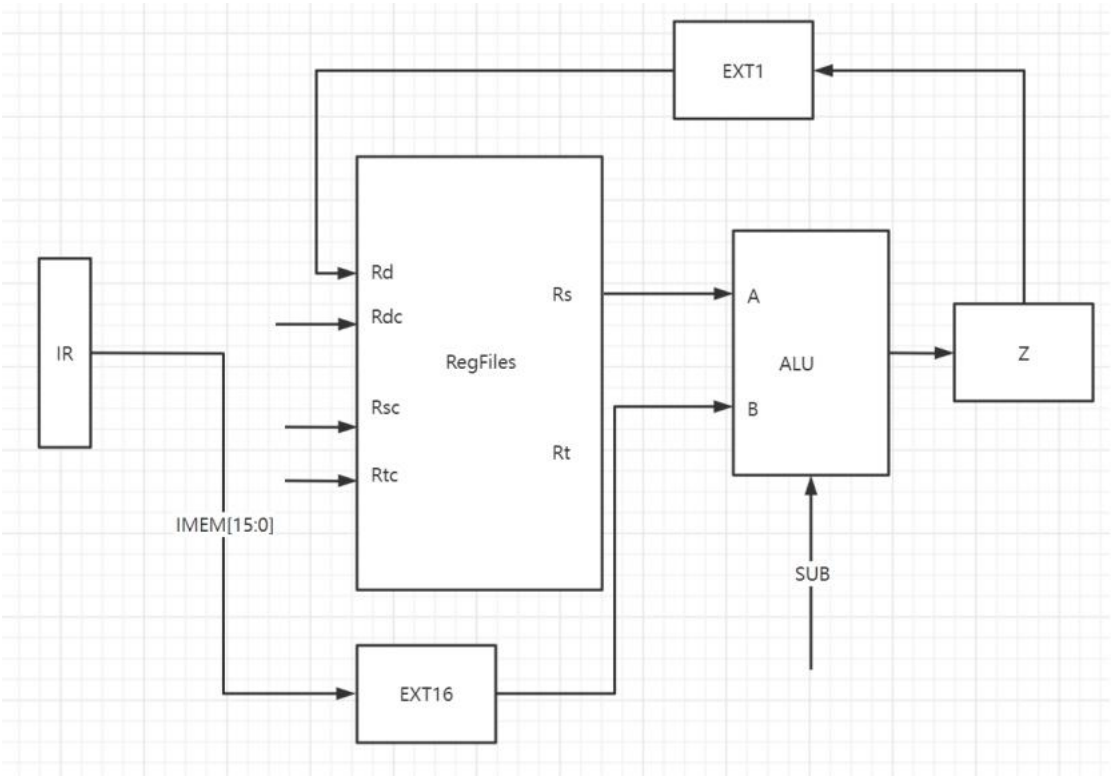
指令	PC	IR	MEM		RegFile	ALU		EXT16	Z
			Address	MD	Rd	A	B		
Fetch	Z	MD	PC			PC	4		ALU
SLTI					Z	Rs	EXT16	IR[15:0]	ALU

指令通路：

Fetch



SLTI:



9）addu

周期数：4

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 读出 rs 和 rt 寄存器的内容；

c) 由 ALU 完成计算；

d) 把计算结果写入寄存器堆中的寄存器。

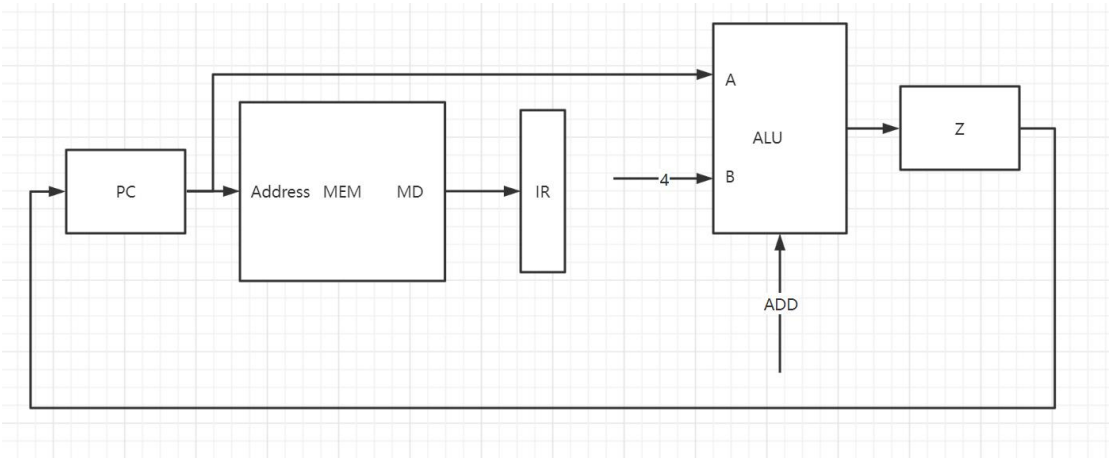
输入来源：

指令	PC	IR	MEM		RegFile	ALU		EXT16	Z
			Address	MD	Rd	A	B		

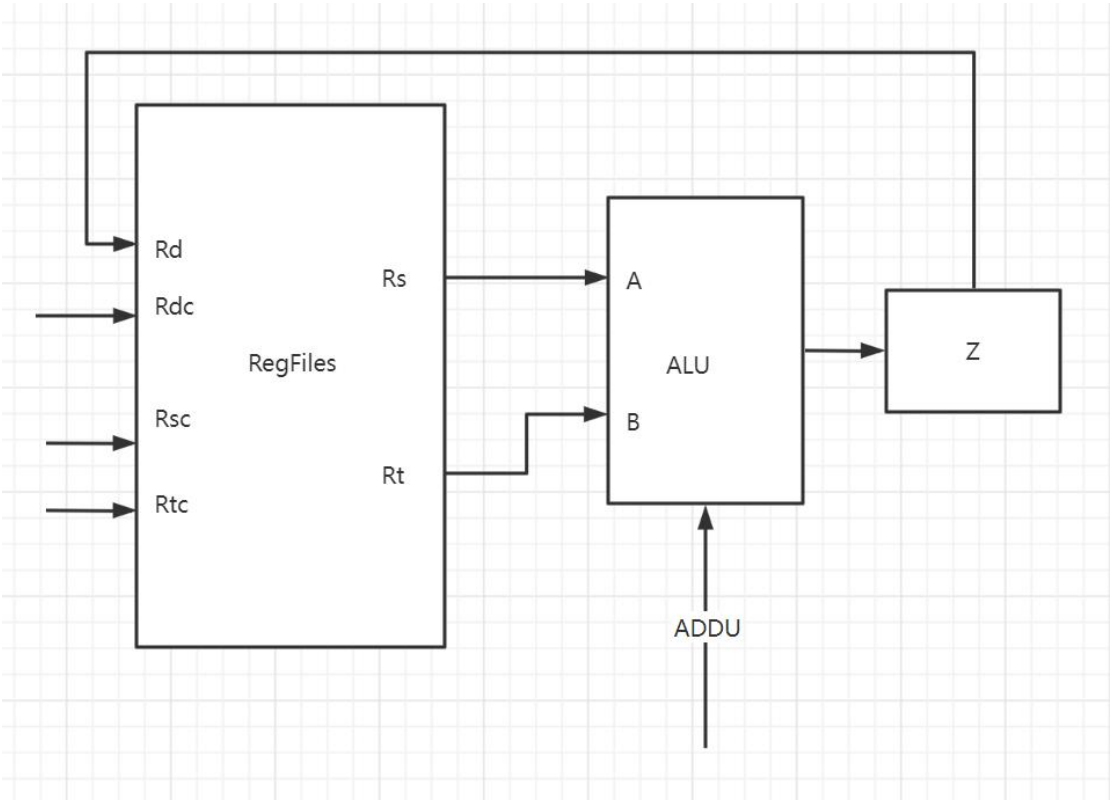
Fetch	Z	MD	PC			PC	4		ALU
ADDU					Z	Rs	Rt	IR[15:0]	ALU

指令通路：

Fetch



ADDU:



10）and

周期数：4

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 读出 rs 和 rt 寄存器的内容；

c) 由 ALU 完成计算；

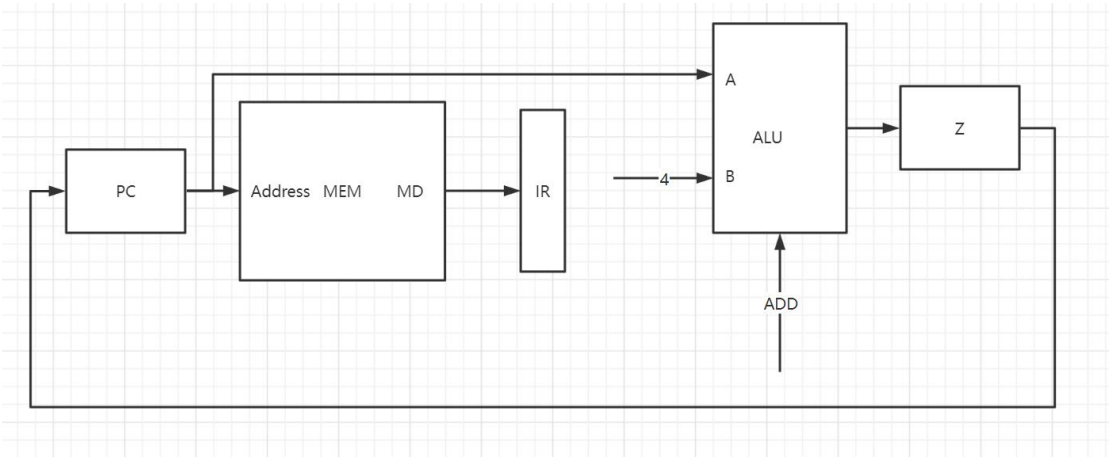
d) 把计算结果写入寄存器堆中的寄存器。

输入来源：

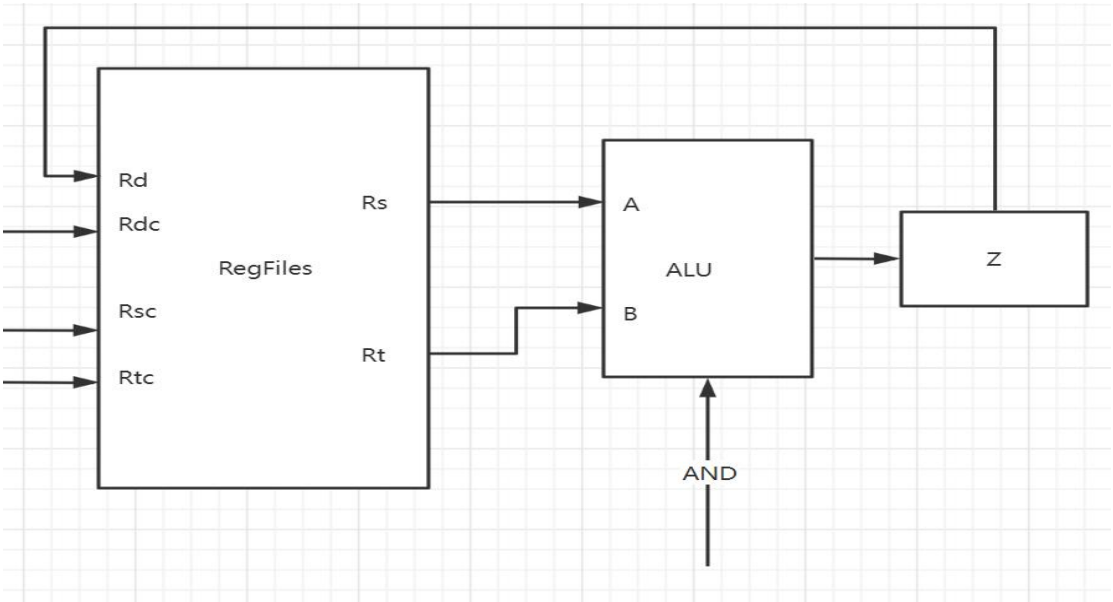
指令	PC	IR	MEM		RegFile	ALU		EXT16	Z
			Address	MD	Rd	A	B		
Fetch	Z	MD	PC			PC	4		ALU
AND					Z	Rs	Rt	IR[15:0]	ALU

指令通路：

Fetch



AND:



11）beq

周期数：4

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 读出 rs 和 rt 两个寄存器的数据并锁存

C) ALU 计算转移地址并锁存；

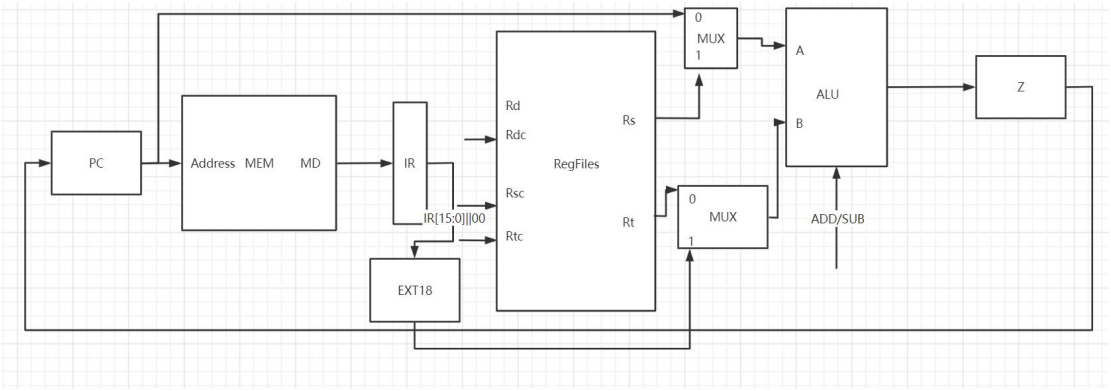
c) 由 ALU 比较两个寄存器数据，并决定是否把转移地址写入 PC。

输入来源：

指令	PC	IR	MEM		RegFile	ALU		EXT18	Z
			Address	MD	Rd	A	B		

Fetch	Z	MD	PC			PC	4		ALU
BEQ	Z					PC	EXT18	IR[15:0]||00	ALU

指令通路：



12）bne

周期数：4

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 读出 rs 和 rt 两个寄存器的数据并锁存

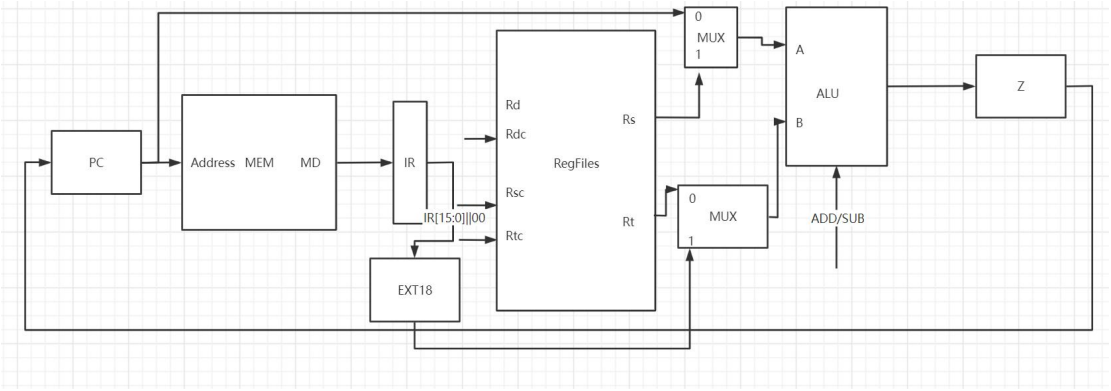
C) ALU 计算转移地址并锁存；

c) 由 ALU 比较两个寄存器数据，并决定是否把转移地址写入 PC。

输入来源：

指令	PC	IR	MEM		RegFile	ALU		EXT18	Z
			Address	MD	Rd	A	B		
Fetch	Z	MD	PC			PC	4		ALU
BNE	Z					PC	EXT18	IR[15:0]||00	ALU

指令通路：



13）j

周期数：2

操作流程：

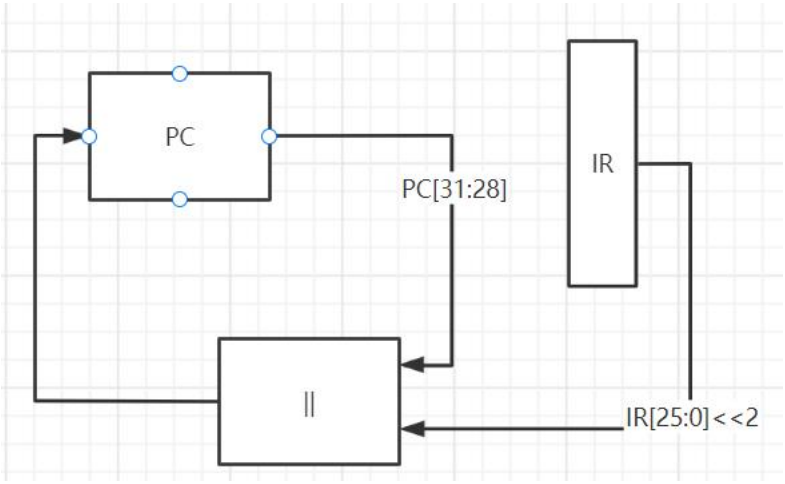
a) 根据 PC 取指令，并把 PC 加 4；

b) 指令中的 address 左移两位与 PC 的高 4 位拼接起来，写入 PC。

输入来源：

指令	PC	IR	MEM		RegFile	ALU		||		Z
			Address	MD	Rd	A	B	A	B	
Fetch	Z	MD	PC			PC	4			ALU
J	||							IR[25:0]<<2	PC[31:28]	

指令通路：

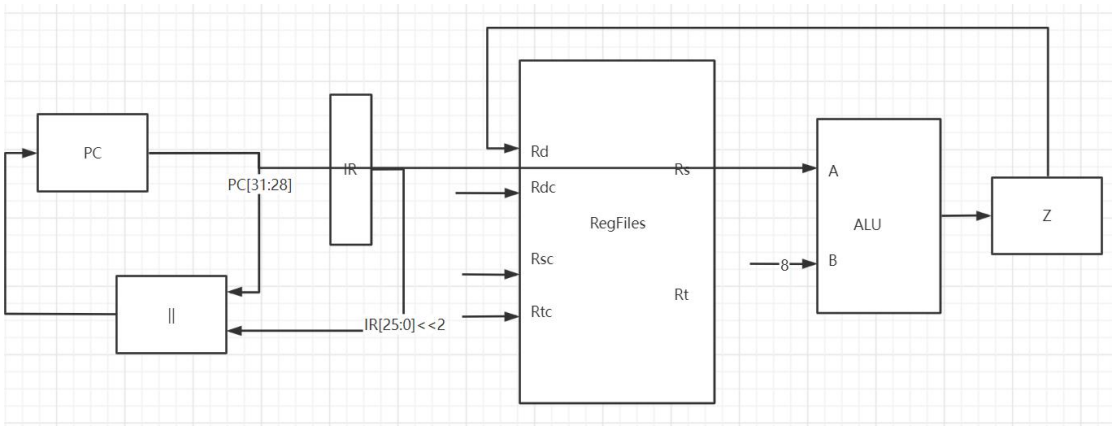


14）jal
周期数：4
操作流程：
a) 根据 PC 取指令，并把 PC 加 4；
b) 取址
c)PC 加 8 送 ALU 运算。
D)结果写回 rf。指令中的 address 左移两位与 PC 的高 4 位拼接起来，写入 PC

输入来源：

指令	PC	IR	MEM		RegFile	ALU		||		Z
			Address	MD	Rd	A	B	A	B	
Fetch	Z	MD	PC			PC	4			ALU
JAL	||							IR[25:0]<<2	PC[31:28]	

指令通路：



15）jr

周期数：2

操作流程：

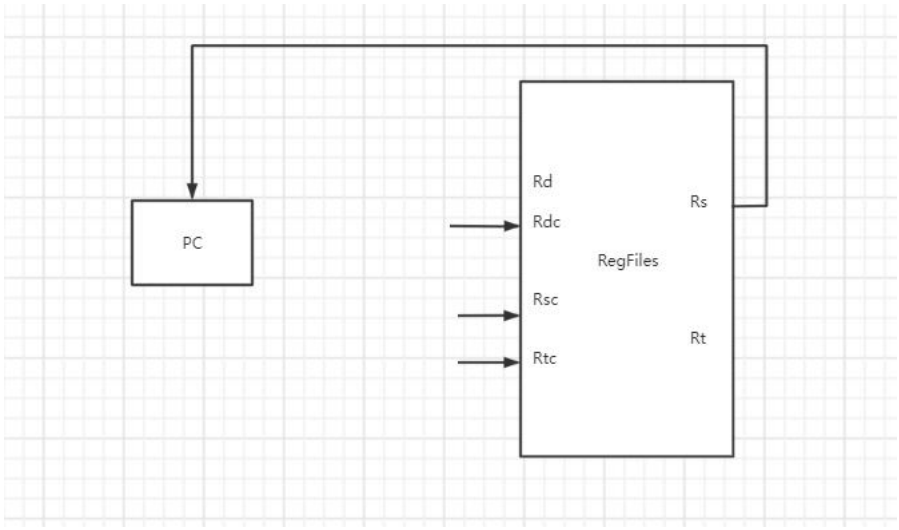
a) 根据 PC 取指令，并把 PC 加 4；

b) 读出 rs 寄存器的内容并写入 pc。

输入来源：

指令	PC	IR	MEM		RegFile
			Address	MD	Rd
Fetch	Z	MD	PC		
JR	Rs				

指令通路：



16）lw

周期数：5

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 对指令译码并读出 rs 寄存器的内容；

c) rs 寄存器的内容与指令中的偏移量 offset 相加，计算得到存储器地址；

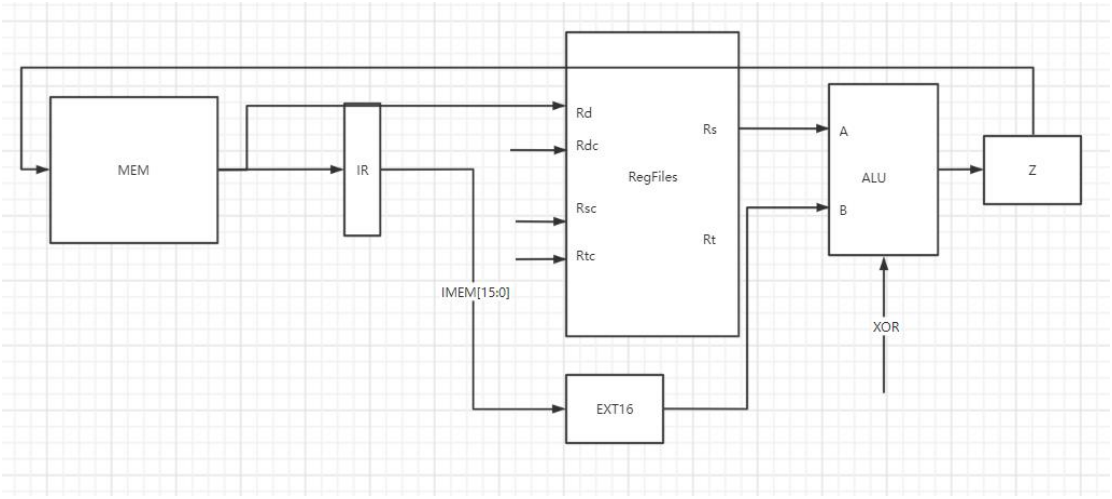
d) 使用计算好的地址访问存储器，从中读出一个 32 位的数据；

e) 把该数据写入寄存器堆中的 rt 寄存器。

输入来源：

指令	PC	IR	MEM		RegFile	ALU		Z
			Address	MD	Rt	A	B	
Fetch	Z	MD	PC			PC	4	ALU
LW			Z		MD	Rs	IR[15:0]	ALU

指令通路：



17）xor

周期数：4

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 读出 rs,rt 寄存器的内容；

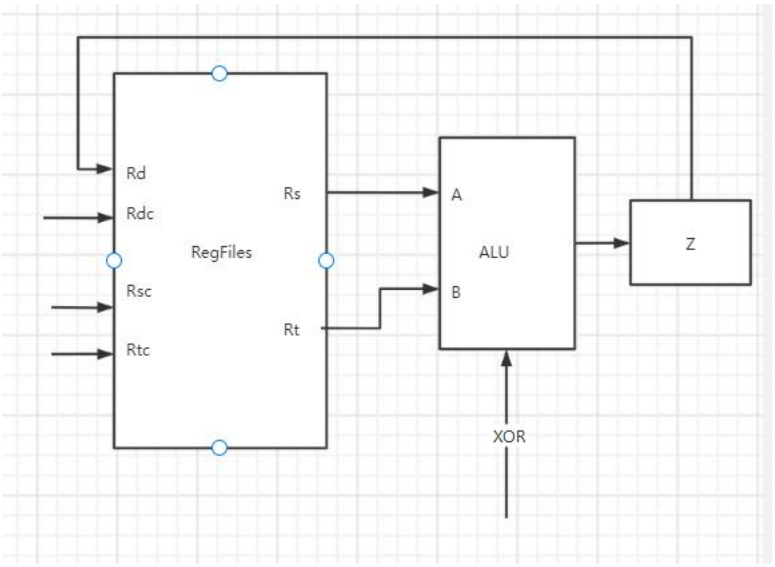
c) 由 ALU 完成计算；

d) 把计算结果拓展后写入寄存器堆中的寄存器。

输入来源：

指令	PC	IR	MEM		RegFile	ALU		EXT16	Z
			Address	MD	Rd	A	B		
Fetch	Z	MD	PC			PC	4		ALU
XOR					Z	Rs	Rt		ALU

指令通路：



18）nor

周期数：4

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 读出 rs,rt 寄存器的内容；

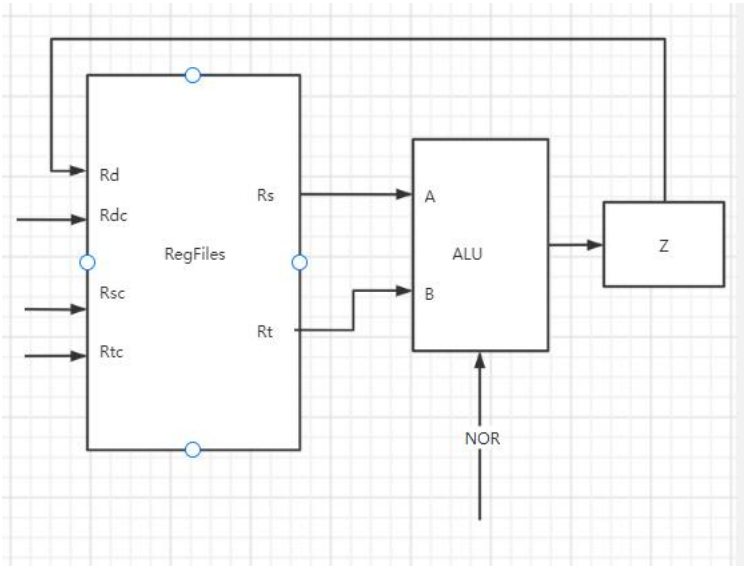
c) 由 ALU 完成计算；

d) 把计算结果拓展后写入寄存器堆中的寄存器。

输入来源：

指令	PC	IR	MEM		RegFile	ALU		EXT16	Z
			Address	MD	Rd	A	B		
Fetch	Z	MD	PC			PC	4		ALU
NOR					Z	Rs	Rt		ALU

指令通路：



19）Or

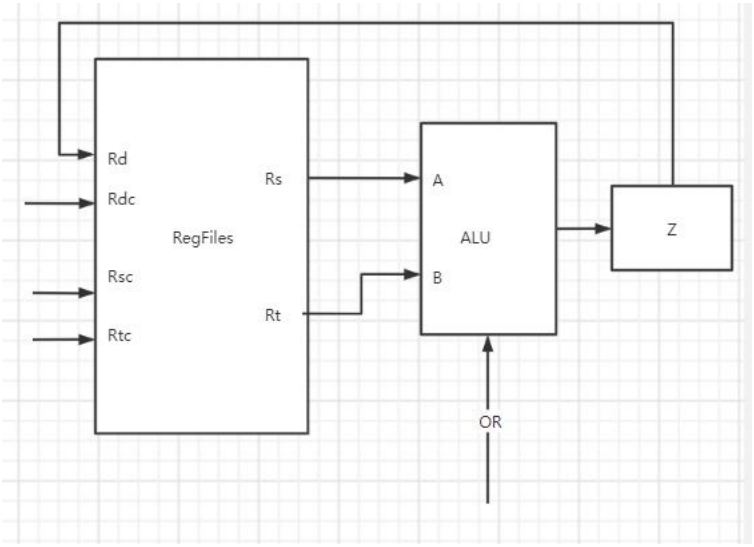
周期数：4

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；
b) 读出 rs,rt 寄存器的内容；
c) 由 ALU 完成计算；
d) 把计算结果拓展后写入寄存器堆中的寄存器。
输入来源：

指令	PC	IR	MEM		RegFile	ALU		EXT16	Z
			Address	MD	Rd	A	B		
Fetch	Z	MD	PC			PC	4		ALU
OR					Z	Rs	Rt		ALU

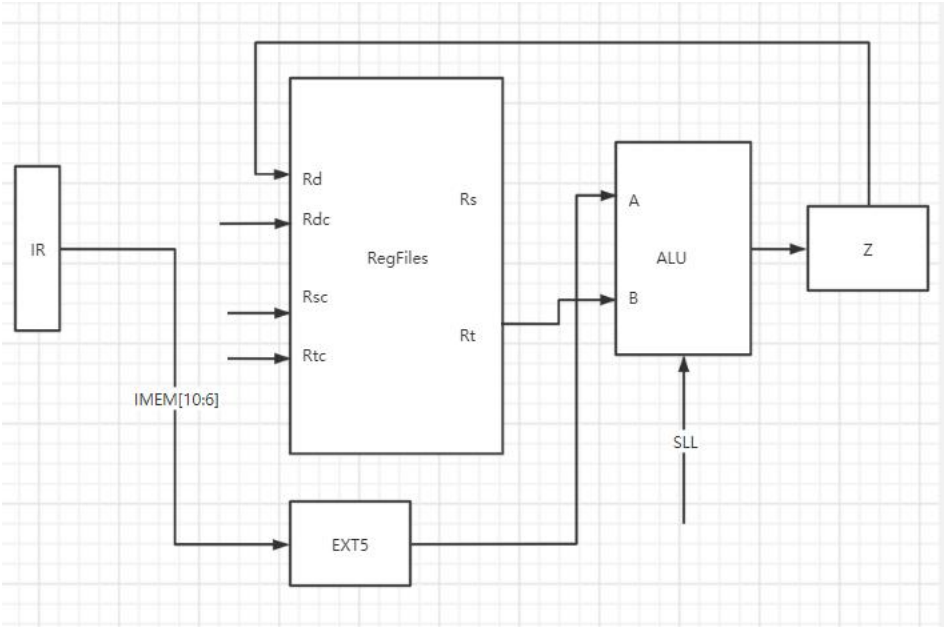
指令通路：



20）sll
周期数：4
操作流程：
a) 根据 PC 取指令，并把 PC 加 4；
b) 读出 rs,rt 寄存器的内容；
c) 由 ALU 完成计算；
d) 把计算结果拓展后写入寄存器堆中的寄存器。
输入来源：

指令	PC	IR	MEM		RegFile	ALU		EXT5	Z
			Address	MD	Rd	A	B		
Fetch	Z	MD	PC			PC	4		ALU
SLL					Z	EXT5	Rt	IR[10:6]	ALU

指令通路：



21）sllv

周期数：4

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 读出 rs,rt 寄存器的内容；

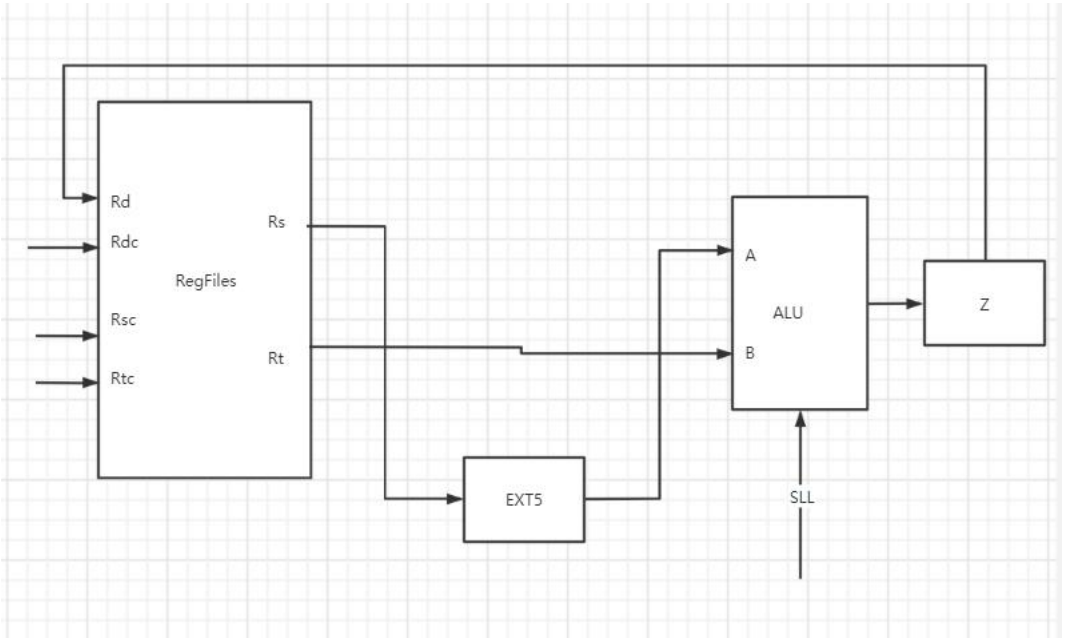
c) 由 ALU 完成计算；

d) 把计算结果拓展后写入寄存器堆中的寄存器。

输入来源：

指令	PC	IR	MEM		RegFile	ALU		EXT5	Z
			Address	MD	Rd	A	B		
Fetch	Z	MD	PC			PC	4		ALU
SLLV					Z	EXT5	Rt	Rs	ALU

指令通路：



22）sltu

周期数：4

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 读出 rs,rt 寄存器的内容；

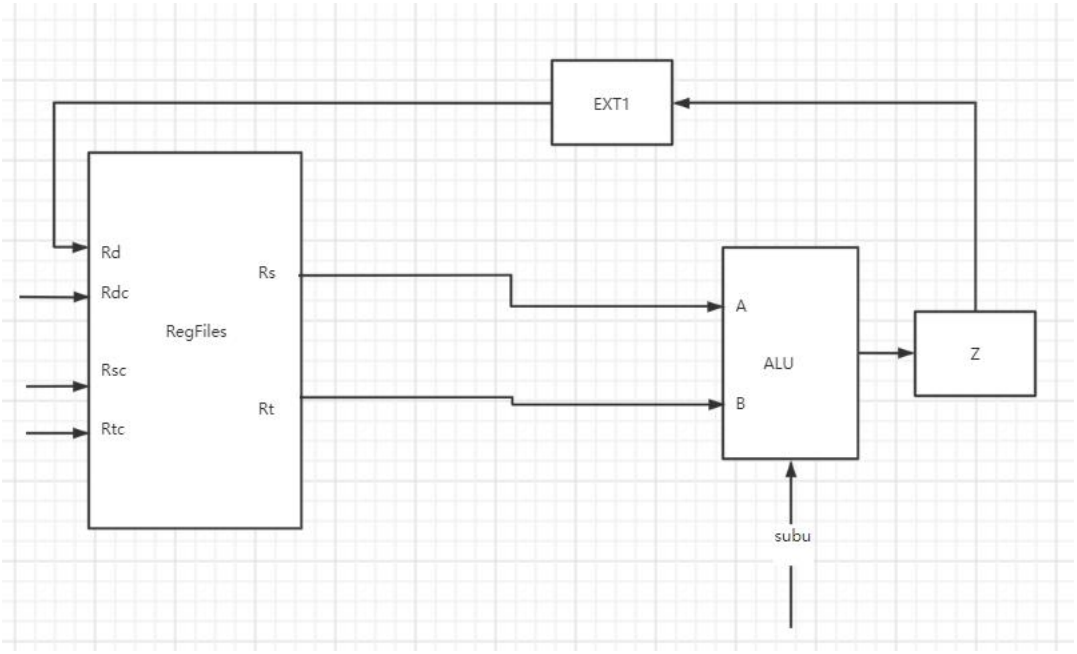
c) 由 ALU 完成计算；

d) 把计算结果拓展后写入寄存器堆中的寄存器。

输入来源：

指令	PC	IR	MEM		RegFile	ALU		EXT1	Z
			Address	MD	Rd	A	B		
Fetch	Z	MD	PC			PC	4		ALU
SLTU					EXT1	Rs	Rt	Z	ALU

指令通路：



23）sra

周期数：4

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 读出 rs,rt 寄存器的内容；

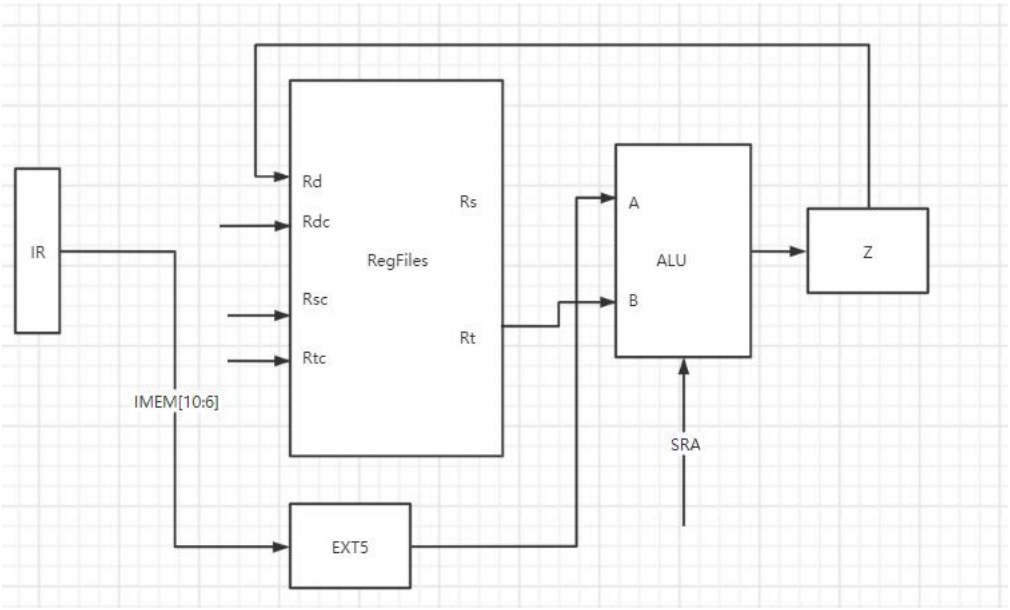
c) 由 ALU 完成计算；

d) 把计算结果拓展后写入寄存器堆中的寄存器。

输入来源：

指令	PC	IR	MEM		RegFile	ALU		EXT5	Z
			Address	MD	Rd	A	B		
Fetch	Z	MD	PC			PC	4		ALU
SRA					Z	EXT5	Rt	IR[10:6]	ALU

指令通路：



24）srl

周期数：4

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 读出 rs,rt 寄存器的内容；

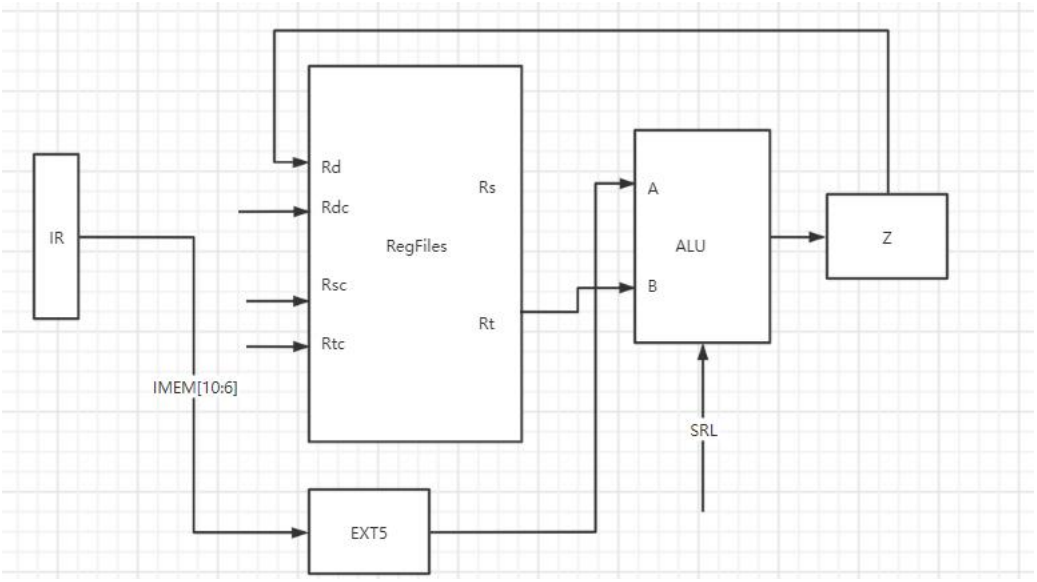
c) 由 ALU 完成计算；

d) 把计算结果拓展后写入寄存器堆中的寄存器。

输入来源：

指令	PC	IR	MEM		RegFile	ALU		EXT5	Z
			Address	MD	Rd	A	B		
Fetch	Z	MD	PC			PC	4		ALU
SRA					Z	EXT5	Rt	IR[10:6]	ALU

指令通路：



25）subu

周期数：4

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 读出 rs 寄存器的内容以及指令低 16 位送拓展器；

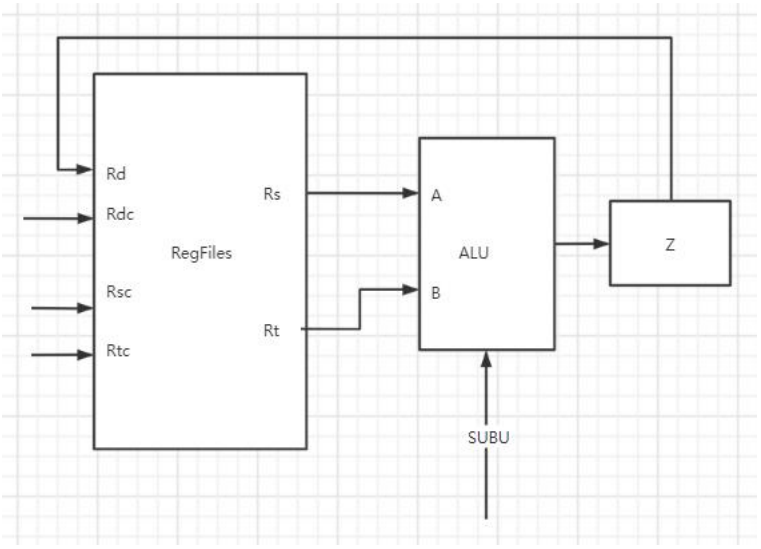
c) 由 ALU 完成计算；

d) 把计算结果写入寄存器堆中的寄存器。

输入来源：

指令	PC	IR	MEM		RegFile	ALU		Z
			Address	MD	Rd	A	B	
Fetch	Z	MD	PC			PC	4	ALU
SUBU					Z	Rs	Rt	ALU

指令通路：



26）sw

周期数：4

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 对指令译码并读出 rs 和 rt 寄存器的内容；

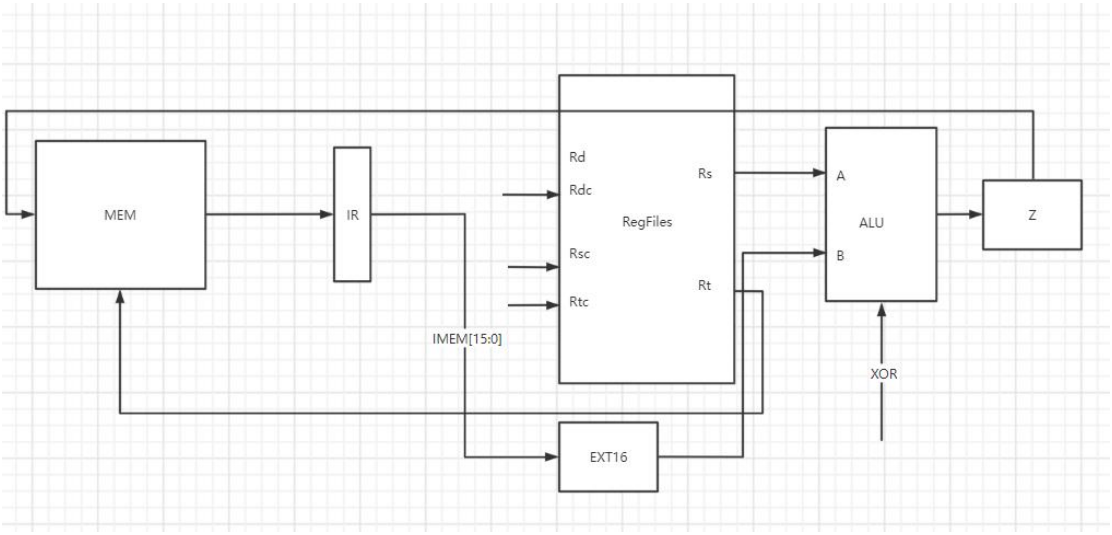
c) rs 寄存器的内容与指令中的偏移量 offset 相加，计算得到存储器地址；

d) 使用计算好的地址访问存储器，将 rt 寄存器内容写入；

输入来源：

指令	PC	IR	MEM		RegFile	ALU		Z
			Address	MD	Rd	A	B	
Fetch	Z	MD	PC			PC	4	ALU
LW			Z	Rt		Rs	IR[15:0]	ALU

指令通路：



27）add

周期数：4

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 读出 rs 寄存器的内容以及指令低 16 位送拓展器；

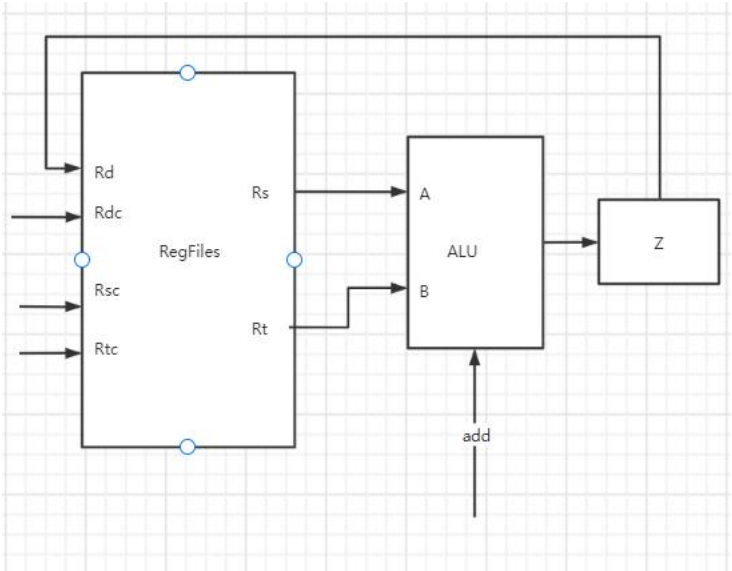
c) 由 ALU 完成计算；

d) 把计算结果写入寄存器堆中的寄存器。

输入来源：

指令	PC	IR	MEM		RegFile	ALU		Z
			Address	MD	Rd	A	B	
Fetch	Z	MD	PC			PC	4	ALU
ADD					Z	Rs	Rt	ALU

指令通路：



28）sub

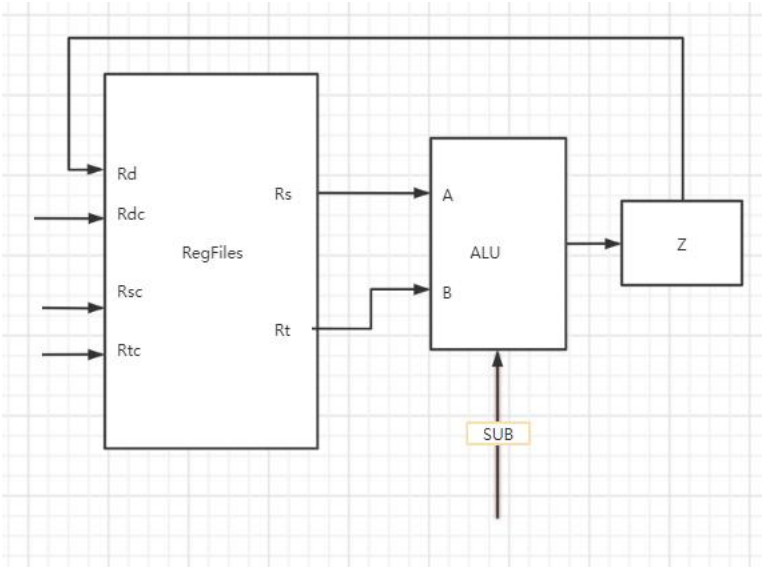
周期数：4

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；
b) 读出 rs 寄存器的内容以及指令低 16 位送拓展器；
c) 由 ALU 完成计算；
d) 把计算结果写入寄存器堆中的寄存器。
输入来源：

指令	PC	IR	MEM		RegFile	ALU		Z
			Address	MD	Rd	A	B	
Fetch	Z	MD	PC			PC	4	ALU
SUB					Z	Rs	Rt	ALU

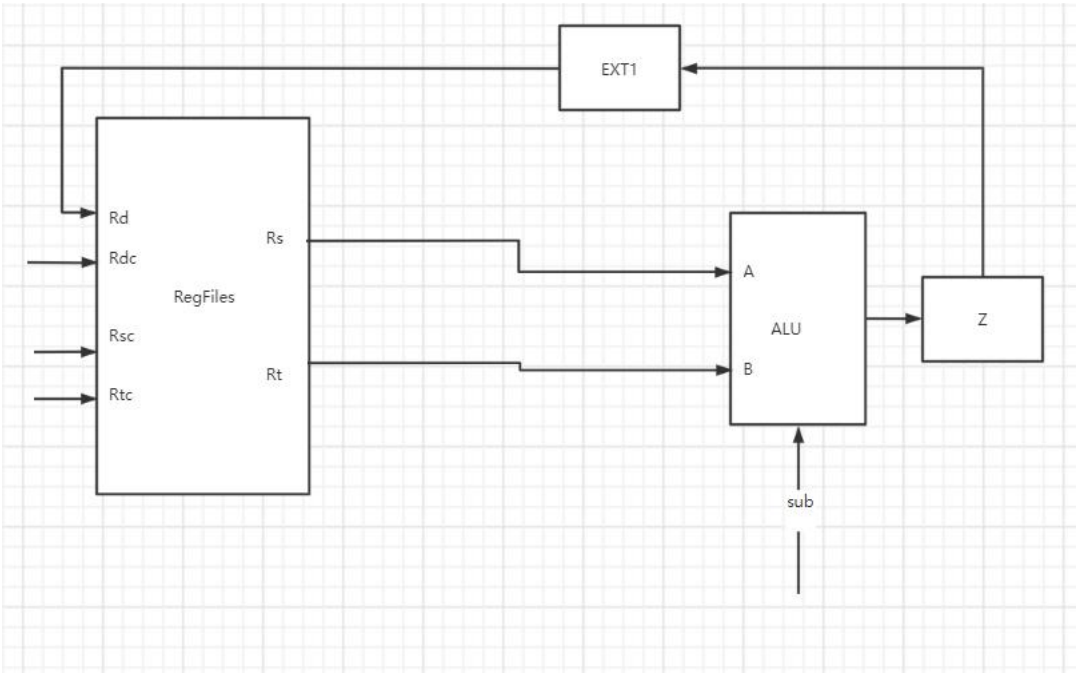
指令通路：



29）slt
周期数：4
操作流程：
a) 根据 PC 取指令，并把 PC 加 4；
b) 读出 rs,rt 寄存器的内容；
c) 由 ALU 完成计算；
d) 把计算结果拓展后写入寄存器堆中的寄存器。
输入来源：

指令	PC	IR	MEM		RegFile	ALU		EXT1	Z
			Address	MD	Rd	A	B		
Fetch	Z	MD	PC			PC	4		ALU
SLT					EXT1	Rs	Rt	Z	ALU

指令通路：



30）srlv

周期数：4

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 读出 rs,rt 寄存器的内容；

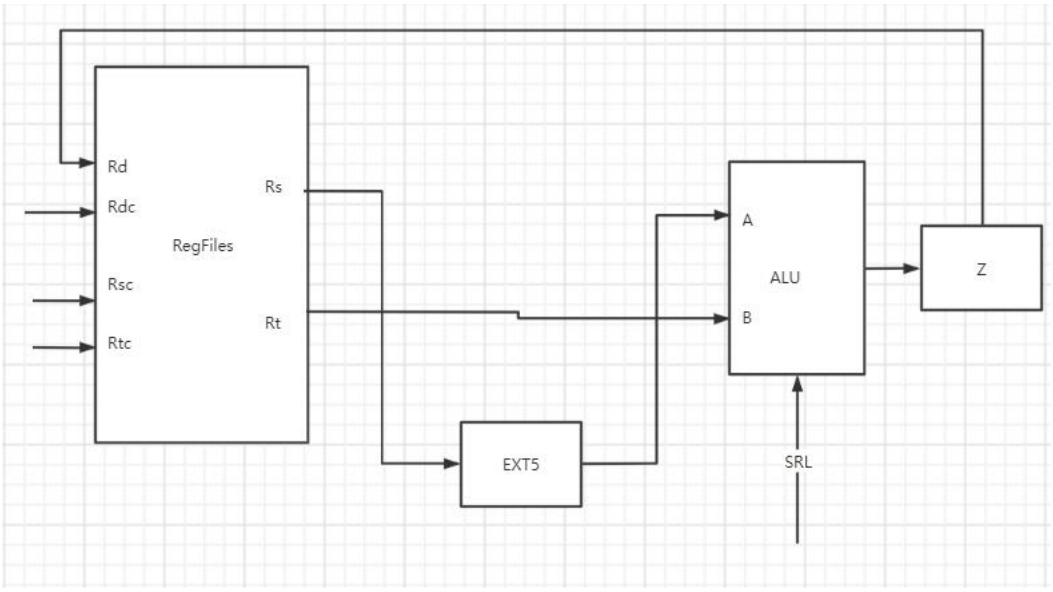
c) 由 ALU 完成计算；

d) 把计算结果拓展后写入寄存器堆中的寄存器。

输入来源：

指令	PC	IR	MEM		RegFile	ALU		EXT5	Z
			Address	MD	Rd	A	B		
Fetch	Z	MD	PC			PC	4		ALU
SLLV					Z	EXT5	Rt	Rs	ALU

指令通路：



31）srav

周期数：4

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 读出 rs,rt 寄存器的内容；

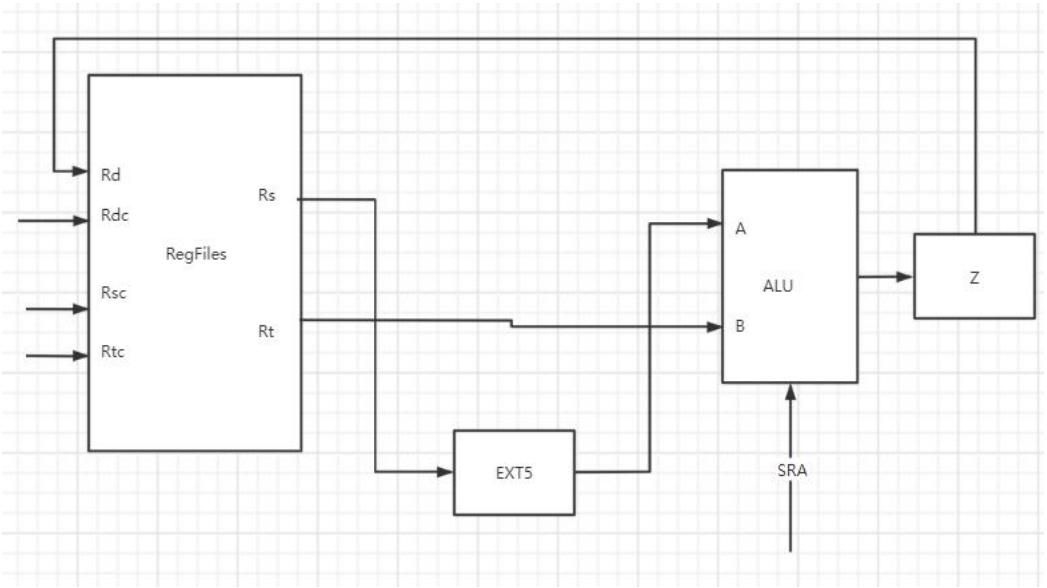
c) 由 ALU 完成计算；

d) 把计算结果拓展后写入寄存器堆中的寄存器。

输入来源：

指令	PC	IR	MEM		RegFile	ALU		EXT5	Z
			Address	MD	Rd	A	B		
Fetch	Z	MD	PC			PC	4		ALU
SRAV					Z	EXT5	Rt	Rs	ALU

指令通路：



32）clz

周期数：4

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 读出 rs 寄存器的内容；

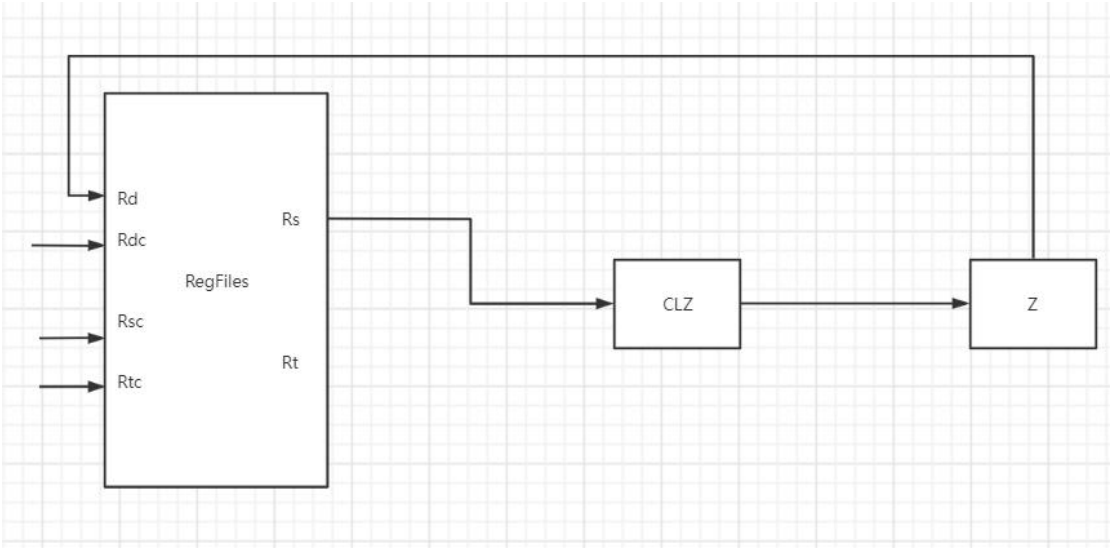
c) 由 CLZ 完成计算；

d) 把计算结果写入 rd 寄存器。

输入来源：

指令	PC	IR	MEM		RegFile	ALU		CLZ	Z
			Address	MD	Rd	A	B		
Fetch	Z	MD	PC			PC	4		ALU
CLZ					Z			Rs	CLZ

指令通路：



33）divu

周期数：4

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 读出 rs,rt 寄存器的内容；

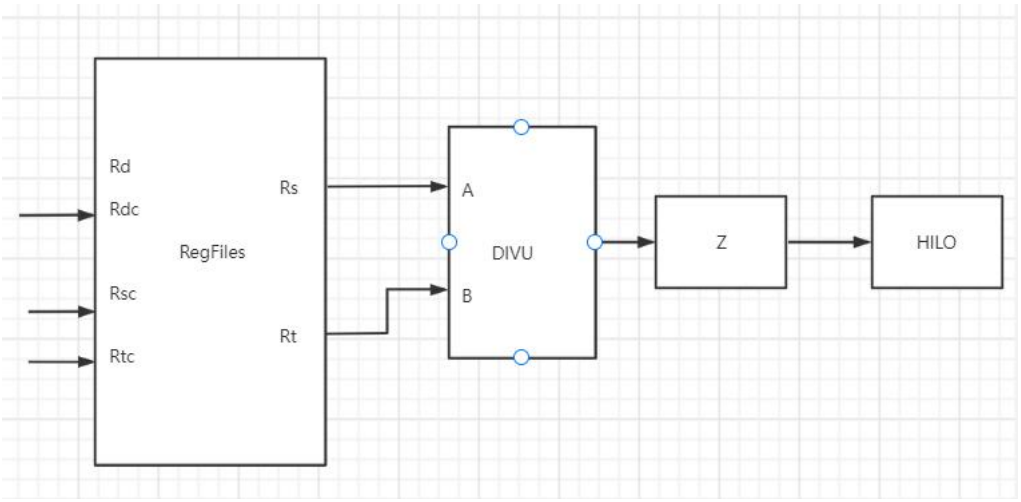
c) 由运算单元完成计算；

d) 把计算结果写入 HILO 寄存器。

输入来源：

指令	PC	IR	MEM		RegFile	ALU		DIVU		Z	HILO
			Address	MD	Rd	A	B	A	B		
Fetch	Z	MD	PC			PC	4			ALU	
DIVU								rs	rt	DIVU	Z

指令通路：



34）eret

周期数：3

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

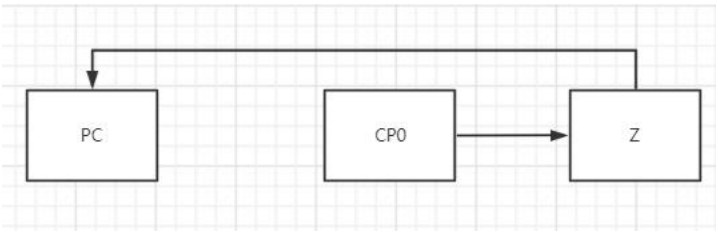
b) CP0 进行协处理；

c) 中断地址写回 pc。

输入来源：

指令	PC	IR	MEM		RegFile	ALU		CP0		Z
			Address	MD	Rd	A	B	ERET		
Fetch	Z	MD	PC			PC	4			ALU
DIVU	Z							1		CP0

指令通路：



35）jalr

周期数：4

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 读取 rs 寄存器并锁存。

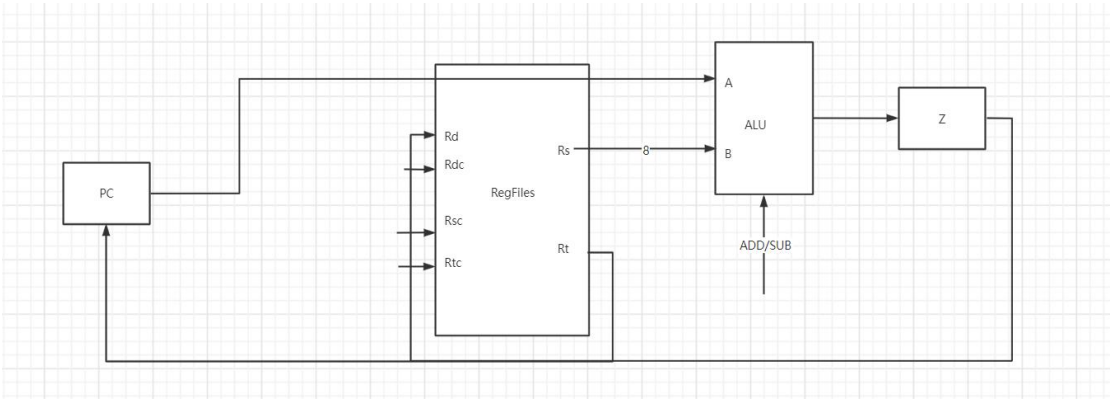
C)PC 加 8 存入 Z

d)rs 写入 PC,z 写入 rd 寄存器

输入来源：

指令	PC	IR	MEM		RegFile	ALU		Z
			Address	MD	Rd	A	B	
Fetch	Z	MD	PC			PC	4	ALU
JAL	Rs				Z	PC	8	ALU

指令通路：



36）lb

周期数：5

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 对指令译码并读出 rs 寄存器的内容；

c) rs 寄存器的内容与指令中的偏移量 offset 相加，计算得到存储器地址；

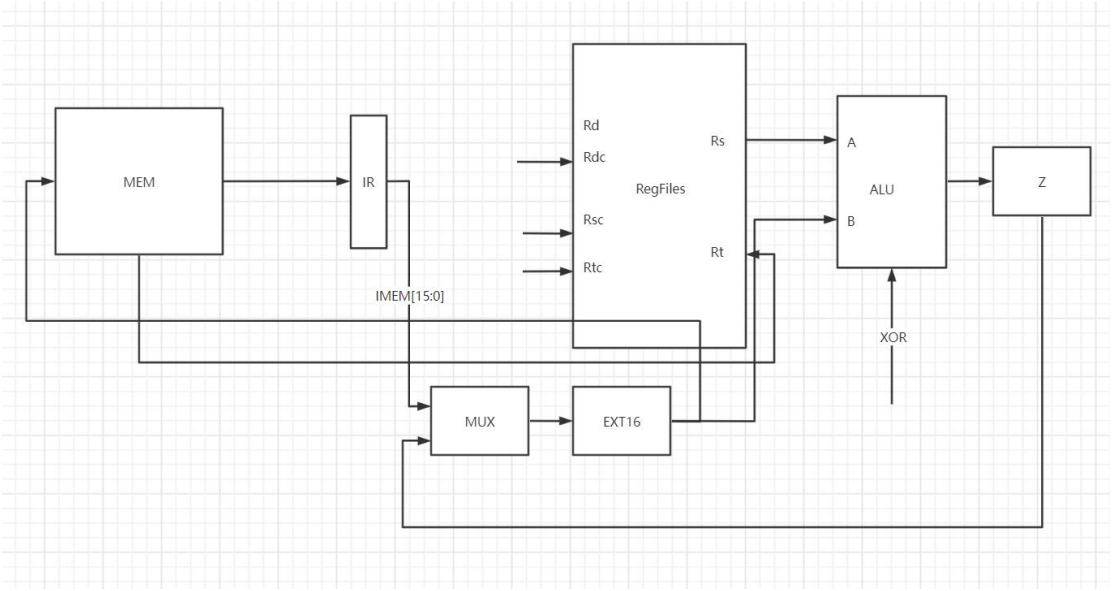
d) 使用计算好的地址访问存储器，从中读出一个 8 位的数据；

e) 把该数据符号拓展后写入寄存器堆中的 rt 寄存器。

输入来源：

指令	PC	IR	MEM		RegFile	ALU		Z	EXT16
			Address	MD	Rt	A	B		
Fetch	Z	MD	PC			PC	4	ALU	
LB			Z		EXT16	Rs	IR[15:0]	ALU	MD

指令通路：



37）lbu

周期数：5

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 对指令译码并读出 rs 寄存器的内容；

c) rs 寄存器的内容与指令中的偏移量 offset 相加，计算得到存储器地址；

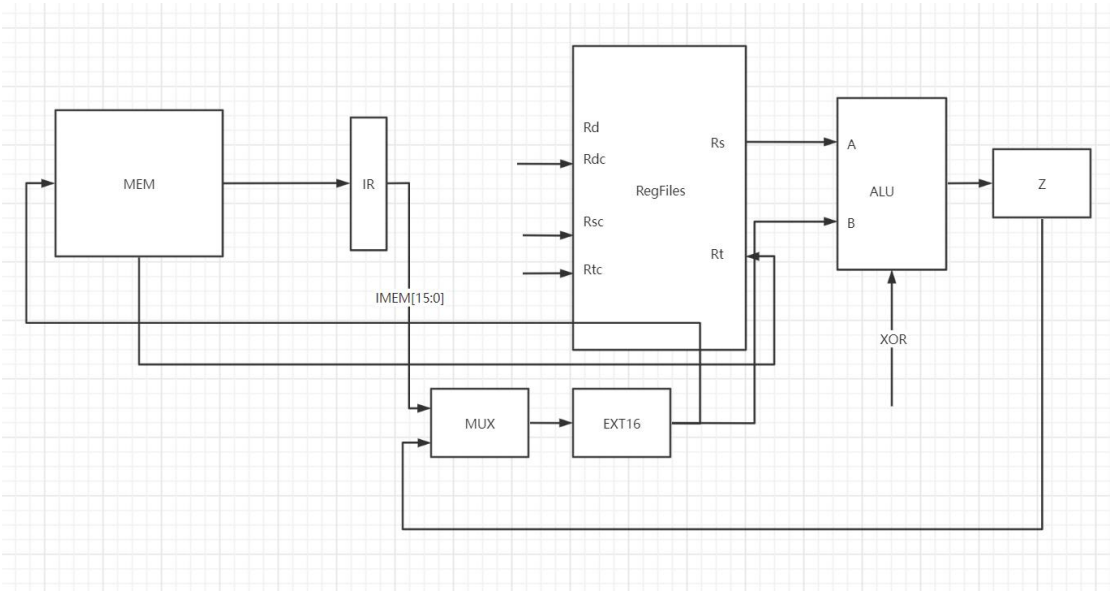
d) 使用计算好的地址访问存储器，从中读出一个 8 位的数据；

e) 把该数据 0 拓展后写入寄存器堆中的 rt 寄存器。

输入来源：

指令	PC	IR	MEM		RegFile	ALU		Z	EXT16
			Address	MD	Rt	A	B		
Fetch	Z	MD	PC			PC	4	ALU	
LBU			Z		EXT16	Rs	IR[15:0]	ALU	MD

指令通路：



38）lhu

周期数：5

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 对指令译码并读出 rs 寄存器的内容；

c) rs 寄存器的内容与指令中的偏移量 offset 相加，计算得到存储器地址；

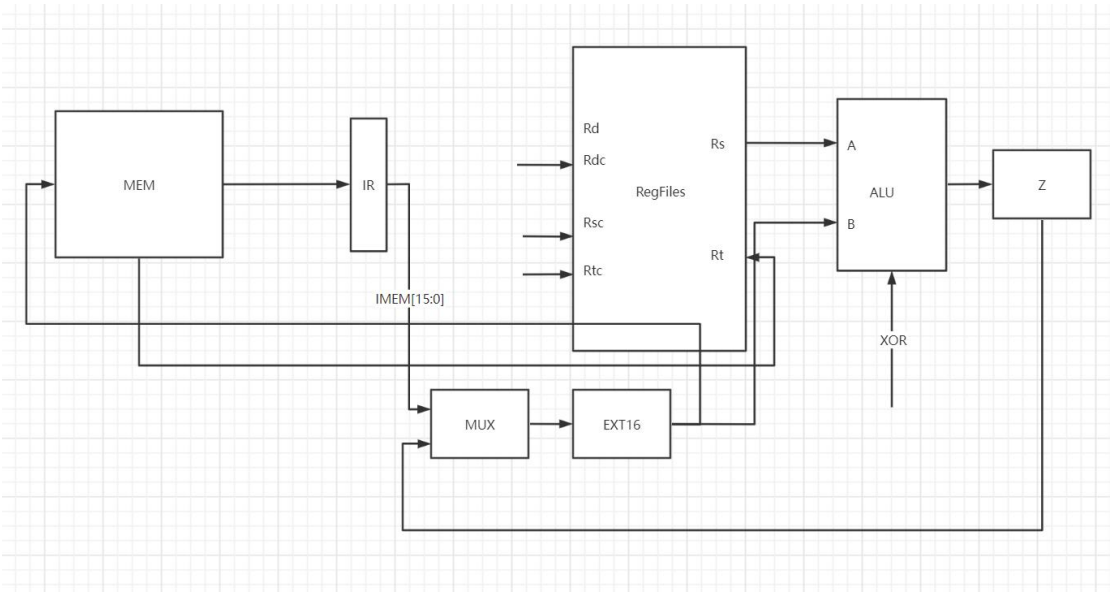
d) 使用计算好的地址访问存储器，从中读出一个 16 位的数据；

e) 把该数据 0 拓展后写入寄存器堆中的 rt 寄存器。

输入来源：

指令	PC	IR	MEM		RegFile	ALU		Z	EXT16
			Address	MD	Rt	A	B		
Fetch	Z	MD	PC			PC	4	ALU	
LHU			Z		EXT16	Rs	IR[15:0]	ALU	MD

指令通路：



39）sb

周期数：4

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 对指令译码并读出 rs,rt 寄存器的内容；

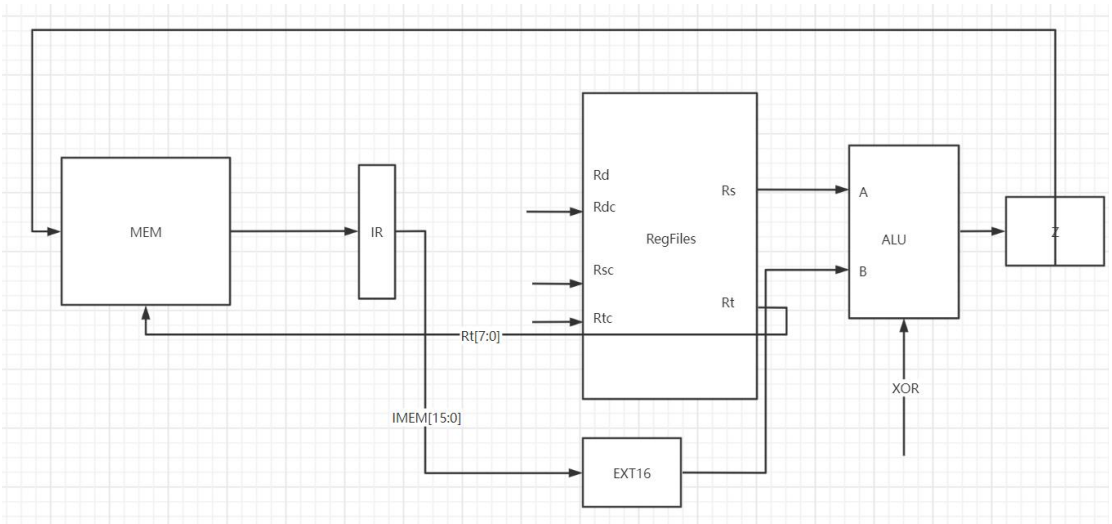
c) rs 寄存器的内容与指令中的偏移量 offset 相加，计算得到存储器地址；

d) 将 rt 低 8 位写入存储器；

输入来源：

指令	PC	IR	MEM		RegFile	ALU		Z
			Address	MD	Rt	A	B	
Fetch	Z	MD	PC			PC	4	ALU
SB			Z	Rt[7:0]		Rs	IR[15:0]	ALU

指令通路：



40）sh

周期数：4

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 对指令译码并读出 rs,rt 寄存器的内容；

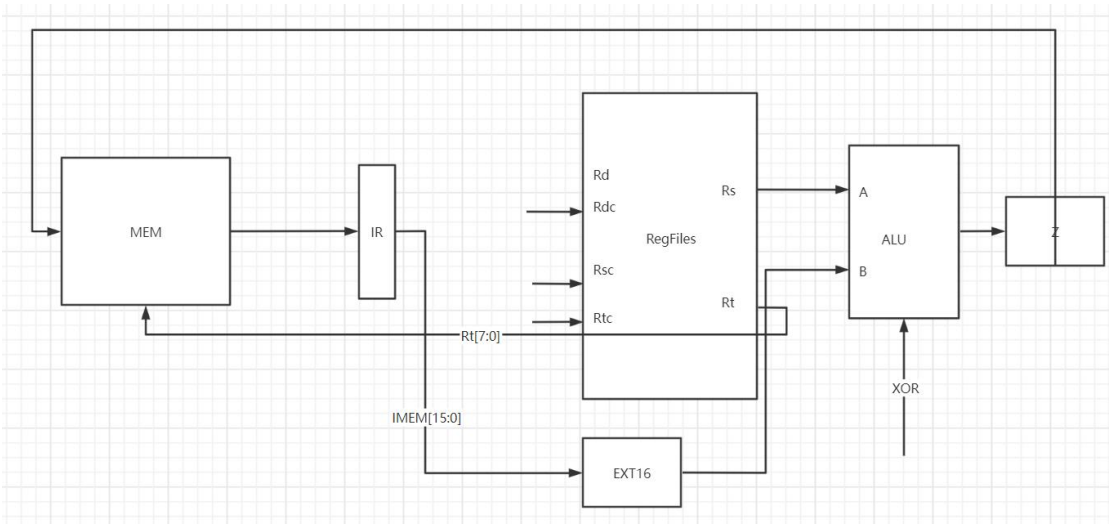
c) rs 寄存器的内容与指令中的偏移量 offset 相加，计算得到存储器地址；

d) 将 rt 低 16 位写入存储器；

输入来源：

指令	PC	IR	MEM		RegFile	ALU		Z
			Address	MD	Rt	A	B	
Fetch	Z	MD	PC			PC	4	ALU
SB			Z	Rt[7:0]		Rs	IR[15:0]	ALU

指令通路：



41）lh

周期数：5

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 对指令译码并读出 rs 寄存器的内容；

c) rs 寄存器的内容与指令中的偏移量 offset 相加，计算得到存储器地址；

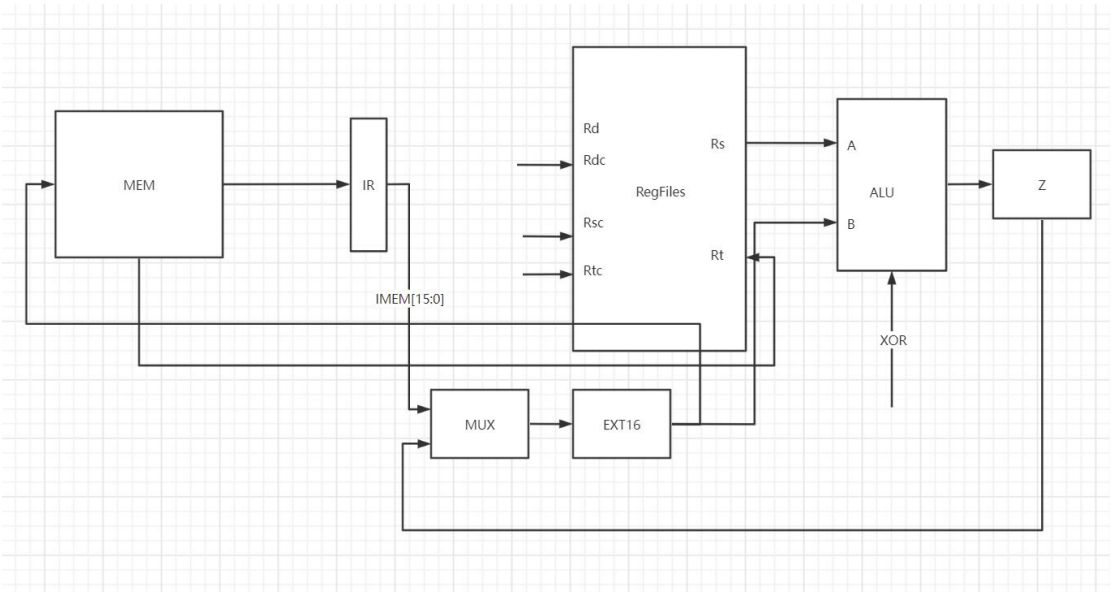
d) 使用计算好的地址访问存储器，从中读出一个 16 位的数据；

e) 把该数据符号拓展后写入寄存器堆中的 rt 寄存器。

输入来源：

指令	PC	IR	MEM		RegFile	ALU		Z	EXT16
			Address	MD	Rt	A	B		
Fetch	Z	MD	PC			PC	4	ALU	
LH			Z		EXT16	Rs	IR[15:0]	ALU	MD

指令通路：

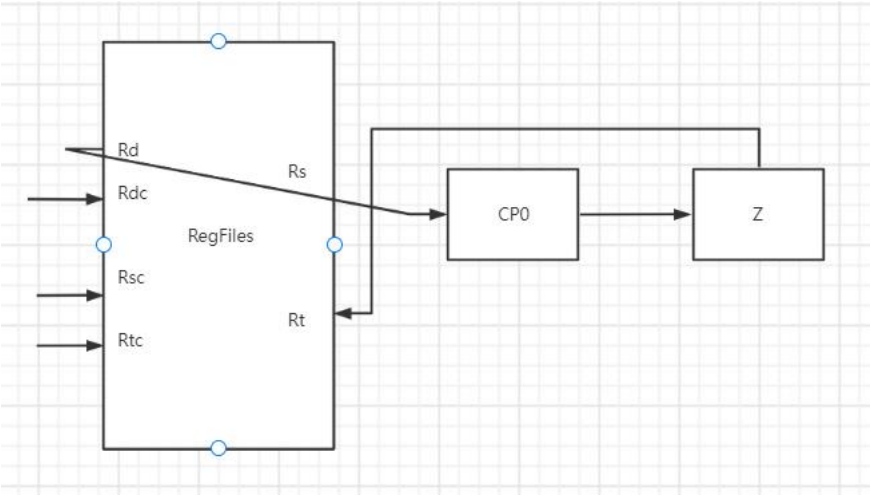


42）mfc0
周期数：2
操作流程：
a) 根据 PC 取指令，并把 PC 加 4；
b) rt<-CP0[rd]。

输入来源：

指令	PC	IR	MEM		RegFile	ALU		CP0		Z
			Address	MD	Rt	A	B	addr		
Fetch	Z	MD	PC			PC	4			ALU
MFC0					Z			rd		CP0

指令通路：



43）mfhi
周期数：2

操作流程：

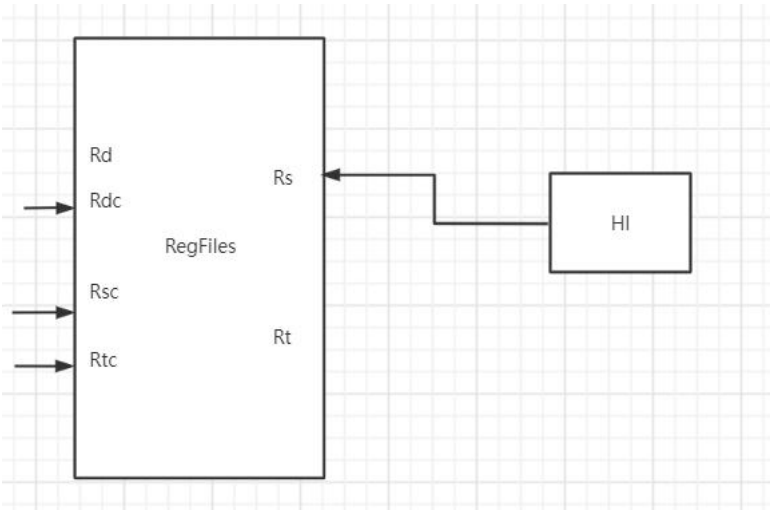
a) 根据 PC 取指令，并把 PC 加 4；

b) rs<-HI。

输入来源：

指令	PC	IR	MEM		RegFile	ALU		Z	HI
			Address	MD	Rs	A	B		
Fetch	Z	MD	PC			PC	4	ALU	
MFHI					HI				

指令通路：



44）mflo

周期数：2

操作流程：

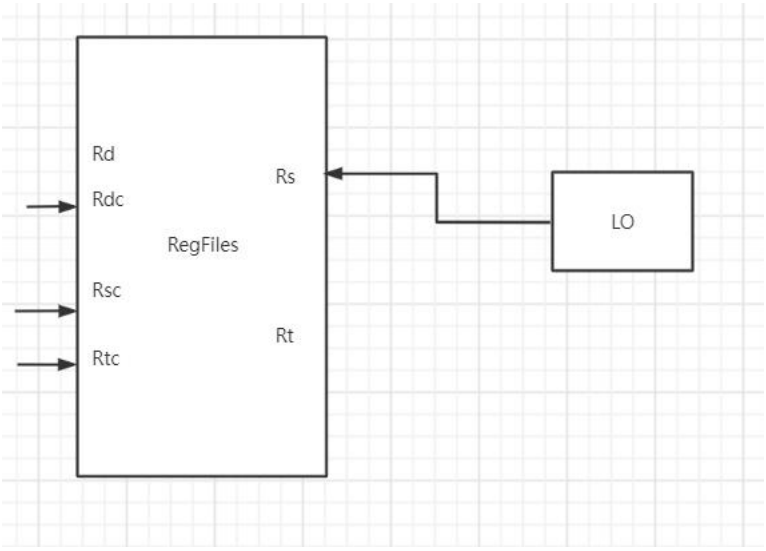
a) 根据 PC 取指令，并把 PC 加 4；

b) rs<-HI。

输入来源：

指令	PC	IR	MEM		RegFile	ALU		Z	LO
			Address	MD	Rs	A	B		
Fetch	Z	MD	PC			PC	4	ALU	
MFLO					LO				

指令通路：

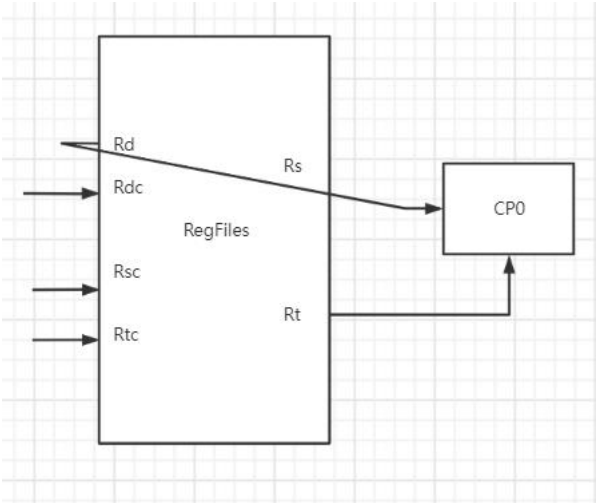


45）mtc0
周期数：3
操作流程：
a) 根据 PC 取指令，并把 PC 加 4；
b) 读取 rd，rt；
c) rt->CP0[rd]。

输入来源：

指令	PC	IR	MEM		RegFile	ALU		CP0		Z
			Address	MD	Rt	A	B	addr	data	
Fetch	Z	MD	PC			PC	4			ALU
MFC0								rd	rt	

指令通路：



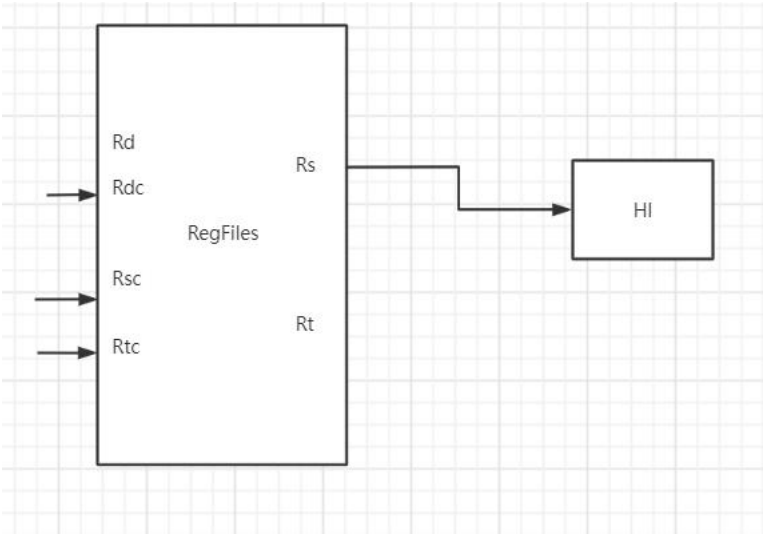
46）mthi
周期数：2
操作流程：

a) 根据 PC 取指令，并把 PC 加 4；
b) 读取 rs 赋给 HI；

输入来源：

指令	PC	IR	MEM		RegFile	ALU		Z	HI
			Address	MD	Rs	A	B		
Fetch	Z	MD	PC			PC	4	ALU	
MTHI									Rs

指令通路：

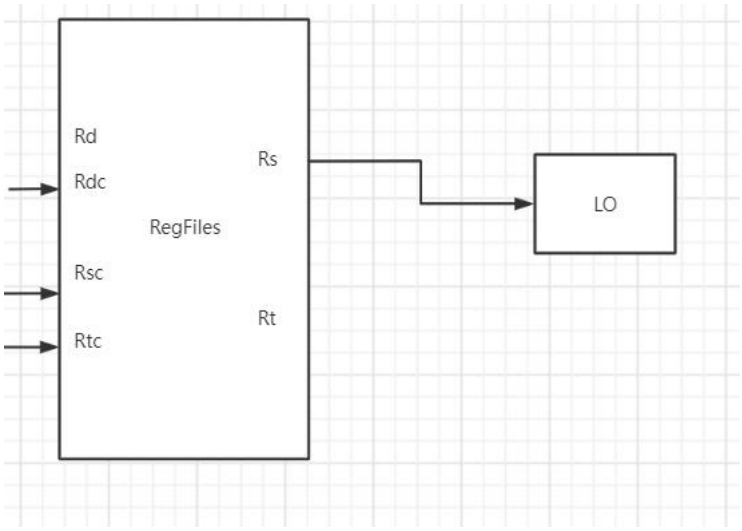


47）mtlo
周期数：2
操作流程：
a) 根据 PC 取指令，并把 PC 加 4；
b) 读取 rs 赋给 HI；

输入来源：

指令	PC	IR	MEM		RegFile	ALU		Z	LO
			Address	MD	Rs	A	B		
Fetch	Z	MD	PC			PC	4	ALU	
MTLO									Rs

指令通路：



48）mul

周期数：4

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 读出 rs,rt 寄存器的内容；

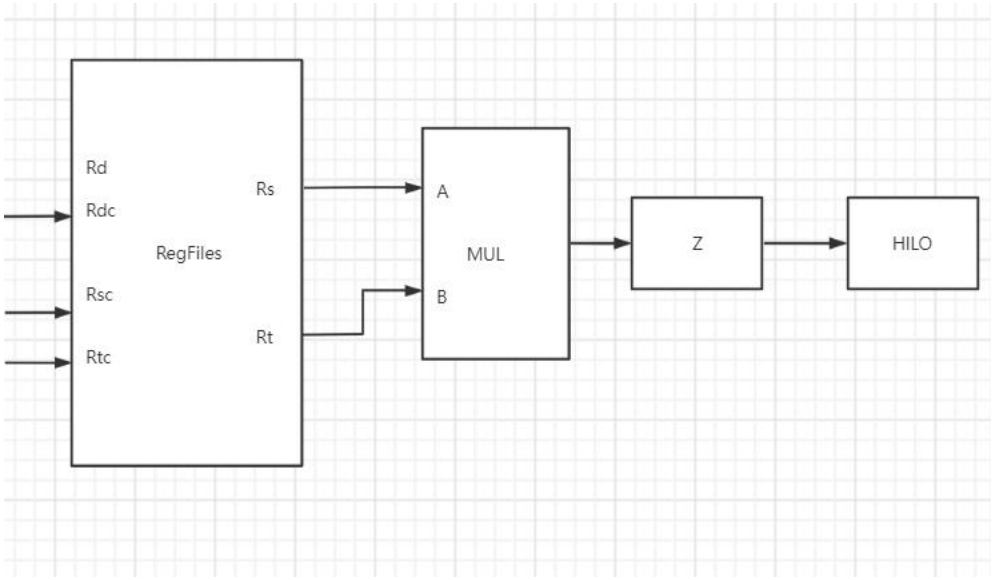
c) 由运算单元完成计算；

d) 把计算结果写入 HILO 寄存器。

输入来源：

指令	PC	IR	MEM		RegFile	ALU		MUL		Z	HILO
			Address	MD	Rd	A	B	A	B		
Fetch	Z	MD	PC			PC	4			ALU	
MUL								rs	rt	MUL	Z

指令通路：



49）multu

周期数：4

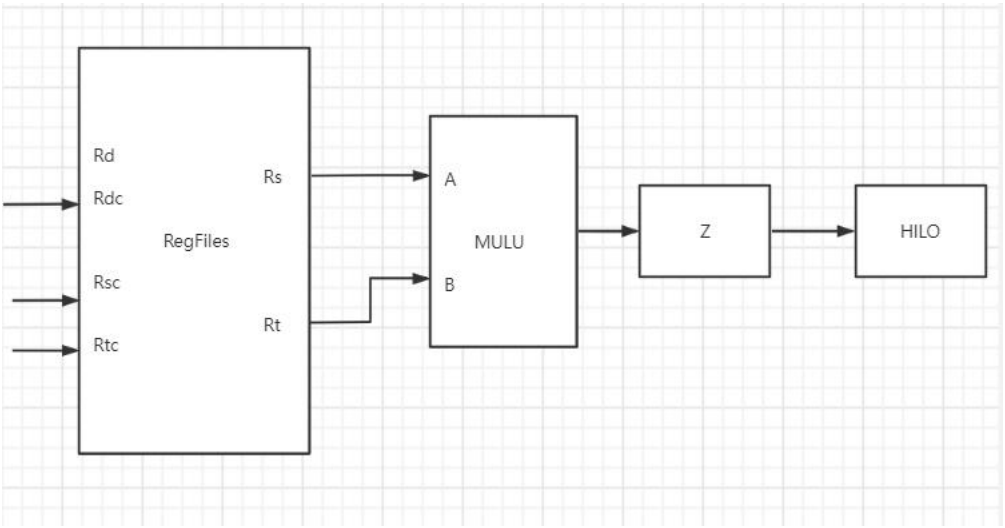
操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 读出 rs,rt 寄存器的内容；
c) 由运算单元完成计算；
d) 把计算结果写入 HILO 寄存器。
输入来源：

指令	PC	IR	MEM		RegFile	ALU		MULU		Z	HILO
			Address	MD	Rd	A	B	A	B		
Fetch	Z	MD	PC			PC	4			ALU	
MULU								rs	rt	MULU	Z

指令通路：

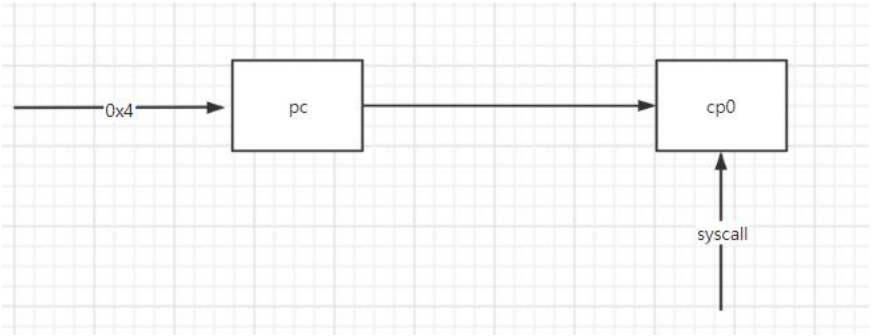


50）syscall
周期数：3
操作流程：
a) 根据 PC 取指令，并把 PC 加 4；
b) CP0 进行协处理；
c) 异常入口地址赋给 pc

输入来源：

指令	PC	IR	MEM		RegFile	ALU		CP0		Z
			Address	MD	Rt	A	B	Pc	exception	
Fetch	Z	MD	PC			PC	4			ALU
syscall	0x4							pc	syscall	

指令通路：

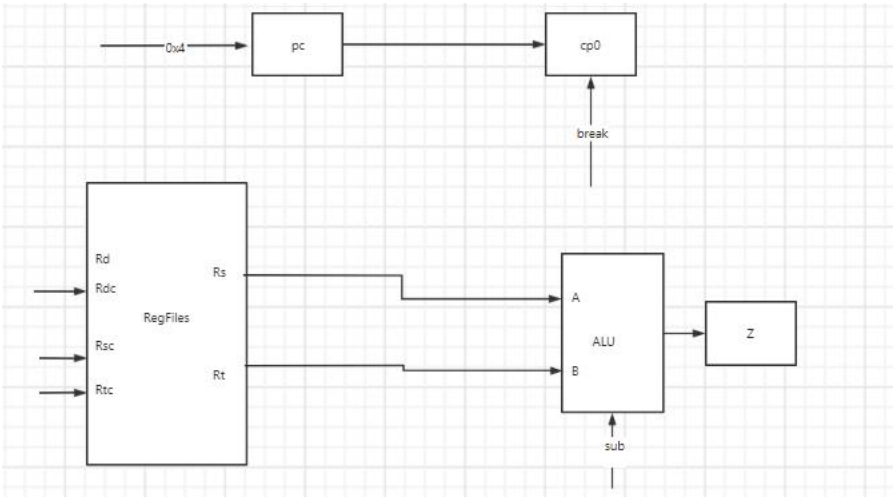


51）teq
周期数：4
操作流程：
a) 根据 PC 取指令，并把 PC 加 4；
b) rs, rt 取值；
c) 比较 rs,rt(送 ALU 作差）
d) 若相等转 cp0 处理，异常入口地址赋给 pc

输入来源：

指令	PC	IR	MEM		RegFile	ALU		CP0		Z
			Address	MD	Rt	A	B	Pc	exception	
Fetch	Z	MD	PC			PC	4			ALU
Teq	0x4					Rs	Rt	pc	syscall	ALU

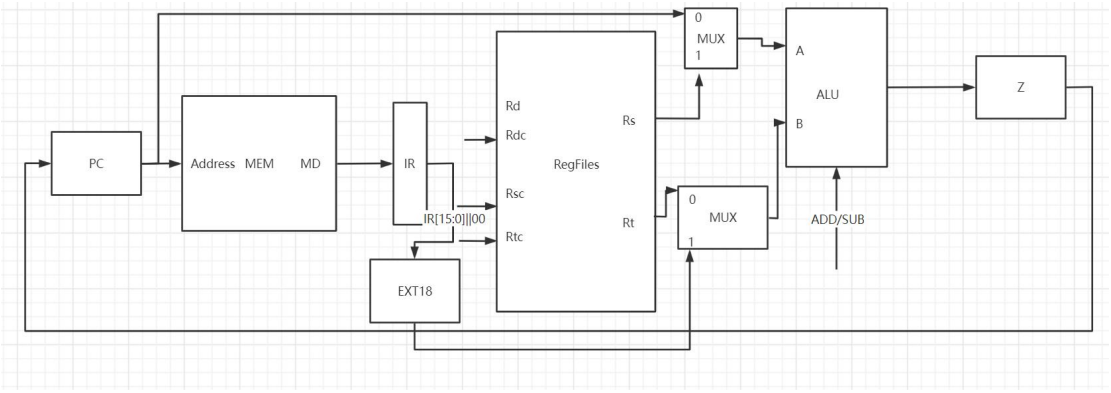
指令通路：



52）bgez
周期数：4
操作流程：
a) 根据 PC 取指令，并把 PC 加 4；
b) 读出 rs 寄存器的数据并锁存
C)同时 ALU 计算转移地址并锁存；
d) 由 ALU 比较 rs 和 0，并决定是否把转移地址写入 PC。
输入来源：

指令	PC	IR	MEM		RegFile	ALU		EXT18	Z
			Address	MD	Rd	A	B		
Fetch	Z	MD	PC			PC	4		ALU
BNE	Z					PC	EXT18	IR[15:0]||00	ALU

指令通路：



53）Break

周期数：3

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

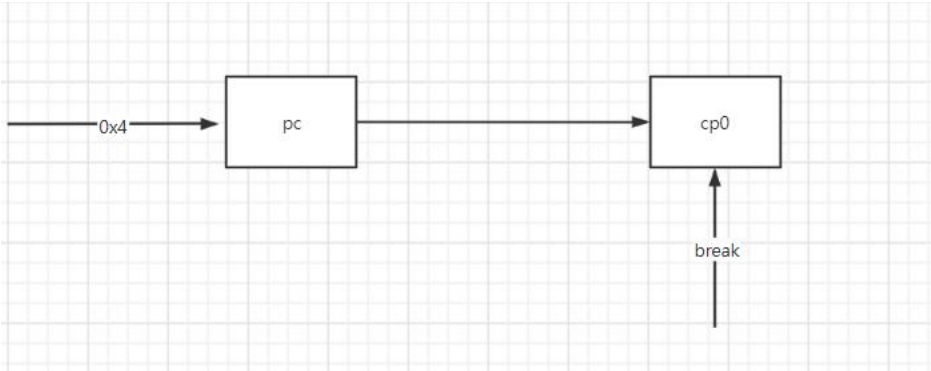
b) CP0 进行协处理；

c) 异常入口地址赋给 pc

输入来源：

指令	PC	IR	MEM		RegFile	ALU		CP0		Z
			Address	MD	Rt	A	B	Pc	exception	
Fetch	Z	MD	PC			PC	4			ALU
break	0x4							pc	break	

指令通路：



54）Div

周期数：4

操作流程：

a) 根据 PC 取指令，并把 PC 加 4；

b) 读出 rs,rt 寄存器的内容；

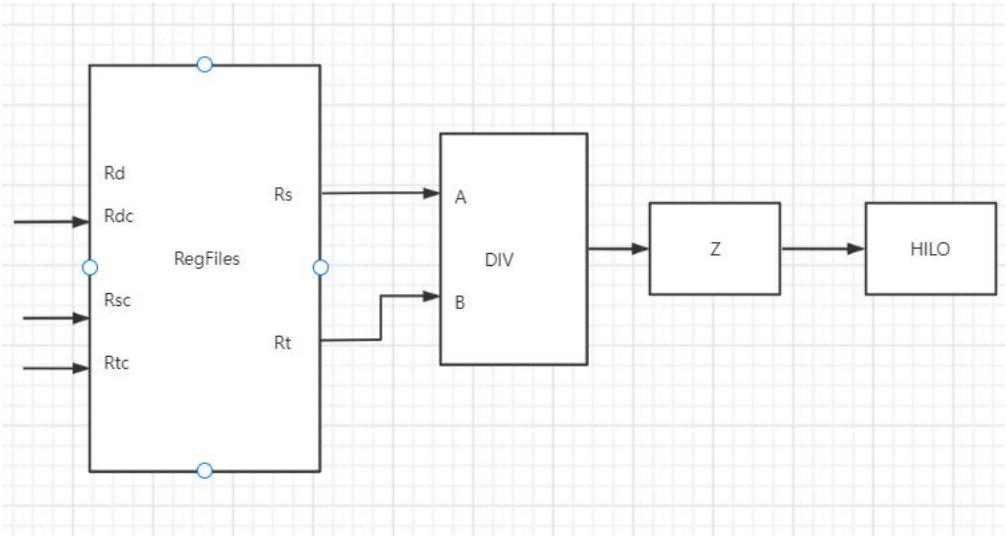
c) 由运算单元完成计算；

d) 把计算结果写入 HILO 寄存器。

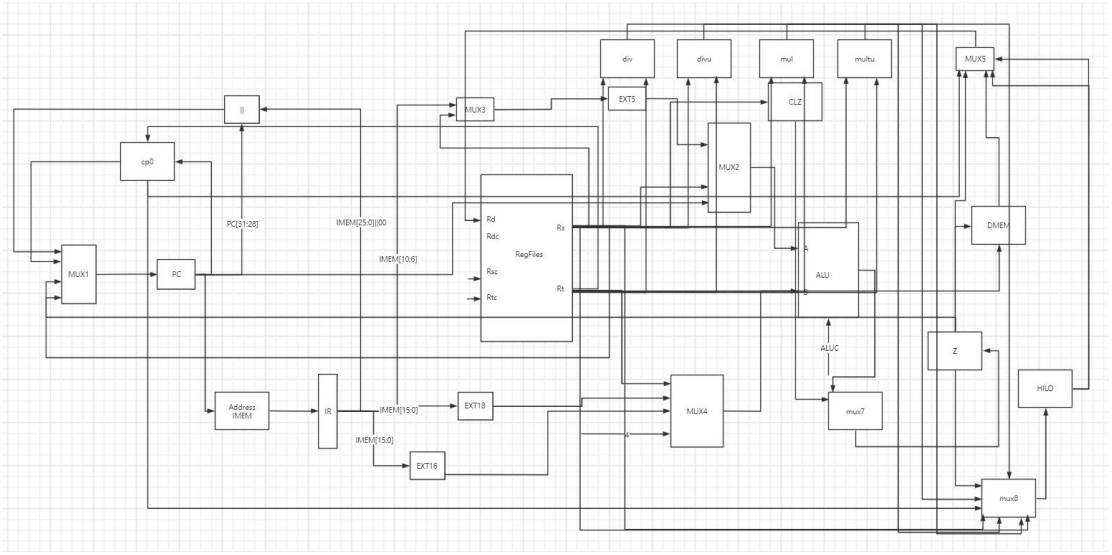
输入来源：

指令	PC	IR	MEM		RegFile	ALU		DIV		Z	HILO
			Address	MD	Rd	A	B	A	B		
Fetch	Z	MD	PC			PC	4			ALU	
DIV								rs	rt	DIV	Z

指令通路：



电路总图：



三、模块建模

本实验主要按照 cpu 的构建分成了一下几个模块：控制器，alu，存储器，以及一些存储暂存器（如 RegZ），以及一些运算部件，例如 clz、multu 等等，此外为了对于异常进行处理，还设置了 cp0，具体的模块连接逻辑可以在上面这张图上看出。

下面分别列出所有的模块以及相应的代码。

顶层模块

```
module sccomp_dataflow(
    input clk_in,
```

```
    input reset,
    output [31:0]inst,
    output [31:0]pc
    );
    reg    [31:0] tmp_data_pc[2:0];
    reg    [31:0] tmp_data_inst[2:0];
    wire [31:0] addr;
    wire [10:0] addr1;
    wire [31:0]rdata;
    wire [31:0]inst_tmp,pc_tmp;
    wire wena;
    wire rena;
    wire [31:0]wdata;

    assign addr1 = addr[10:0];
    assign pc=tmp_data_pc[0];
    assign inst=inst_tmp;

    always @(pc_tmp) begin
            tmp_data_pc[0]=pc_tmp;
    end
    cpu54 sccpu(clk_in,reset,inst_tmp,rdata,pc_tmp,addr,wdata,wena,rena);
    dmem
dmemory(.clk(clk_in),.wena(wena),.rena(rena),.addr(addr1),.data_in(wdata),.data_out(rdata));
    imem imemory(pc_tmp,inst_tmp);
endmodule
```

cpu 模块

```
module cpu54(
    input clk,
    input rst,
    input [31:0] instr,
    input [31:0] rdata,
    output [31:0] pc,
    output [31:0] waddr,
    output [31:0] wdata,
    output dm_we,
    output dm_re
    );
    wire rsPos;//rs????-

    wire [1:0]m1;//0-- || ,1--cp0,2--Z,3--Rs
```

```
wire [1:0]m2;//01--ext5,1--rs,2--pc,3--xx
wire m3;//0--ir[10:6],1--rs
wire [1:0]m4;//0--rt,1--ext18,2--ext16,3--4
wire [2:0]m5;//0--Z,1--dmem,2--hi,3--lo,4--cp0
wire [1:0]m6;//0--rd,1--rt,2--31
wire m7;//0--alu,1--clz
wire [2:0]m8;//0--Z,1--cp0,2--div,3--divu,4--mul,5--multu,6--rs
wire [2:0]m9;//0--Z,1--cp0,2--div,3--divu,4--mul,5--multu,6--rs
wire [1:0]m10;//0--lw,1--lh,2--lb
wire [1:0]m11;//0--sw,1--sh,2--sb
wire [31:0] res_m1;
wire [31:0] res_m2;
wire [4:0] res_m3;
wire [31:0] res_m4;
wire [31:0] res_m5;
wire [4:0] res_m6;
wire [31:0] res_m7;
wire [31:0] res_m8;
wire [31:0] res_m9;
wire [31:0] res_m10;
wire [31:0] res_m11;
wire div_start;
wire zin;
wire zout;
wire ir_in;
wire decoder_ena;
wire hi_ena;
wire lo_ena;
wire pc_ena;
wire cp0_ena;

wire pc_ena2;
wire [3:0]aluc;
wire rf_w;
wire rf_clk;
wire sext;

wire zero;
wire carry;
wire negative;
wire overflow;

wire [53:0]decoded_instr;
```

```
        wire [31:0] res_alu;
        wire [31:0] res_pc;
        wire [31:0] res_pc2;

        wire [31:0] rs;
        wire [31:0] rt;
        wire [31:0] rd;

        wire [31:0] res_z;
        wire [31:0] res_hi;
        wire [31:0] res_lo;
        wire [31:0] res_ext5;
        wire [31:0] res_ext16;
        wire [31:0] res_ext18;
        wire [31:0] IR;
        wire [31:0] res_ii;
        wire busy;//???busy??
        wire [31:0] res_q;//?
        wire [31:0] res_r;//??
        wire [31:0] res_qu;//?
        wire [31:0] res_ru;//??
        wire [63:0] res_mul;
        wire [63:0] res_multu;
        // wire [31:0] zaddr;
//        wire [31:0] res_z;
        wire [31:0] res_clz;
        wire [31:0] res_ext16_dm;
        wire [31:0] res_ext8_dm;
        wire [31:0] res_ext16_rt;
        wire [31:0] res_ext8_rt;
        wire [4:0] rd_addr;
        wire[4:0] rt_addr;
        assign waddr = res_z;

        assign pc = res_pc2;
        assign rd_addr=IR[15:11];
        assign rt_addr=IR[20:16];
        wire [31:0] cp0_rdata,exc_addr,cp0_status;
        wire [4:0] cp0_cause;

        wire [31:0] empty=32'bz;
        assign rsPos=!rs[31];
        control_unit cu(.clk(clk),.rst(rst),.z(zero),.rsPos(rsPos),.instr(decoded_instr),
        .mux1(m1),.mux2(m2),.mux3(m3),.mux4(m4),.mux5(m5),.mux6(m6),.mux7(m7),.mux8(m8),
```

```
.mux9(m9),.mux10(m10),.mux11(m11),
    .div_start(div_start),.zin(zin),.zout(zout),.ir_in(ir_in),.aluc(aluc),.rf_w(rf_w),.dm_we(dm_we)
,
    .dm_re(dm_re),.hi_ena(hi_ena),.lo_ena(lo_ena),.decoder_ena(decoder_ena),.sext(sext),.cause
(cp0_cause),.pc_ena(pc_ena),.pc_ena2(pc_ena2),.cp0_ena(cp0_ena));

    IR irreg(.instr(instr),.ir_in(ir_in),.ir(IR));

    instr_decoder id(.instr_code(IR),.decoder_ena(decoder_ena),.i(decoded_instr));

    pc1 mypc(.rst(rst),.we(pc_ena),.data_in(res_m1),.data_out(res_pc));

    pc1 realpc(.rst(rst),.we(pc_ena2),.data_in(res_pc),.data_out(res_pc2));

    II cpu_ii(res_pc[31:28],IR[25:0],res_ii);

    alu myalu(res_m2,res_m4,aluc,res_alu,zero,carry,negative,overflow);

    RF
cpu_ref(.rst(rst),.we(rf_w),.raddr1(IR[25:21]),.raddr2(IR[20:16]),.waddr(res_m6),.wdata(res_m5),.
rdata1(rs),.rdata2(rt));

    cp0
my_cp0(.clk(clk),.rst(rst),.ena(cp0_ena),.mfc0(decoded_instr[44]),.mtc0(decoded_instr[45]),.pc(re
s_pc),.cp0_addr(rd_addr),
    .wdata(rt),.exception(decoded_instr[53]||decoded_instr[52]||decoded_instr[51]),.eret(decoded
_instr[50]),.cause(cp0_cause),.rdata(cp0_rdata),.status(cp0_status),.exc_addr(exc_addr));

    RegZ regZ(.rst(rst),.zin(zin),.zout(zout),.wdata(res_m7),.rdata(res_z));

    HILO hi(.wena(hi_ena),.rst(rst),.wdata(res_m8),.rdata(res_hi));
    HILO lo(.wena(lo_ena),.rst(rst),.wdata(res_m9),.rdata(res_lo));

    DIVU
divu(.dividend(rs),.divisor(rt),.start(div_start),.clock(clk),.reset(rst),.q(res_qu),.r(res_ru),.busy(bus
y));
    DIV                                                                                    div
(.dividend(rs),.divisor(rt),.start(div_start),.clock(clk),.reset(rst),.q(res_q),.r(res_r),.busy(busy));
    MULT    mult (.clk(clk),.reset(rst),.a(rs),.b(rt), .z(res_mul));
    MULTU multu(.clk(clk),.reset(rst),.a(rs),.b(rt), .z(res_multu));
    clz CLZ(.num(rs),.result(res_clz));

    mux4 #(32) selector1(.a(res_ii),.b(exc_addr),.c(res_z),.d(rs),.choose(m1),.res(res_m1));
    mux4 #(32) selector2(.a(res_ext5),.b(rs),.c(res_pc),.d(empty),.choose(m2),.res(res_m2));
```

```
    mux    #(5) selector3(.a(IR[10:6]),.b(rs[4:0]),.choose(m3),.res(res_m3));
    mux4 #(32) selector4(.a(rt),.b(res_ext18),.c(res_ext16),.d(32'd4),.choose(m4),.res(res_m4));
    mux8                                                                    #(32)
selector5(.a(res_z),.b(res_m10),.c(res_hi),.d(res_lo),.e(cp0_rdata),.f(res_mul[31:0]),.g(empty),.h(e
mpty),.choose(m5),.res(res_m5));//0--Z,1--dmem,2--hi,3--lo,4--cp0
//0--rd,1--rt,2--31
    mux4 #(5) selector6(.a(rd_addr),.b(rt_addr),.c(5'd31),.d(5'bz),.choose(m6),.res(res_m6));
//0--alu,1--clz
    mux    #(32) selector7(.a(res_alu),.b(res_clz),.choose(m7),.res(res_m7));
//0--Z,1--cp0,2--div,3--divu,4--mul,5--multu,6--rs      hi //Òª , Ä!!!!!!!!!!!!
    mux8                                                                    #(32)
selector8(.a(res_z),.b(cp0_rdata),.c(res_r),.d(res_ru),.e(empty),.f(res_multu[63:32]),.g(rs),.h(empt
y),.choose(m8),.res(res_m8));
    mux8                                                                    #(32)
selector9(.a(res_z),.b(cp0_rdata),.c(res_q),.d(res_qu),.e(empty),.f(res_multu[31:0]),.g(rs),.h(empty)
,.choose(m9),.res(res_m9));
/*      wire [1:0]m10;//0--lw,1--lh,2--lb
    wire [1:0]m11;//0--sw,1--sh,2--sb*/

    mux4                                                                    #(32)
selector10(.a(rdata),.b(res_ext16_dm),.c(res_ext8_dm),.d(32'bz),.choose(m10),.res(res_m10));
    mux4                                                                    #(32)
selector11(.a(rt),.b(res_ext16_rt),.c(res_ext8_rt),.d(32'bz),.choose(m11),.res(wdata));

    EXT #(5)     ext5 (.numb(res_m3),.isSign(sext),.res(res_ext5));
    EXT #(16)    ext16(.numb(IR[15:0]),.isSign(sext),.res(res_ext16));
    EXT #(18)    ext18(.numb({IR[15:0],2'b00}),.isSign(sext),.res(res_ext18));

    EXT #(16)    ext16_dm(.numb(rdata[15:0]),.isSign(sext),.res(res_ext16_dm));
    EXT #(8)     ext8_dm(.numb(rdata[7:0]),.isSign(sext),.res(res_ext8_dm));

    EXT #(16)    ext16_rt(.numb(rt[15:0]),.isSign(sext),.res(res_ext16_rt));
    EXT #(8)     ext8_rt(.numb(rt[7:0]),.isSign(sext),.res(res_ext8_rt));

endmodule
```

译码器模块

```
module instr_decoder(
    input [31:0]instr_code,
    input decoder_ena,
    output reg [53:0] i

    );
    wire [11:0] tmp;
```

```
    assign tmp = {instr_code[31:26],instr_code[5:0]};
    always @ (*)
    begin
        if (decoder_ena)begin

            casez(tmp)
                12'b000000100000                :i                  =
54'b000000000000000000000000000000000000000000000000000001;//add 0
                12'b000000100001                :i                  =
54'b000000000000000000000000000000000000000000000000000010;//addu 1
                12'b000000100010                :i                  =
54'b000000000000000000000000000000000000000000000000000100;//sub 2
                12'b000000100011                :i                  =
54'b000000000000000000000000000000000000000000000000001000;//subu 3
                12'b000000100100                :i                  =
54'b000000000000000000000000000000000000000000000000010000;//and 4
                12'b000000100101                :i                  =
54'b000000000000000000000000000000000000000000000000100000;//or 5
                12'b000000100110                :i                  =
54'b000000000000000000000000000000000000000000000001000000;//xor 6
                12'b000000100111                :i                  =
54'b000000000000000000000000000000000000000000000010000000;//nor 7
                12'b000000101010                :i                  =
54'b000000000000000000000000000000000000000000000100000000;//slt 8    //rd â†?(rs < rt)
                12'b000000101011                :i                  =
54'b000000000000000000000000000000000000000000001000000000;//sltu 9
                12'b000000000000                :i                  =
54'b000000000000000000000000000000000000000000010000000000;//sll 10    //rd â†? rt << sa
                12'b000000000010                :i                  =
54'b000000000000000000000000000000000000000000100000000000;//srl 11
                12'b000000000011                :i                  =
54'b000000000000000000000000000000000000000001000000000000;//sra 12
                12'b000000000100                :i                  =
54'b000000000000000000000000000000000000000010000000000000;//sllv 13 //rd â†? rt << rs
                12'b000000000110                :i                  =
54'b000000000000000000000000000000000000000100000000000000;//srlv 14
                12'b000000000111                :i                  =
54'b000000000000000000000000000000000000001000000000000000;//srav 15
                12'b000000001000                :i                  =
54'b000000000000000000000000000000000000010000000000000000;//jr 16
                12'b001000??????                :i                  =
54'b000000000000000000000000000000000000100000000000000000;//addi 17
                12'b001001??????                :i                  =
54'b000000000000000000000000000000000001000000000000000000;//addiu 18
```

12'b001100?????? :i =
54'b000000000000000000000000000000000100000000000000000;//andi 19
12'b001101?????? :i =
54'b000000000000000000000000000000001000000000000000000;//ori 20
12'b001110?????? :i =
54'b000000000000000000000000000000001000000000000000000;//xori 21
12'b100011?????? :i =
54'b000000000000000000000000000000001000000000000000000;//lw 22
12'b101011?????? :i =
54'b000000000000000000000000000000001000000000000000000;//sw 23
12'b000100?????? :i =
54'b000000000000000000000000000000001000000000000000000;//beq 24
12'b000101?????? :i =
54'b000000000000000000000000000000001000000000000000000;//bne 25
12'b001010?????? :i =
54'b000000000000000000000000000000001000000000000000000;//slti 26
12'b001011?????? :i =
54'b000000000000000000000000000000001000000000000000000;//sltiu 27
12'b001111?????? :i =
54'b000000000000000000000000000000001000000000000000000;//lui 28
12'b000010?????? :i =
54'b000000000000000000000000000000001000000000000000000;//j 29
12'b000011?????? :i =
54'b000000000000000000000000000000001000000000000000000;//jal 30
12'b011100100000 :i =
54'b000000000000000000000000000000001000000000000000000;//clz 31
12'b000000011011 :i =
54'b000000000000000000000000000000001000000000000000000;//divu 32
12'b000000011010 :i =
54'b000000000000000000000000000000001000000000000000000;//div 33
12'b011100000010 :i =
54'b000000000000000000000000000000001000000000000000000;//mul 34
12'b000000011001 :i =
54'b000000000000000000000000000000001000000000000000000;//mulu 35
12'b000000001001 :i =
54'b000000000000000000000000000000001000000000000000000;//jalr 36
12'b000001?????? :i =
54'b000000000000000000000000000000001000000000000000000;//bgez 37
12'b100001?????? :i =
54'b000000000000000000000000000000001000000000000000000;//lh 38
12'b100000?????? :i =
54'b000000000000000000000000000000001000000000000000000;//lb 39
12'b100100?????? :i =
54'b000000000000000000000000000000001000000000000000000;//lbu 40

```
                12'b100101??????                    :i                  =
54'b000000000000100000000000000000000000000000000000000000;//lhu 41
                12'b101000??????                    :i                  =
54'b000000000001000000000000000000000000000000000000000000;//sb 42
                12'b101001??????                    :i                  =
54'b000000000001000000000000000000000000000000000000000000;//sh 43
                //              12'b010000000000        :i              =
54'b000000000100000000000000000000000000000000000000000000;//mfc0 44 cp0->rf
                //              12'b010000000000        :i              =
54'b000000001000000000000000000000000000000000000000000000;//mtc0 45
                12'b000000010000                    :i                  =
54'b000000010000000000000000000000000000000000000000000000;//mfhi 46 rf->hi
                12'b000000010001                    :i                  =
54'b000000100000000000000000000000000000000000000000000000;//mthi 47
                12'b000000010010                    :i                  =
54'b000001000000000000000000000000000000000000000000000000;//mflo 48
                12'b000000010011                    :i                  =
54'b000010000000000000000000000000000000000000000000000000;//mtlo 49
                12'b010000011000                    :i                  =
54'b000100000000000000000000000000000000000000000000000000;//eret 50 cp0->pc
                12'b000000001100                    :i                  =
54'b001000000000000000000000000000000000000000000000000000;//syscall 51
                12'b000000110100                    :i                  =
54'b010000000000000000000000000000000000000000000000000000;//teq 52
                12'b000000001101                    :i                  =
54'b100000000000000000000000000000000000000000000000000000;//break 53
                default:    begin
                    if (instr_code[31:21]==11'b01000000000)begin
                        i                                               =
54'b000000000100000000000000000000000000000000000000000000;//mfc0 44
                    end
                    else if(instr_code[31:21]==11'b01000000100)begin
                        i                                               =
54'b000000001000000000000000000000000000000000000000000000;//mtc0 45
                    end
                    else begin
                        i = 54'bx;
                    end
                end


            endcase

        end
```

```
    end
endmodule
```

控制器模块

```
module control_unit(
    input clk,
    input rst,
    input z,
    input rsPos,//rs 是否非-
    input [53:0]instr,//译码之后的指令
    output reg [1:0]mux1,//0-- || ,1--cp0,2--Z,3--Rs
    output reg [1:0]mux2,//01--ext5,1--rs,2--pc,3--xx
    output reg mux3,//0--ir[10:6],1--rs
    output reg [1:0]mux4,//0--rt,1--ext18,2--ext16,3--4
    output reg [2:0]mux5,//0--Z,1--dmem,2--hi,3--lo,4--cp0
    output reg [1:0]mux6,//0--rd,1--rt,2--31
    output reg mux7,//0--alu,1--clz
    output reg [2:0]mux8,//0--Z,1--cp0,2--div,3--divu,4--mul,5--multu,6--rs//hi
    output reg [2:0]mux9,//0--Z,1--cp0,2--div,3--divu,4--mul,5--multu,6--rs//lo
    output reg [1:0]mux10,//0--lw,1--lh,2--lb
    output reg [1:0]mux11,//0--sw,1--sh,2--sb
    output reg div_start,
    output reg zin,
    output reg zout,
    output reg ir_in,
    output reg [3:0]aluc,
    output reg rf_w,
    output reg dm_we,
    output reg dm_re,
    output reg hi_ena,
    output reg lo_ena,
    output reg decoder_ena,
    output sext,
    output [4:0]cause,
    output reg pc_ena,
    output reg pc_ena2,
    output reg cp0_ena
    );
    reg [2:0]state;
    parameter [4:0]C_SYS = 5'b01000, C_BREAK = 5'b01001, C_TEQ = 5'b01101, C_ERET =
5'b00000;
    assign cause=instr[51]? C_SYS : (instr[53] ? C_BREAK :(instr[52] ? C_TEQ : (instr[50] ?
C_ERET :5'bz)));
```

```
    assign sext = instr[17] | instr[18]  | instr[28] | instr[27] | instr[22] | instr[23] | instr[24] |
instr[25] | instr[26] | instr[38] | instr[39];
    parameter sif = 3'b000,sid = 3'b001,sexe = 3'b010,smem = 3'b011,swb = 3'b100;
    // assign decoder_ena = (state==sif && !rst)?1:0;
    always@(posedge clk or posedge rst)begin
        if(rst)begin
            state =sif;
            rf_w=0;
            zin=0;
            zout=0;
            div_start=0;
            ir_in=1;
            dm_we=0;
            dm_re=0;
            hi_ena=0;
            lo_ena=0;
            pc_ena=0;
            pc_ena2=1;
            cp0_ena=0;
        end
        else if(state==sif)begin
            //pc 取址译码,pc+4
            pc_ena=0;
            pc_ena2=1;
            rf_w=0;
            zout=0;
            div_start=0;
            dm_we=0;
            dm_re=0;//读指令
            hi_ena=0;
            lo_ena=0;
            aluc=4'b0010;
            mux2=2'd2;
            mux4=2'd3;
            zin=1;
            ir_in=1;
            mux7=0;//alu->z
            // mux5=3'bx;
            // zout=1;
            decoder_ena=1;
            cp0_ena=0;
            // if(instr[29]||instr[16])begin
            //      state=swb;
            // end
```

```
        // else begin
        //         state=sid;//状态转移
        // end
        state=sid;//状态转移
    end
    else if(state==sid)begin
        //pc+4 写回 pc,取寄存器地址
        decoder_ena=0;
        rf_w=0;//锁存
        ir_in=0;

        pc_ena2=0;
        pc_ena=1;

        // zout=1;
        //-----寄存器取址------
        if (instr[30])begin
            mux6=2'd2;
        end
        else if (instr[28:17] || instr[45:37] || instr[50])
            mux6=2'd1;    // I rt[20:16]
        else
            mux6=2'd0;//R rd[15:11]
        //--------------------



        //-----z->pc------
        zout=1;
        zin=0;
        mux1=2'd2;

        //----------------

        if(instr[15:0] || instr[28:17] || instr[43:31] || instr[52]||instr[30])begin//4+5
            state=sexe;
        end
        else begin//eret,syscall,break,clz,mf*,mt*//3
            cp0_ena=0;
            state=swb;//异常地址去写回 pc
        end

    end
    else if(state==sexe)begin
```

```
//执行指令
if (!instr[30]&&!instr[36])begin
        zin=1;
        zout=0;
end

pc_ena=0;
//------状态转移------
if(instr[22] || instr[23] || instr[43:38]||instr[25:24])begin//l*,s*+bne,beq
        state=smem;
end
else begin
        state=swb;
end
//--------------------
if(instr[9:0]||instr[25:24]||instr[52])begin//alu 运算指令+bne,beq
        mux2=2'd1;//rs
        mux4=2'd0;//rt
        // if(!instr[25:24])//bne,beq 不要写进暂存器，只需要看 zero
        mux7=0;//alu-->z
end
else if(instr[15:10])begin//sll,sra,srl
        if (instr[15:13])begin
                mux3=1;
        end
        else
                mux3=0;//sa
        mux2=0;//ext5
        mux4=2'd0;//rt
        mux7=0;//alu-->z
end
else if(instr[28:26]||instr[23:17]||instr[43:38])begin//立即数计算指令+l*+s*,
        mux2=2'd1;//rs
        mux4=2'd2;//ext16
        mux7=0;//alu-->z
end
else if(instr[30]||instr[36])begin//jal,jalr
        // mux2=2'd2;//pc
        // mux4=2'd3;//4
        // mux7=0;//alu-->z
        zin=0;
        zout=1;
        rf_w=1;
        mux5=0;
```

```
                end
                else if(instr[37])begin
                    zin=1;
                    zout=0;
                    mux7=0;//alu-->z
                    mux2=2'd2;//pc
                    mux4=2'd1;//ext18
                end
                else if(instr[33:32])begin//divu,div
                    div_start=1;
                end
                else if(instr[31])begin
                    zin=1;
                    zout=0;
                    mux7=1;//clz-->z
                end
                aluc[3] = instr[8] || instr[9] || instr[10] || instr[11] || instr[12] || instr[13] || instr[14] ||
instr[15] || instr[26] || instr[27] || instr[28];
                aluc[2] = instr[4] || instr[5] || instr[6] || instr[7] || instr[10] || instr[11] || instr[12] ||
instr[13] || instr[14] || instr[15] || instr[19] || instr[20] || instr[21];
                aluc[1] = instr[0] || instr[2] || instr[6] || instr[7] || instr[8] || instr[9] || instr[10] ||
instr[13] || instr[17] || instr[21] || instr[24] || instr[25] || instr[26] || instr[27] || instr[52] ;
                aluc[0] = instr[2] || instr[3] || instr[5] || instr[7] || instr[8] || instr[11] || instr[14] ||
instr[20] || instr[24] || instr[25] || instr[26] || instr[52];
            end
            else if(state==smem)begin
                //访问存储器+bne,beq
                if(!instr[25:24])begin
                    zout=1;//z 存储的是访问存储器的地址
                    zin=0;
                end
                if(instr[42] || instr[43] || instr[23])begin//s*
                    state=sif;
                    dm_we=1;//可写
                end
                else if(!instr[25:24])begin//l*
                    state=swb;
                    mux5=1;
                    dm_re=1;//可读
                end
                //--------X->dmem----------
                if(instr[23])begin//sw
                    mux11=0;
                end
```

```
        else if(instr[43])begin//sh
            mux11=1;
        end
        else if(instr[42])begin//sb
            mux11=2;
        end
        else if(instr[22])begin//lw
            mux10=0;
        end
        else if(instr[38]||instr[41])begin//lh
            mux10=1;
        end
        else if(instr[39]||instr[40])begin//lb
            mux10=2;
        end
        //-----------------------
        //----------------
        if((instr[25] && !z) || (instr[24] && z))begin
            aluc=4'b0010;
            mux2=2'd2;//pc
            mux4=2'd1;//ext18
            zin=1;
            zout=0;
            mux7=0;//alu-->z
            state=swb;
        end
        else if(instr[25:24])begin

            state=sif;
        end

    end
    else if(state==swb)begin
        //结果写回 regfiles
        state=sif;
        zout=1;
        zin=0;
        pc_ena2=0;
        // decoder_ena=1;
        // ir_in=1;
        // pc_ena2=1;
        //--------X->pc----------
        if (instr[29] || instr[30])begin//j 或 jal 指令
            //|| ->pc
```

```
        mux1=2'd0;
        pc_ena=1;
end
else if(instr[16]||instr[36])begin//jr 指令
        mux1=2'd3;//rs->pc
        pc_ena=1;
end
else if(instr[53:50])begin//eret 等异常处理指令
        cp0_ena=1;
        mux1=2'd1;//cp0->pc
        pc_ena=1;
end
else if(instr[25:24]||(instr[37] && rsPos))begin//beq,bne,begz
        mux1=2'd2;//Z->pc
        pc_ena=1;
end
//-----------------------

//--------X->rf----------//0--Z,1--dmem,2--hi,3--lo,4--cp0
if(instr[15:0]||instr[28:26]||instr[21:17]||instr[31])begin//计算指令+jal+jalr+clz
        //z->rd
        rf_w=1;
        mux5=0;//z->rd
end
else if(instr[41:38]||instr[22])begin//l*
        //dmem->rd
        rf_w=1;
        mux5=1;//dmem->rd
        if(instr[22])begin
                mux10=0;
        end
        else if(instr[41]||instr[38])begin//lh,lhu
                mux10=1;
        end
        else if(instr[40:39])begin//lb,lbu
                mux10=2;
        end
end
else if(instr[46])begin//mfhi
        rf_w=1;
        mux5=2;//hi->rd
end
else if(instr[48])begin//mflo
        rf_w=1;
```

```
            mux5=3;//lo->rd
        end
        else if(instr[44])begin//mfc0
            rf_w=1;
            mux5=4;//cp0->rf
        end
        else if(instr[34])begin
            rf_w=1;
            mux5=5;
        end
        //-----------------------




//--------X->hilo----------//0--Z,1--cp0,2--div,3--divu,4--mul,5--multu,6--rs-->hi,6--rs->lo
        if(instr[32])begin//divu
            mux8=3;
            mux9=3;
            hi_ena=1;
            lo_ena=1;
        end
        else if(instr[33])begin//div
            mux8=2;
            mux9=2;
            hi_ena=1;
            lo_ena=1;
        end
        // else if(instr[34])begin//mul
        //         mux8=4;
        //         mux9=4;
        //         hi_ena=1;
        //         lo_ena=1;
        // end
        else if(instr[35])begin//multu
            mux8=5;
            mux9=5;
            hi_ena=1;
            lo_ena=1;
        end
        else if(instr[47])begin//mthi
            mux8=6;
            hi_ena=1;
        end
```

```
                else if(instr[49])begin//mtlo
                        mux9=6;
                        lo_ena=1;
                end
                //div,divu,mul,multu
                //-----------------------

                //--------X->cp0----------
                if(instr[45])begin//mtc0

                end
            end
        end


endmodule
```

Regfiles 模块

```
module RF(
    input clk,
    input rst,
    input we,
    input [4:0] raddr1,
    input [4:0] raddr2,
    input [4:0] waddr,
    input [31:0] wdata,
//     output reg [31:0] rdata1,
//     output reg [31:0] rdata2
    output[31:0] rdata1,
    output[31:0] rdata2
     );
     reg [31:0] array_reg[31:0];
     integer i;
     assign rdata1=(raddr1==0)?0:array_reg[raddr1];
     assign rdata2=(raddr2==0)?0:array_reg[raddr2];
     always @(posedge clk or posedge rst) begin
            if (rst)
            begin
                for(i=0;i<32;i=i+1)
                    array_reg[i] <= 0;
            end
            else
```

```
        begin
            if(we && waddr!=0)begin
                array_reg[waddr]=wdata;
            end
        end
    end
endmodule
```

多路选择器模块

```
module mux # (parameter WIDTH=32)(
    input [WIDTH-1:0]a,
    input [WIDTH-1:0]b,
    input choose,
    output reg [WIDTH-1:0]res

    );
    always@(*)
    begin
        case(choose)
            1'b0: res=a;
            1'b1: res=b;
        endcase
    end

endmodule

module mux4 # (parameter WIDTH=32)(
    input [WIDTH-1:0]a,
    input [WIDTH-1:0]b,
    input [WIDTH-1:0]c,
    input [WIDTH-1:0]d,
    input [1:0]choose,
    output reg [WIDTH-1:0]res

    );
    always@(*)
    begin
        case(choose)
            2'd0: res=a;
            2'd1: res=b;
            2'd2: res=c;
            2'd3: res=d;
```

```
            endcase
        end

endmodule

module mux8 # (parameter WIDTH=32)(
        input [WIDTH-1:0]a,
        input [WIDTH-1:0]b,
        input [WIDTH-1:0]c,
        input [WIDTH-1:0]d,
        input [WIDTH-1:0]e,
        input [WIDTH-1:0]f,
        input [WIDTH-1:0]g,
        input [WIDTH-1:0]h,
        input [2:0]choose,
        output reg [WIDTH-1:0]res

        );
        always@(*)
        begin
            case(choose)
                3'd0: res=a;
                3'd1: res=b;
                3'd2: res=c;
                3'd3: res=d;
                3'd4: res=e;
                3'd5: res=f;
                3'd6: res=g;
                3'd7: res=h;
            endcase
        end

endmodule
```

级联模块

```
module II(
        inout [3:0] a,
        input [25:0]b,
        output [31:0]c
        );
        assign c={a,b<<2};
endmodule
```

Pc 模块

```
module pc1(
    input clk,
    input rst,
    input [31:0]data_in,
    output    reg [31:0] data_out

    );
    always @ (posedge clk or posedge rst) begin
        if(rst)
            data_out <= 32'h00400000;
        else
            data_out <= data_in;
    end
endmodule
```

dmem 模块

```
module dmem(
    input clk,//存储器时钟信号，上升沿时向  ram  内部写入数据
    input wena,//存储器读写有效信号，高电平为写有效，低电平为读有效
    input rena,//存储器读写有效信号，高电平为写有效，低电平为读有效
    input [10:0] addr, //  输入地址，指定数据读写的地址
    input [31:0] data_in,
    output [31:0] data_out //  存储器读出的数据，ram  工作时持续输出相应地址的数据
    );
     reg [31:0] data[2047:0];
    assign data_out=data[addr];
    always @(negedge clk) begin
        if(wena)
        begin
            data[addr]<=data_in;
        end
//          else if(rena)
//          begin

//          end
    end
endmodule
```

imem 模块

```
module imem(
```

```
    input [31:0]addr,
    output [31:0]meminst
);

     integer i;
     wire[31:0] myaddr=(addr-32'h00400000);
     dist_mem_gen_0 uut(.a(myaddr[12:2]),.spo(meminst));

//      reg [31:0] reg_inst[1023:0];
//      initial begin
////            $readmemh("D:/A-STUDY/PLOG/EXP/CPU31/cpu_31/MARS_TO_txt/lwswtest.txt",
reg_inst);
//                  $readmemh("D:/A-STUDY/PLOG/EXP/CPU31/cpu_31/MARS_TO_txt/_4_jr.txt",
reg_inst);
//      end
//      assign meminst= reg_inst[(addr-32'h00400000)/4];

Endmodule
```

CLZ 模块

```
module clz(
      input [31:0] num,
      output reg [31:0] result=0
      );
      integer i=31;
      always @(*) begin
            result=0;
            for(i=31;!num[i]&&i>=0;i=i-1)begin
                  result=result+1;
            end
      end

endmodule
```

CP0 模块

```
module cp0(
      input clk,
      input rst,
      input ena,
      input mfc0, //读
      input mtc0, //写
```

```
input [31:0]pc,
input [4:0] cp0_addr,// cp0 中的寄存器地址  读写需要  rd
input [31:0] wdata,
input exception,//异常发生信号
input eret,
input [4:0]cause,
output [31:0] rdata,
output [31:0] status,
output reg [31:0]exc_addr //异常发生地址
);

reg [31:0]cp0_reg[31:0];//CP0 寄存器
parameter [4:0]STATUS=12, CAUSE=13, EPC=14;
parameter Syscall=5'b01000,Break=5'b01001,Teq=5'b01101;
assign status = cp0_reg[STATUS];                    //状态
assign rdata = mfc0 ? cp0_reg[cp0_addr] : 32'bz;     //读
always@(posedge ena or posedge rst)begin
// always@(*)begin
    if(rst)begin
        cp0_reg[STATUS]<=32'h1f;
        cp0_reg[CAUSE]<=32'd0;
        cp0_reg[EPC]<=32'd0;
        exc_addr<=32'd0;
    end
    else if (ena)begin
        if(eret)begin//eret 退出中断，status 右移 5 位恢复，
            cp0_reg[STATUS]=cp0_reg[STATUS]>>5;
            // cp0_reg[STATUS]={5'b00000,cp0_reg[STATUS][31:5]};
            exc_addr<=cp0_reg[EPC];
        end
        else if(mtc0)begin//写
            cp0_reg[cp0_addr] = wdata[31:0];
        end
        else if(exception && cp0_reg[STATUS][0])begin//异常信号且没有被屏蔽
            case(cause)
            Syscall:begin
                if(cp0_reg[STATUS][1]==1'b1)begin
                exc_addr<=32'h00400004;//异常处理程序开始地址
                // exc_addr<=32'h4;//异常处理程序开始地址
                cp0_reg[STATUS]=cp0_reg[STATUS]<<5;
                cp0_reg[EPC]<=pc;
                cp0_reg[CAUSE][6:2]<=Syscall;
                end
                else
```

```
                exc_addr<=pc+4;

                end
            Break:begin
                if(cp0_reg[STATUS][2]==1'b1)begin
                exc_addr<=32'h00400004;//异常处理程序开始地址
                // exc_addr<=32'h4;//异常处理程序开始地址
                cp0_reg[STATUS]=cp0_reg[STATUS]<<5;
                cp0_reg[EPC]<=pc;
                cp0_reg[CAUSE][6:2]<=Break;
                end
                else
                exc_addr<=pc+4;

                end
            Teq:begin
                if(cp0_reg[STATUS][3]==1'b1)begin
                exc_addr<=32'h00400004;//异常处理程序开始地址
                // exc_addr<=32'h4;//异常处理程序开始地址
                cp0_reg[STATUS]=cp0_reg[STATUS]<<5;
                cp0_reg[EPC]<=pc;
                cp0_reg[CAUSE][6:2]<=Teq;
                end
                else
                exc_addr<=pc+4;

                end
            default:begin
                // exc_addr<=pc+4;
            end
            endcase

        end
        // else
        //          exc_addr<=0;
    end
end
endmodule
```

Div 模块

```
module DIV(
    input [31:0] dividend,
    input [31:0] divisor,
```

```
    input start,
    input clock,
    input reset,
    output    reg [31:0]q,
    output    reg [31:0]r,
    output reg busy
);
    reg [63:0] temp;
    reg [31:0] temp_b;
    integer cnt=0;
        always @ (posedge clock or posedge reset)begin
            if (reset == 1) begin                                   //????
              cnt=0;
              busy <= 0;

            end
            else begin
            if(start)
                busy=1;
             if(busy)begin

                temp=0;
                if(divisor[31]==1)
                    temp_b=~(divisor-1);
                else
                    temp_b=divisor;
                if(dividend[31]==1)
                    temp[31:0]=~(dividend-1);
                else
                    temp[31:0]=dividend;


                for(cnt=0;cnt<32;cnt=cnt+1)begin
                        temp=temp<<1;
                        if(temp[63:32]>=temp_b)begin
                            temp[63:32]=temp[63:32]-temp_b;
                            temp=temp+1;
                        end

                end
                if((dividend[31]==0&&divisor[31]==1))
                begin
                    q=~temp[31:0]+1;
                    r=temp[63:32];
```

```
                end
                else if((dividend[31]==1&&divisor[31]==0))
                begin
                        q=~temp[31:0]+1;
                        r=~temp[63:32]+1;
                end
                else if((dividend[31]==1&&divisor[31]==1))
                begin
                        q=temp[31:0];
                        r=~temp[63:32]+1;
                end
                else
                begin
                        q=temp[31:0];
                        r=temp[63:32];
                end

            end

        end
    end
endmodule
```

Divu 模块

```
module DIVU(
    input [31:0] dividend,
    input [31:0] divisor,
    input start,
    input clock,
    input reset,
    output   reg [31:0]q,
    output   reg [31:0]r,
    output reg busy
);
    reg [63:0] temp;
    integer cnt=0;
        always @ (posedge clock or posedge reset)begin
            if (reset == 1) begin                                   //ÖŘÖĂ
              cnt=0;
              busy <= 0;
              temp=0;
              temp[31:0]=dividend;
```

```
            end
            else begin
            if(start)
                busy=1;
             if(busy)begin
                temp=0;
                    temp[31:0]=dividend;
                for(cnt=0;cnt<32;cnt=cnt+1)begin
                        temp=temp<<1;
                        if(temp[63:32]>=divisor)begin
                            temp[63:32]=temp[63:32]-divisor;
                            temp=temp+1;
                        end

                end
                q=temp[31:0];
                r=temp[63:32];
            end

            end
        end
endmodule
```

Mul 模块

```
module MULT(
    input clk, //乘法器时钟信号
    input reset, //复位信号，高电平有效
    input [31:0] a, //输入数  a(被乘数)
    input [31:0] b, //输入数  b（乘数）
    output reg [63:0] z //乘积输出  z
    );
        reg [63:0] temp;
        reg [63:0]temp_a;
        reg [31:0] tempax;
        reg [31:0] temp_b;
        integer cnt=0;
        always@(posedge clk or posedge reset)
        begin
            if(reset)
```

```
begin
        temp<=0;

        cnt<=0;
end
else
begin
        temp<=0;

        if(a[31]==0&&b[31]==1)
                begin
                temp_a<={32'b11111111111111111111111111111111,b};
                temp_b<=a;
                end
        else if(a[31]==0&&b[31]==0)
                begin
                temp_a<={32'b0,a};
                  temp_b<=b;
                end
      else if(a[31]==1&&b[31]==1)
                begin
                tempax=~(a-1);
                temp_b=~(b-1);
                temp_a<={32'b0,tempax};
                end
      else
                begin
                temp_a<={32'b11111111111111111111111111111111,a};
                temp_b<=b;
                end


  cnt<=0;
      for(cnt=0;cnt<32;cnt=cnt+1)
        begin

                if(temp_b[0])
                begin
                        temp=temp+temp_a;
                end
                else
                begin
                end
                temp_b=temp_b>>1;
```

```
                    temp_a=temp_a << 1;
                end
            z=temp;
            end
        end
      // assign z=temp;
  endmodule
```

Multu 模块

```
module MULTU(
    input clk,
    input reset,//高电平有效
    input [31:0] a,
    input [31:0] b,
    output reg [63:0] z
    );
    reg [63:0] temp;
    reg [63:0]temp_a;
    reg [31:0] temp_b;
    integer cnt=0;
    always@(posedge clk or posedge reset)
    begin
        if(reset)
        begin
            temp<=0;
            temp_a<={32'b0,a};//无符号加 0
            temp_b<=b;
            cnt<=0;
        end
        else
        begin
        temp<=0;
         temp_a<={32'b0,a};//无符号加 0
         temp_b<=b;
         cnt<=0;
            for(cnt=0;cnt<32;cnt=cnt+1)
             begin

                if(temp_b[0])
                begin
                    temp=temp+temp_a;
                end
                else
```

```
                    begin
                    end
                  temp_b=temp_b>>1;
                 temp_a=temp_a << 1;
                end
            end
            z=temp;
        end
      // assign z=temp;
endmodule
```

RegZ 模块

```
module RegZ(
        input rst,
        input zin,
        input zout,
        input[31:0] wdata,
        output[31:0]rdata
        );
        reg [31:0]data;
        always@(*)
                if(rst)
                        data=32'b0;
                else if(zin)
                        data=wdata;
        assign rdata=zout?data:32'bz;
endmodule
```

HILO 模块

```
module HILO(
        input wena,
        input rst,
        input[31:0] wdata,
        output[31:0]rdata
        );
        reg [31:0]data;
        always@(*)
                if(rst)
                        data<=32'b0;
                else if(wena)
                        data<=wdata;
```

```
    assign rdata=data;
endmodule
```

四、测试模块建模

```
module cpu54_tb(

    );
     reg clk;
        reg reset;
        wire [31:0]pc;
        wire [31:0]inst;
                wire [31:0]res_m1;
                reg [31:0] pc_pre;
        sccomp_dataflow uut(clk,reset,inst,pc);
        integer file_output;
        integer counter = 0;
        initial begin

              file_output = $fopen("C:/Users/Zhenghao/Desktop/result2.txt");
               pc_pre=0;
              reset = 1;
              clk = 0;
              #2      reset = 0;
        end


     always begin
     #0.1 clk=~clk;

          if(clk==1'b1&&reset==0)begin
               if(pc_pre!=pc)begin
               counter = counter+1;
                                    $fdisplay(file_output,"pc: %h",pc);
                                    $fdisplay(file_output,"instr: %h",inst);

$fdisplay(file_output,"regfile0: %h",cpu54_tb.uut.sccpu.cpu_ref.array_reg[0]);

$fdisplay(file_output,"regfile1: %h",cpu54_tb.uut.sccpu.cpu_ref.array_reg[1]);

$fdisplay(file_output,"regfile2: %h",cpu54_tb.uut.sccpu.cpu_ref.array_reg[2]);
```

```
$fdisplay(file_output,"regfile3: %h",cpu54_tb.uut.sccpu.cpu_ref.array_reg[3]);

$fdisplay(file_output,"regfile4: %h",cpu54_tb.uut.sccpu.cpu_ref.array_reg[4]);

$fdisplay(file_output,"regfile5: %h",cpu54_tb.uut.sccpu.cpu_ref.array_reg[5]);

$fdisplay(file_output,"regfile6: %h",cpu54_tb.uut.sccpu.cpu_ref.array_reg[6]);

$fdisplay(file_output,"regfile7: %h",cpu54_tb.uut.sccpu.cpu_ref.array_reg[7]);

$fdisplay(file_output,"regfile8: %h",cpu54_tb.uut.sccpu.cpu_ref.array_reg[8]);

$fdisplay(file_output,"regfile9: %h",cpu54_tb.uut.sccpu.cpu_ref.array_reg[9]);

$fdisplay(file_output,"regfile10: %h",cpu54_tb.uut.sccpu.cpu_ref.array_reg[10]);

$fdisplay(file_output,"regfile11: %h",cpu54_tb.uut.sccpu.cpu_ref.array_reg[11]);

$fdisplay(file_output,"regfile12: %h",cpu54_tb.uut.sccpu.cpu_ref.array_reg[12]);

$fdisplay(file_output,"regfile13: %h",cpu54_tb.uut.sccpu.cpu_ref.array_reg[13]);

$fdisplay(file_output,"regfile14: %h",cpu54_tb.uut.sccpu.cpu_ref.array_reg[14]);

$fdisplay(file_output,"regfile15: %h",cpu54_tb.uut.sccpu.cpu_ref.array_reg[15]);

$fdisplay(file_output,"regfile16: %h",cpu54_tb.uut.sccpu.cpu_ref.array_reg[16]);

$fdisplay(file_output,"regfile17: %h",cpu54_tb.uut.sccpu.cpu_ref.array_reg[17]);

$fdisplay(file_output,"regfile18: %h",cpu54_tb.uut.sccpu.cpu_ref.array_reg[18]);

$fdisplay(file_output,"regfile19: %h",cpu54_tb.uut.sccpu.cpu_ref.array_reg[19]);

$fdisplay(file_output,"regfile20: %h",cpu54_tb.uut.sccpu.cpu_ref.array_reg[20]);

$fdisplay(file_output,"regfile21: %h",cpu54_tb.uut.sccpu.cpu_ref.array_reg[21]);

$fdisplay(file_output,"regfile22: %h",cpu54_tb.uut.sccpu.cpu_ref.array_reg[22]);

$fdisplay(file_output,"regfile23: %h",cpu54_tb.uut.sccpu.cpu_ref.array_reg[23]);

$fdisplay(file_output,"regfile24: %h",cpu54_tb.uut.sccpu.cpu_ref.array_reg[24]);
```

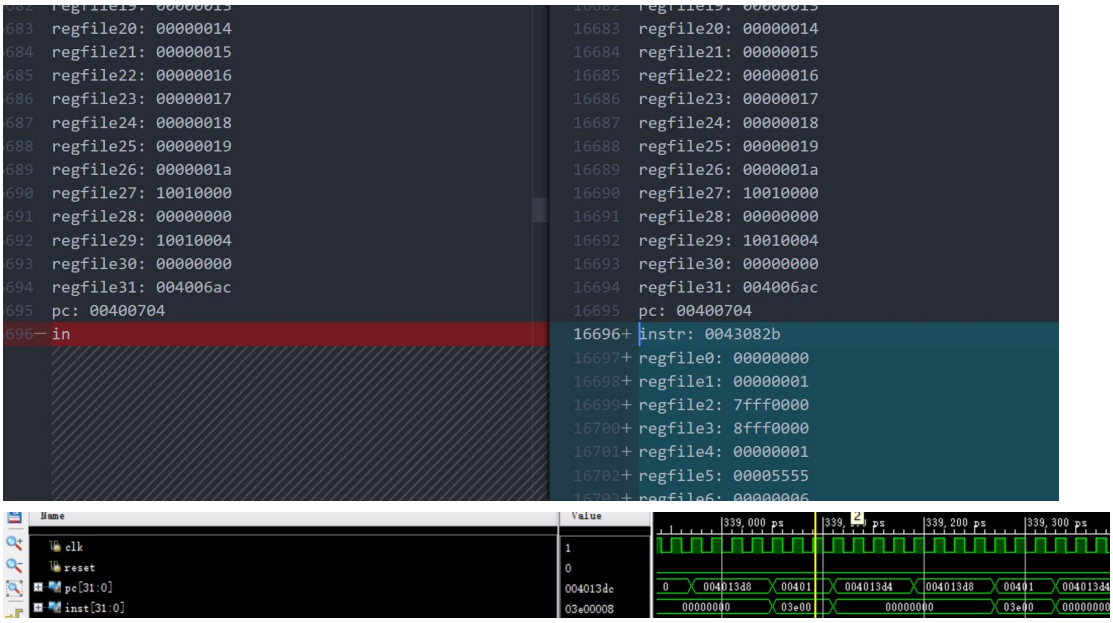
```
$fdisplay(file_output,"regfile25: %h",cpu54_tb.uut.sccpu.cpu_ref.array_reg[25]);

$fdisplay(file_output,"regfile26: %h",cpu54_tb.uut.sccpu.cpu_ref.array_reg[26]);

$fdisplay(file_output,"regfile27: %h",cpu54_tb.uut.sccpu.cpu_ref.array_reg[27]);

$fdisplay(file_output,"regfile28: %h",cpu54_tb.uut.sccpu.cpu_ref.array_reg[28]);

$fdisplay(file_output,"regfile29: %h",cpu54_tb.uut.sccpu.cpu_ref.array_reg[29]);

$fdisplay(file_output,"regfile30: %h",cpu54_tb.uut.sccpu.cpu_ref.array_reg[30]);

$fdisplay(file_output,"regfile31: %h",cpu54_tb.uut.sccpu.cpu_ref.array_reg[31]);
                                    pc_pre=pc;
                end
            end
          end
endmodule
```

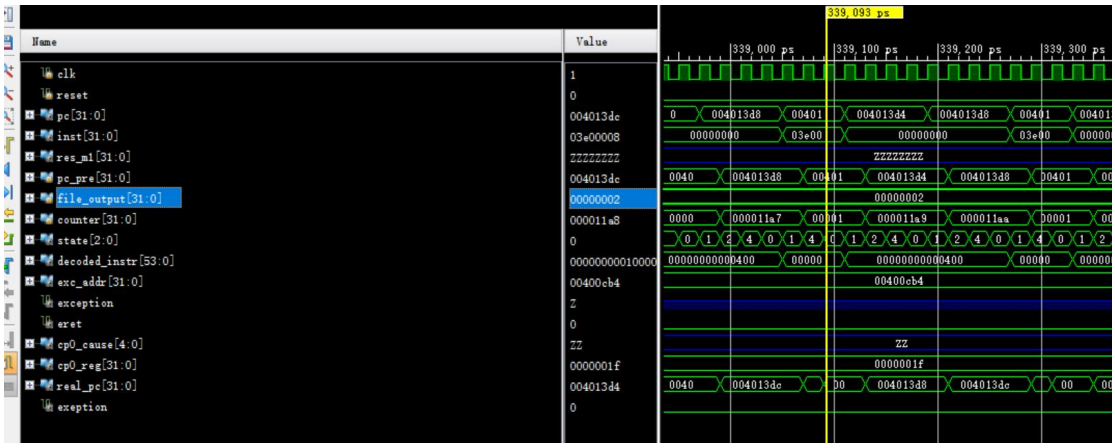
五、实验结果

前仿真：

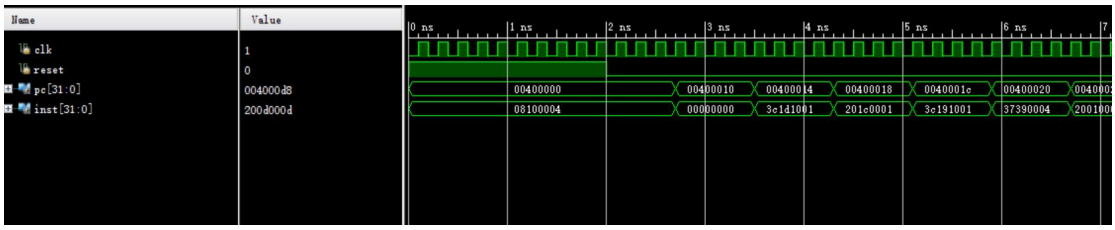
利用 mars 生成 result 进行比对，还用 ip 核进行初始化 imem 进行比对，最终得到结果。

本实验对于 cp0 的调试主要依靠观察波形的变化来调试，其他的指令我会结合标准的答案按照问题所在之处进行逐步的调试。





后仿真：



通过对比同一个 coe 文件在前后仿真下的波形最后得出波形相同，后仿真成功，具体后仿真过程可以在视频文件中看到。