同济大学计算机系

计算机组成原理实验报告

学　　号　1951042

姓　　名　王远洋

专　　业　信息安全

授课老师　张冬冬

目录

一、 实验介绍

在本次实验中，我们将使用 Verilog 语言实现 31 条 MIPS 指令的 CPU 的设计和仿真。

二、 实验目标

- ✧ 深入了解 CPU 的原理
- ✧ 画出实现 31 条指令的 CPU 的通路图
- ✧ 学习使用 Verilog 语言设计实现 31 条指令的 CPU

三、 实验原理

1. 单周期数据通路设计

因为要求设计的 CPU 为单周期，按照单周期数据通路设计的一般性方法，如下所示：

- ➢ 阅读每条指令，对每条指令所需执行的功能与过程都有充分的了解
- ➢ 确定每条指令在执行过程中所用到的部件
- ➢ 使用表格列出指令所用部件，并在表格中填入每个部件的数据输入来源
- ➢ 根据表格所涉及部件和部件的数据输入来源，画出整个数据通路

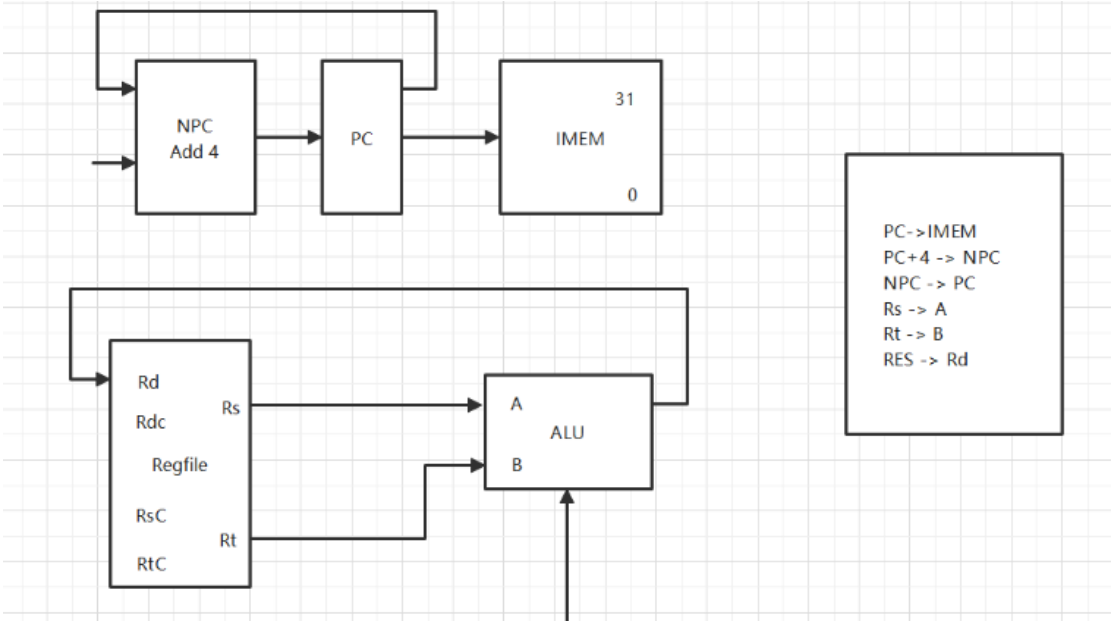
根据方法，下面分析每条指令所使用的部件以及数据通路。

2. 单指令数据通路分析

指令命令总共被分为 3 大类，分别是 R，I，J，其中 R 指令和寄存器有很大的关联，两个操作数都是寄存器； I 指令就是其中有立即数； J 指令甚至没有那么多的操作数，跳转使用，具体分析如下：

2.1 Add 类型指令

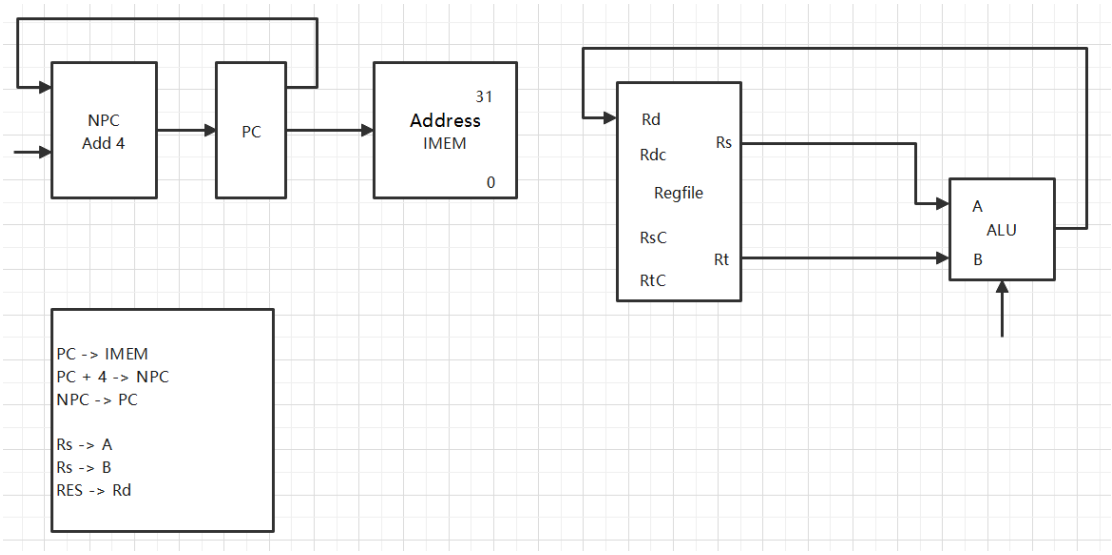
Add 类型的指令包括 add/addu/sub/subu/and/or/xor/nor 共 8 条指令，这些指令只有运算器控制信号不同，除此之外完全一致，因此在同一图中表示，如下：



其中使用的器件为：PC 寄存器、指令存储器、NPC、寄存器文件以及 ALU，注意在使用过程中 ALU 传入的信号的不同。

2.2 Slt 类型指令

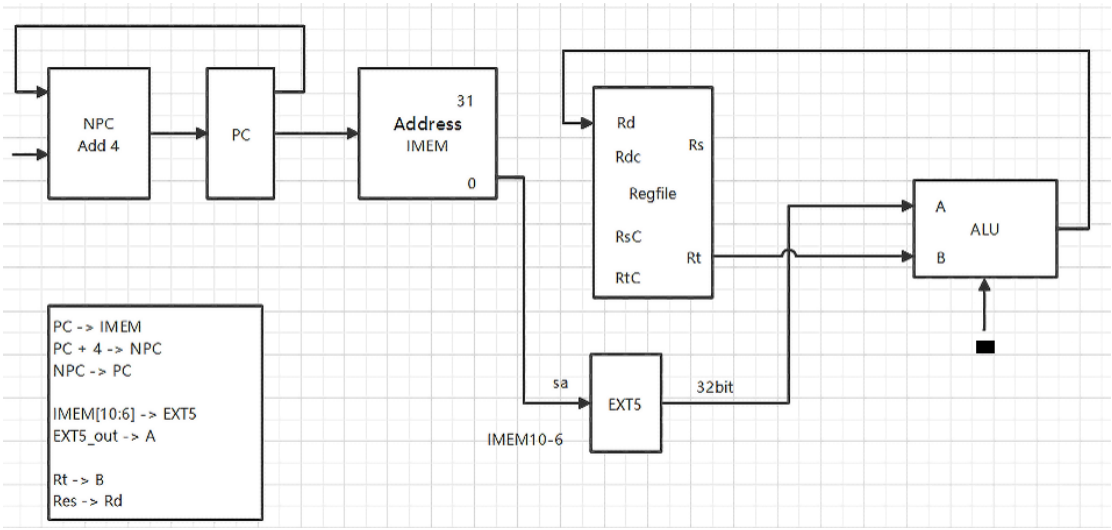
Slt 类型指令包括 slt 和 sltu，其结构通路如图：



使用的器件包括 NPC、PC、指令存储器、寄存器文件以及 ALU，其中，ALU 的信号为 slt 和 sltu.

2.3 Sll 类型指令

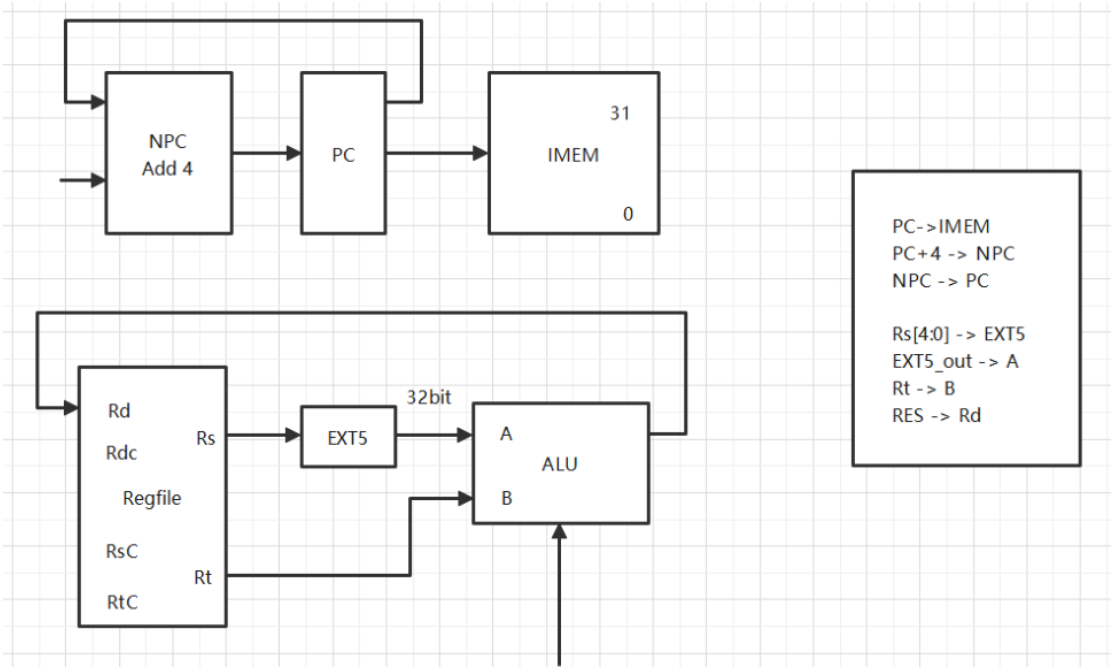
Sll 类型指令包括 sll/srl/sra，其结构通路如图：



使用的器件包括 NPC、PC、指令寄存器、寄存器文件、数据扩展器以及 ALU，其中，ALU 信号为 sll/srl/sra.

2.4 Sllv 类型指令

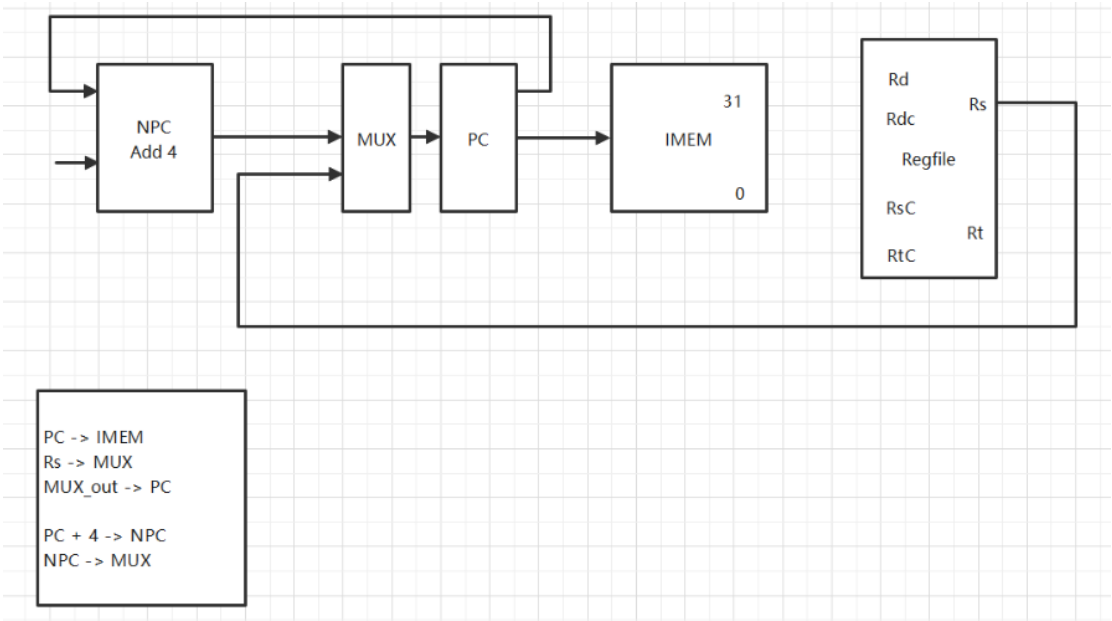
Sllv 类型指令包括 sllv/srlv/srav，其结构通路如图：



使用的器件包括 NPC、PC、指令存储器、寄存器文件、数据扩展器以及 ALU，其中，ALU 信号为 sll/srl/sra.

2.5 Jr 类型指令

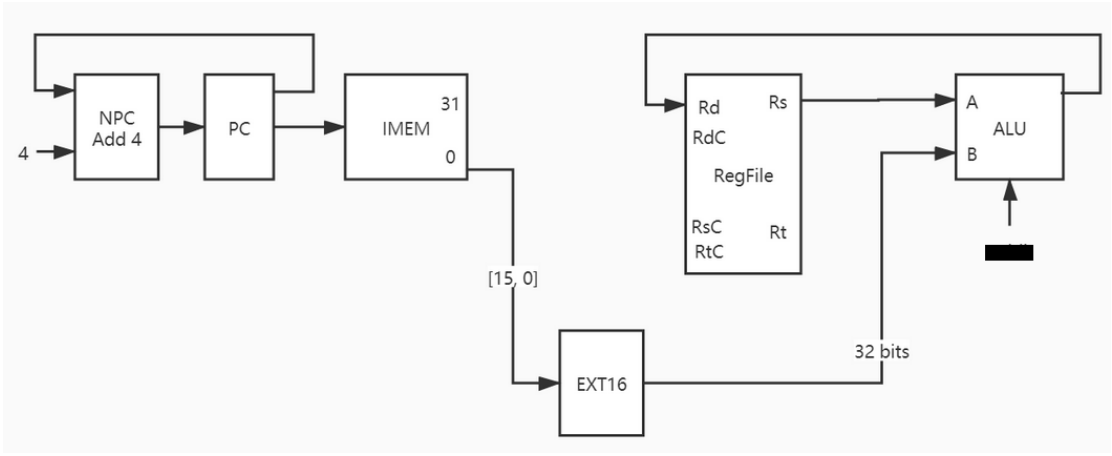
Jr 类型指令即为 jr 指令，其结构通路如图：



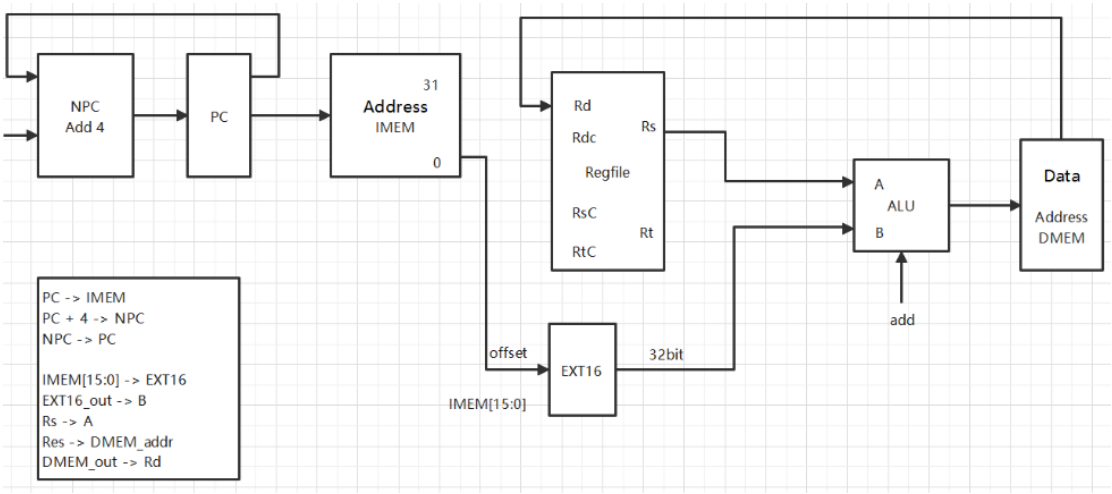
使用的器件包括 NPC、PC、指令存储器、寄存器文件以及二选一 MUX.

2.6 Addi 类型指令

Addi 类型指令包括 addi/addiu/andi/ori/xori，除了 ALU 信号之外完全相同，使用的器件包括 NPC、PC、指令存储器、寄存器文件、数据扩展以及 ALU.

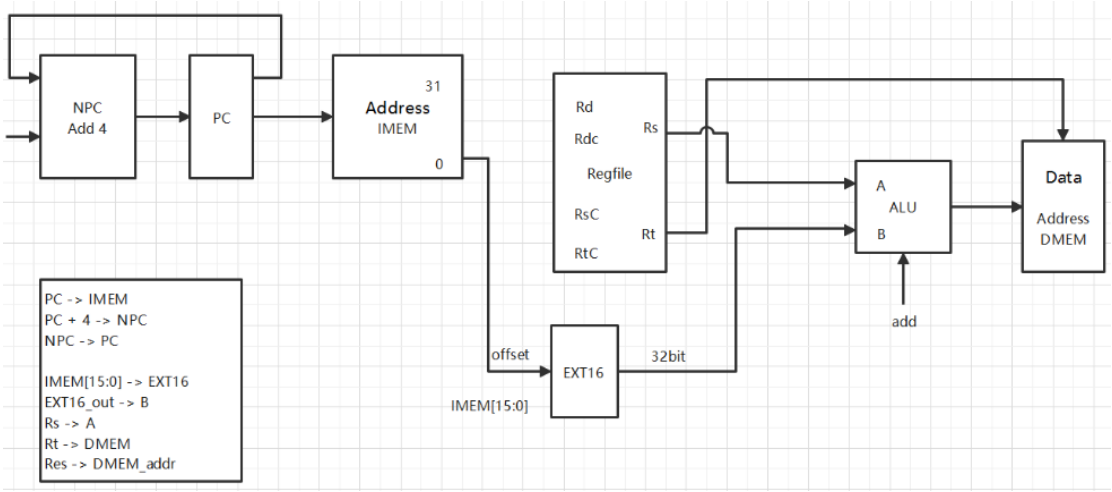


2.7 lw 指令



使用的器件包括 NPC、PC、指令存储器、寄存器文件、数据扩展器、数据存储器以及 ALU.

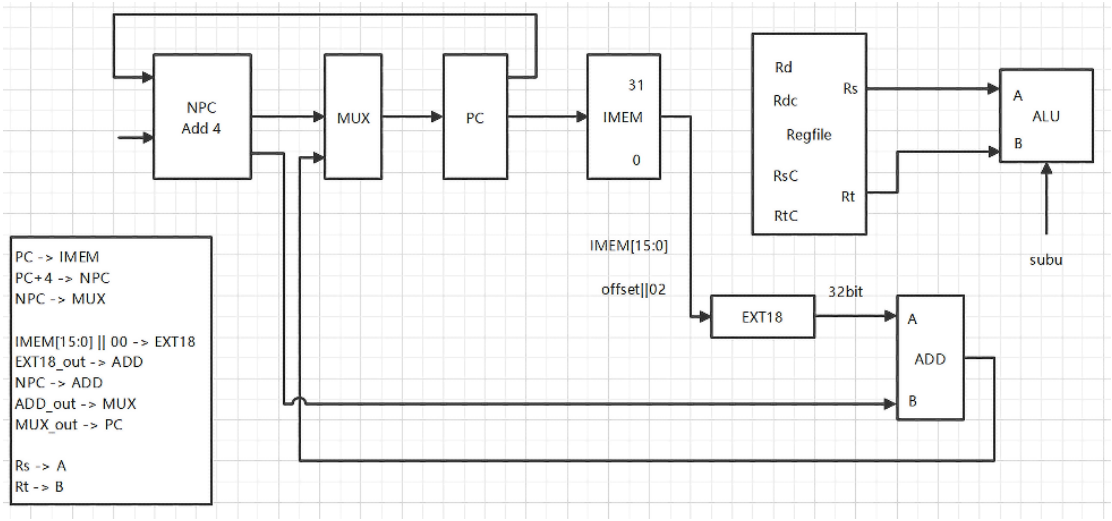
2.8 sw 指令



使用的器件同 lw 指令.

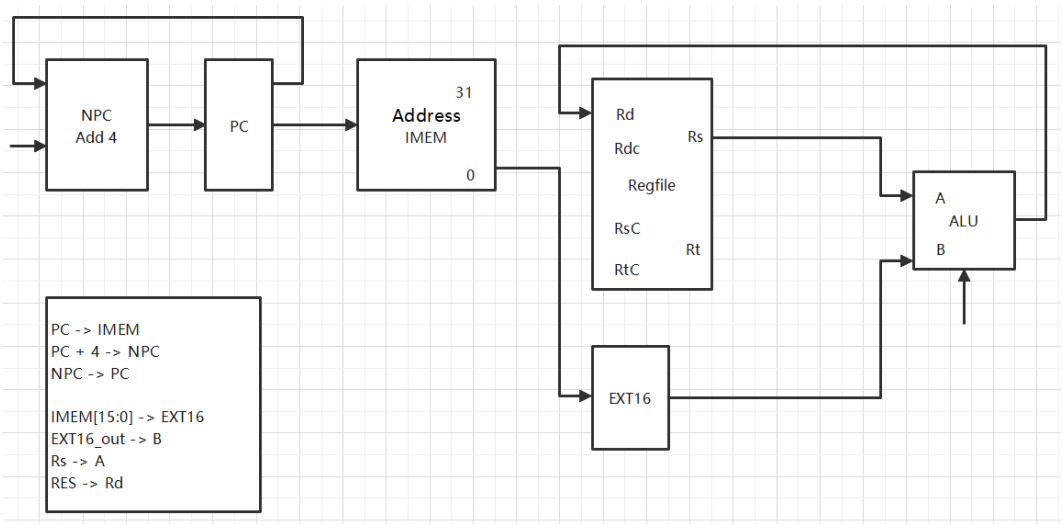
2.9 Beq 和 Bne 指令

Beq 和 Bne 是两条比较跳转指令，区别在于 MUX 的信号不同，其余部分完全相同，其结构通路如图：



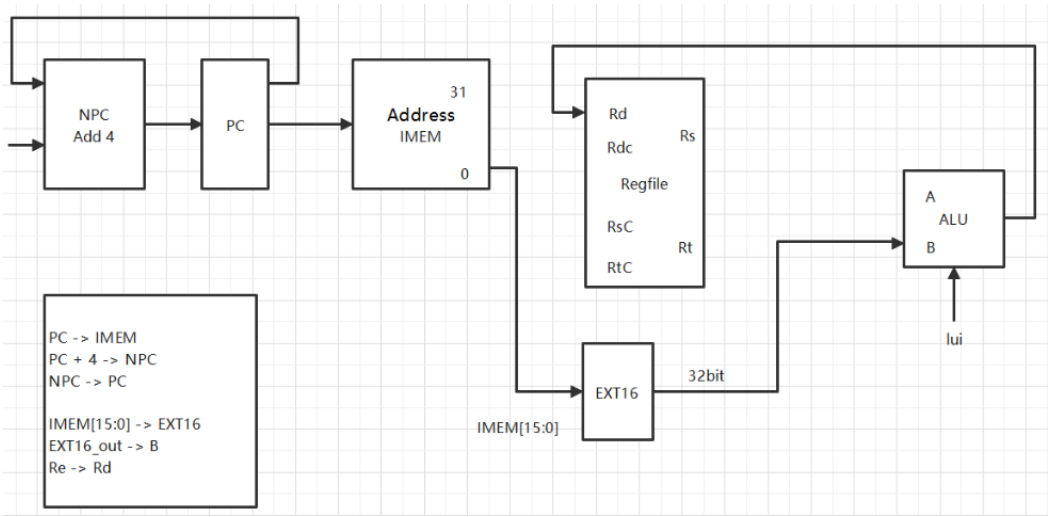
2.10 Slti 和 Sltiu 指令

Slti 和 Sltiu 指令的区别在于 ALU 的控制信号不同，其余部分完全相同，结构通路如图：



使用的器件如图所示，ALU 的控制信号为 slt 和 sltu.

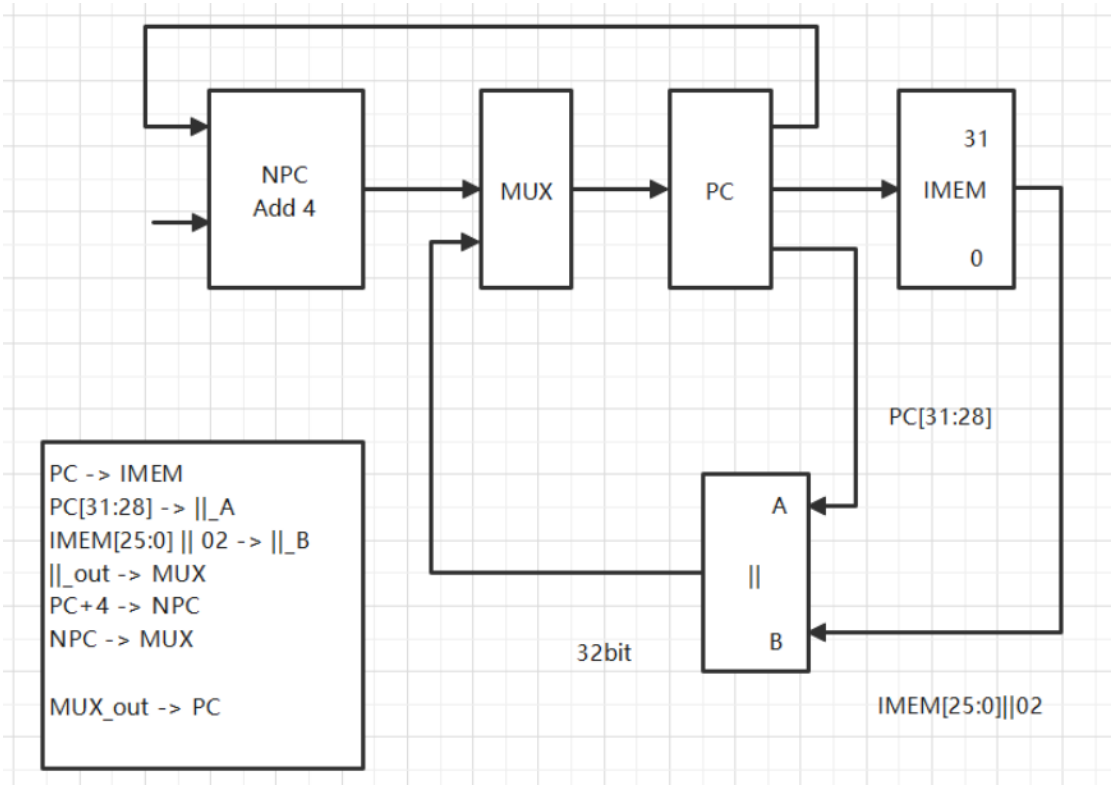
2.11 Lui 指令



Lui 指令比较简单，只要通过 ALU 的运算即可完成，结构通路如上图所示。

2.12 J 指令

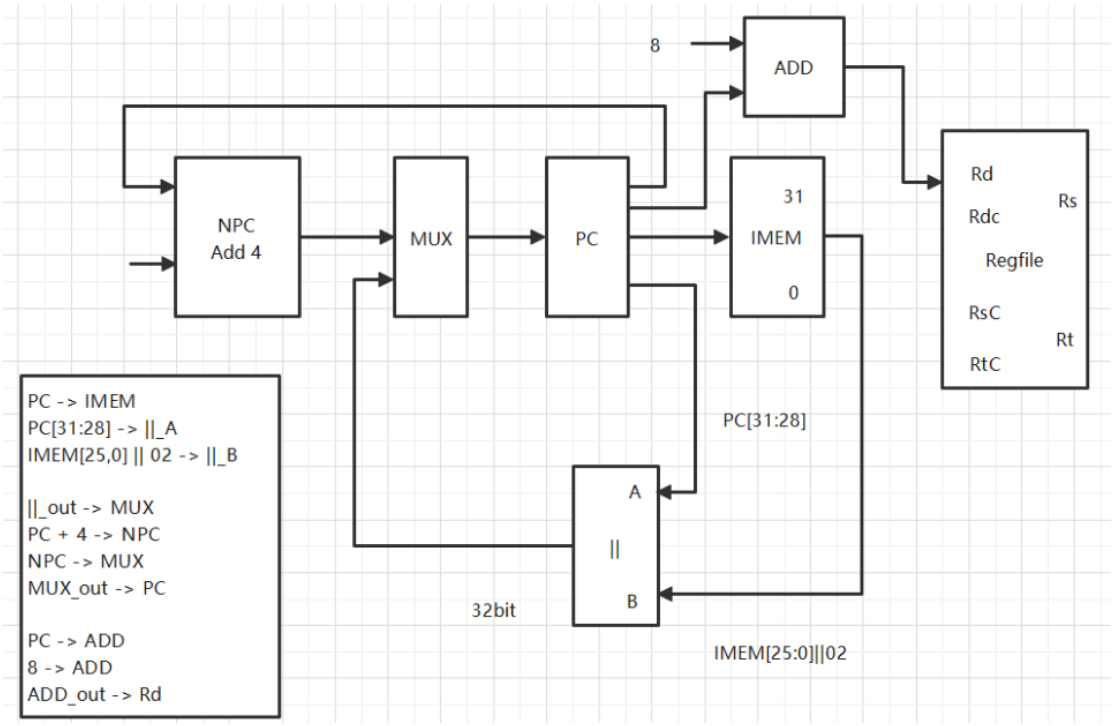
J 指令通路设计如下：



J 指令增加了 II 器件，功能为左移.

2.13 Jal 指令

最后的 Jal 指令如图所示：



3. CPU 整体信号以及通路

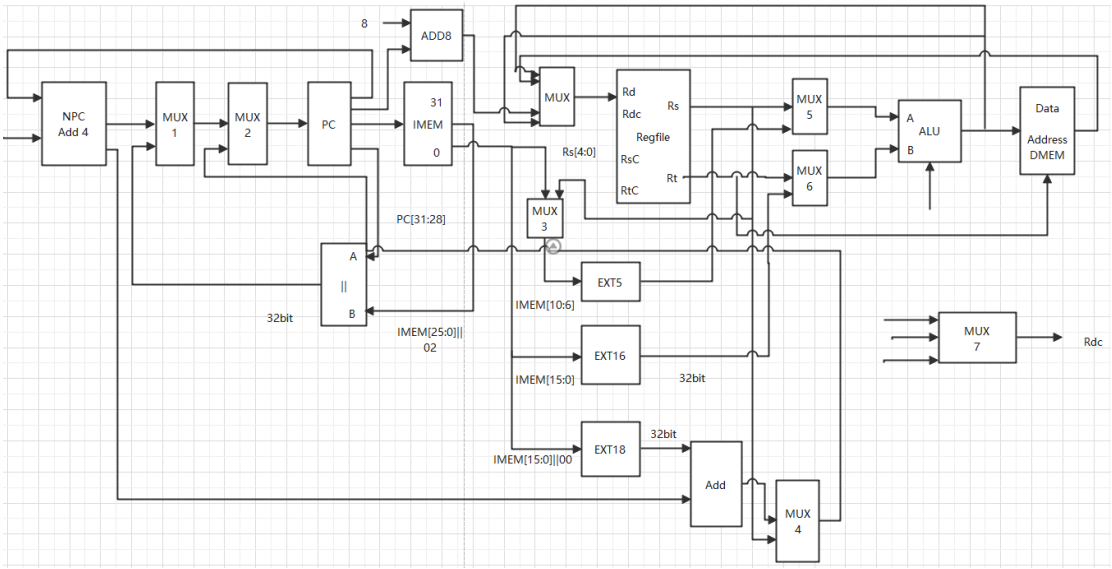
3.1 31 指令控制部件表

31 条指令的控制部件以及输入的信号如下表所示：

指令	PC	IM	RF	ALU		EXT1	EXT5	EXT16	EXT18	MUX	DMEM		ADD		||	
			Wdata	A	B						Data_in	addr	A	B	A	B
add	PC+4	PC	ALU	RF.RD1	RF.RD2											
addu	PC+4	PC	ALU	RF.RD1	RF.RD2											
sub	PC+4	PC	ALU	RF.RD1	RF.RD2											
subu	PC+4	PC	ALU	RF.RD1	RF.RD2											
and	PC+4	PC	ALU	RF.RD1	RF.RD2											
or	PC+4	PC	ALU	RF.RD1	RF.RD2											
xor	PC+4	PC	ALU	RF.RD1	RF.RD2											
nor	PC+4	PC	ALU	RF.RD1	RF.RD2											
slt	PC+4	PC	EXT1	RF.RD1	RF.RD2	ALU										
sltu	PC+4	PC	EXT1	RF.RD1	RF.RD2	ALU										
sll	PC+4	PC	ALU	EXT	RF.RD2		IM[10:6]									
srl	PC+4	PC	ALU	EXT	RF.RD2		IM[10:6]									
sra	PC+4	PC	ALU	EXT	RF.RD2		IM[10:6]									
sllv	PC+4	PC	ALU	EXT	RF.RD2		RF.RD1[4:0]									
srlv	PC+4	PC	ALU	EXT	RF.RD2		RF.RD1[4:0]									
srav	PC+4	PC	ALU	EXT	RF.RD2		RF.RD1[4:0]									
jr	MUX_ou	PC	RF.RD1							PC+4&RF.RD1						
addi	PC+4	PC	ALU	RF.RD1	EXT16			IMEM[15:0]								
addiu	PC+4	PC	ALU	RF.RD1	EXT16			IMEM[15:0]								
andi	PC+4	PC	ALU	RF.RD1	EXT16			IMEM[15:0](无符号)								
ori	PC+4	PC	ALU	RF.RD1	EXT16			IMEM[15:0](无符号)								
xori	PC+4	PC	ALU	RF.RD1	EXT16			IMEM[15:0](无符号)								
lw	PC+4	PC	DM	RF.RD1	EXT16			IMEM[15:0]				ALU				
sw	PC+4	PC	DM	RF.RD1	EXT16			IMEM[15:0]			RF.RD2	ALU				
beq	ADD	PC		RF.RD1	RF.RD2				IMEM[15:0] || 00				EXT18	PC+4		
bne	ADD	PC		RF.RD1	RF.RD2				IMEM[15:0] || 00				EXT18	PC+4		
slti	PC+4	PC	EXT1	RF.RD1	EXT16	ALU		IMEM[15:0]								
sltiu	PC+4	PC	EXT1	RF.RD1	EXT16	ALU		IMEM[15:0]								
j	MUX_ou	PC													PC[31:28]	IMEM[25:0]
jal	MUX_ou	PC	ADD							PC+4 : ||			PC	8	PC[31:28]	IMEM[25:0]

3.2 CPU 结构通路图

结合所有单条指令的结构图，通过引入多路选择器的方式将所有信号整合，便完成了单周期 31 条指令 CPU 的通路图：



3.3 CPU 指令控制信号

在 31 条指令中， CPU 各部件的控制信号具体如下所示：

		PCReg_ena	DM_ena	DM_wen	DM_rena	RF_ena	RF_we	RF_Rsc	RF_Rtc	RF_Rdc	MUX1	MUX2	MUX3	MUX4	MUX5	MUX6	MUX7	MUX_41	ALUC[3]	ALUC[2]	ALUC[1]	ALUC[0]	EXT16	EXT5	EXT18
2	add	1	0	0	0	1	1	IM[25:21]	IM[20:16]	IM[15:11]	0	0			0	0	00	00	0	0	1	0			
3	addu	1	0	0	0	1	1	IM[25:21]	IM[20:16]	IM[15:11]	0	0			0	0	00	00	0	0	0	0			
4	sub	1	0	0	0	1	1	IM[25:21]	IM[20:16]	IM[15:11]	0	0			0	0	00	00	0	0	1	1			
5	subu	1	0	0	0	1	1	IM[25:21]	IM[20:16]	IM[15:11]	0	0			0	0	00	00	0	0	0	1			
6	and	1	0	0	0	1	1	IM[25:21]	IM[20:16]	IM[15:11]	0	0			0	0	00	00	0	1	0	0			
7	or	1	0	0	0	1	1	IM[25:21]	IM[20:16]	IM[15:11]	0	0			0	0	00	00	0	1	0	1			
8	xor	1	0	0	0	1	1	IM[25:21]	IM[20:16]	IM[15:11]	0	0			0	0	00	00	0	1	1	0			
9	nor	1	0	0	0	1	1	IM[25:21]	IM[20:16]	IM[15:11]	0	0			0	0	00	00	0	1	1	1			
10	slt	1	0	0	0	1	1	IM[25:21]	IM[20:16]	IM[15:11]	0	0			0	0	00	11	1	0	1	1			

		PCReg_ena	DM_ena	DM_wen	DM_rena	RF_ena	RF_we	RF_Rsc	RF_Rtc	RF_Rdc	MUX1	MUX2	MUX3	MUX4	MUX5	MUX6	MUX7	MUX_41	ALUC[3]	ALUC[2]	ALUC[1]	ALUC[0]	EXT16	EXT5	EXT18
11	sltu	1	0	0	0	1	1	IM[25:21]	IM[20:16]	IM[15:11]	0	0			0	0	00	11	1	0	1	0			
12	sll	1	0	0	0	1	1		IM[20:16]	IM[15:11]	0	0	0		1	0	00	00	1	1	1	X		0	
13	srl	1	0	0	0	1	1		IM[20:16]	IM[15:11]	0	0	0		1	0	00	00	1	1	0	1		0	
14	sra	1	0	0	0	1	1		IM[20:16]	IM[15:11]	0	0	0		1	0	00	00	1	1	0	0		0	
15	sllv	1	0	0	0	1	1	IM[25:21]	IM[20:16]	IM[15:11]	0	0	1		1	0	00	00	1	1	1	X		0	
16	srlv	1	0	0	0	1	1	IM[25:21]	IM[20:16]	IM[15:11]	0	0	1		1	0	00	00	1	1	0	1		0	
17	srav	1	0	0	0	1	1	IM[25:21]	IM[20:16]	IM[15:11]	0	0	1		1	0	00	00	1	1	0	0		0	
18	jr	1	0	0	0	1	0	IM[25:21]			0	1		1											
19	addi	1	0	0	0	1	1	IM[25:21]		IM[20:16]	0	0			0	1	01	00	0	0	1	0	1		
20	addiu	1	0	0	0	1	1	IM[25:21]		IM[20:16]	0	0			0	1	01	00	0	0	0	0	1		

		PCReg_ena	DM_ena	DM_wen	DM_rena	RF_ena	RF_we	RF_Rsc	RF_Rtc	RF_Rdc	MUX1	MUX2	MUX3	MUX4	MUX5	MUX6	MUX7	MUX_41	ALUC[3]	ALUC[2]	ALUC[1]	ALUC[0]	EXT16	EXT5	EXT18
20	addiu	1	0	0	0	1	1	IM[25:21]		IM[20:16]	0	0			0	1	01	00	0	0	0	0	1		
21	andi	1	0	0	0	1	1	IM[25:21]		IM[20:16]	0	0			0	1	01	00	0	1	0	0	0		
22	ori	1	0	0	0	1	1	IM[25:21]		IM[20:16]	0	0			0	1	01	00	0	1	0	1	0		
23	xori	1	0	0	0	1	1	IM[25:21]		IM[20:16]	0	0			0	1	01	00	0	1	1	0	0		
24	lw	1	1	1	1	1	1	IM[25:21]		IM[20:16]	0	0			0	1	01	01	0	0	0	0	1		
25	sw	1	1	1	0	1	0	IM[25:21]	IM[20:16]		0	0			0	1			0	0	0	0	1		
26	beq	1	0	0	0	1	0	IM[25:21]	IM[20:16]		0	alu		0	0	0			0	0	1	1			1
27	bne	1	0	0	0	1	0	IM[25:21]	IM[20:16]		0	alu		0	0	0			0	0	0	1			1
28	slti	1	0	0	0	1	1	IM[25:21]		IM[20:16]	0	0			0	1	01	11	1	0	1	1	1		
29	sltiu	1	0	0	0	1	1	IM[25:21]		IM[20:16]	0	0			0	1	01	11	1	0	1	0	1		
30	lui	1	0	0	0	1	1			IM[20:16]	0	0				1	01	00	1	0	0	X	0		
31	j	1	0	0	0	0	0				1	0													
32	jal	1	0	0	0	1	1			R[3:1]	1	0					10	10							

通过以上所有的结构图以及信号，便可以设计出 CPU.

四、 模块设计

1. 顶层模块设计

根据网站提交要求，顶层模块设置如下：

```
module sccomp_dataflow(
    input clk_in,
    input reset,
    output [31:0] inst,
    output [31:0] pc
    );
```

2. CPU 模块

CPU 模块与顶层模块进行信号的交换控制，对外主要是 DM 和 IM 的信号控制，内部主要实现 CPU 各部件之间的信号输入输出，形成完整通路，模块设计如下：

```
module cpu(
    input clk,
    input rst,
    input [31:0] IM_inst,
    input [31:0] DM_rdata,
    output [31:0] pc,
    output [31:0] DM_addr,
    output [31:0] DM_wdata,
    output DM_ena,
    output DM_we,
    output DM_re
    );
```

3. IMEM 模块

指令存储器的 IP 核由 vivado 提供，需要在使用时调用，设计如下：

```
module IMEM(
    input [10:0] addr,
    output [31:0] instr
    );
```

4. DMEM 模块

DMEM 模块需要人工书写，其模块设计如下：

```
module DMEM(
    input clk,
    input ena,
    input wsignal,
    input rsignal,
    input [10:0] addr,
    input [31:0] wdata,
    output [31:0] rdata
    );
```

5. Regfile 模块

Regfile 是寄存器文件，模块设计如下：

```
module regfile(
    input clk,
    input ena,
    input rst,
    input we,  // high--write, low--read
    input [4:0] Rsc,
    input [4:0] Rtc,
    input [4:0] Rdc,
    input [31:0] Rd,
    output [31:0] Rs,
    output [31:0] Rt
    );
```

6. ALU 模块

ALU 模块负责运算部分，模块设计如下：

```
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
    );
```

7. PCReg 模块

PC 寄存器，初始化时为 pc 变量赋值，模块设计如下：

```
module PCReg(
    input clk,
    input rst,
    input ena,
    input [31:0] data_in,
    output reg [31:0] data_out
    );
```

8. Controller 模块

Controller 模块负责对各部件信号的控制，此模块可以包含在 CPU 中，为了保证结构清晰，将其单独提取出，设计如下：

```
module Controller(
    input clk,
    input z,
    input [31:0] icode,
    output PC_clk,
    output PC_ena,
    output M1,
    output M2,
    output M3,
    output M4,
    output M5,
    output M6,
    output [1:0] M7,
    output [1:0] M,
    output [3:0] ALUC,
    output RF_ena,
    output RF_we,
    output RF_clk,
    output DM_ena,
    output DM_we,
    output DM_re,
    output EXT16_s
    );
```

9. 其它模块

除了主要的部件之外，还有一些小部件，包括多路选择器、左移功能器件、符号扩展器、加法器以及实现特定加法功能的 NPC 和 ADD8 部件，模块设计比较简单，不做列举。

五、 测试模块设计

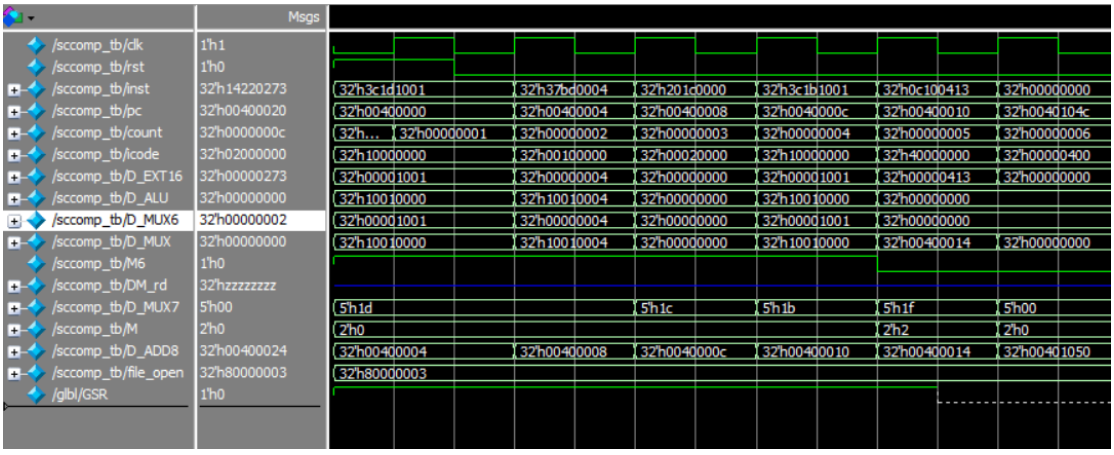
测试模块可以用来检验 CPU 指令设计是否正确，判断标准可以通过寄存器的值进行观察，因此，测试模块实现寄存器值的文件输出，同时也可以设置一些中间变量进行观测，下面是输出到文件的部分代码：

```
.....
$fdisplay(file_open, "regfile0: %h", sc.sccpu.cpu_ref.array_reg[0]);
$fdisplay(file_open, "regfile1: %h", sc.sccpu.cpu_ref.array_reg[1]);
.....
$fdisplay(file_open, "pc: %h", sc.pc);
$fdisplay(file_open, "instr: %h", sc.inst);
.....
sccomp_dataflow sc(
    .clk_in(clk), .reset(rst),
    .inst(inst), .pc(pc)
    );
.....
```

六、 实验结果

1. 前仿真测试

1.1 ModelSim 仿真图像



2. 后仿真测试

待测中……