

同济大学计算机系
计算机组成原理课程设计实验报告



题目：31 条 MIPS 指令 CPU 设计

学 号 _____

姓 名 _____

专 业 _____

授课老师 _____

日期：

一、实验环境部署与硬件配置说明

开发环境：Vivado

语言：Verilog

cpu 框架：Mips

控制器：组合逻辑

二、实验的总体结构

本次实验共分为三步骤：

- 1. 绘制指令数据通路
- 2. 画出 CPU 的总体数据通路，设置控制信号表
- 3. 编写代码

1. 指令单体详解：

(1) ADD

格式：ADD rd, rs, rt

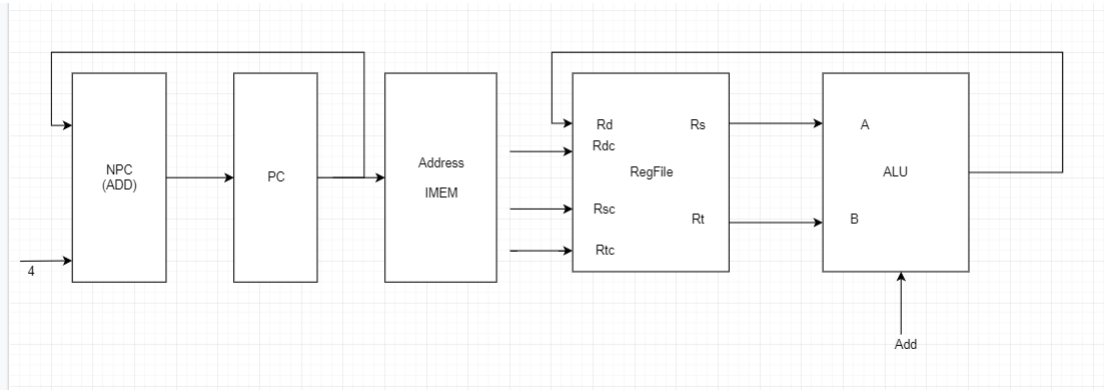
目的：与 32 位数相加

描述： $rd \leftarrow rs + rt$

所用部件：NPC,PC,IMEN,RF,ALU

编号	指令	PC	NPC	IM	RF	ALU	
					Wdata	A	B
1	ADD	NPC	PC	PC	ALU	Rs	Rt

数据通路：



(2) ADDU

格式：ADDU rd, rs, rt

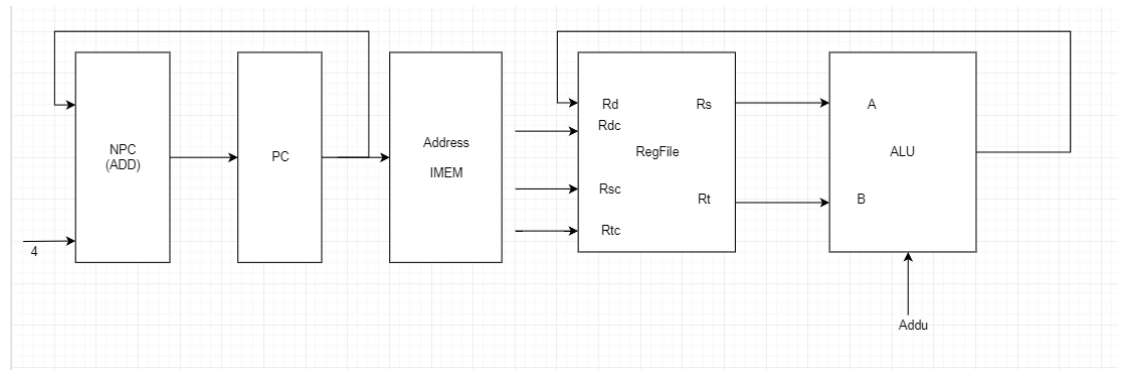
目的：32 位数据相加

描述： $rd \leftarrow rs + rt$

所用部件：NPC,PC,IM,RF,ALU

编号	指令	PC	NPC	IM	RF	ALU	
					Wdata	A	B
1	ADDU	NPC	PC	PC	ALU	Rs	Rt

数据通路:



(3) SUB

格式: SUB rd, rs, rt MIPS32

目的: 与 32 位数相减

描述: $rd \leftarrow rs - rt$

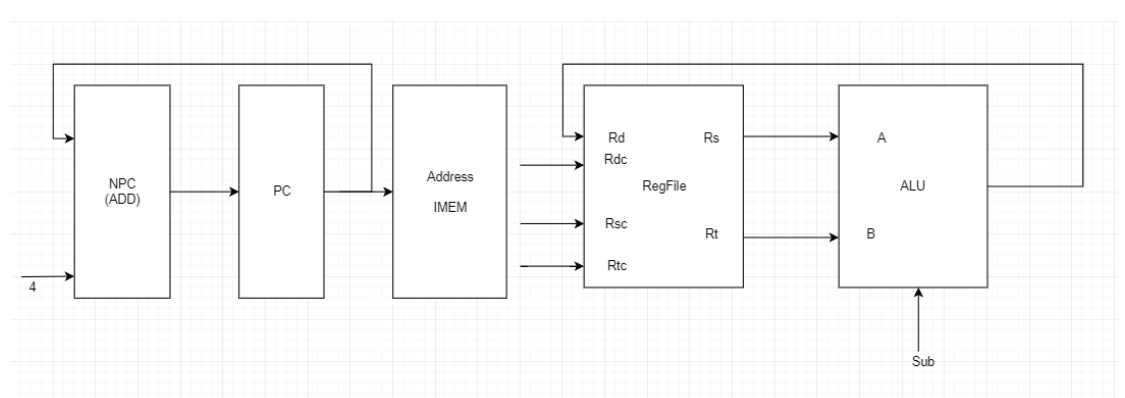
将通用寄存器中存的 32 位数据 rs 与 rt 相减产生一个 32 位数据存入目标寄存器 rd。

- 如果发生了溢出，则 rd 不改变并且产生一个溢出的异常。
- 如果相加不溢出，则产生的 32 位数据直接存入目标寄存器 rd。

所用部件: NPC,PC,IM,RF,ALU

编号	指令	PC	NPC	IM	RF	ALU	
					Wdata	A	B
3	SUB	NPC	PC	PC	ALU	Rs	Rt

数据通路:



(4) SUBU

格式: SUBU rd, rs, rt

目的: 32 位数据相减

描述: $rd \leftarrow rs - rt$

将通用寄存器中存的 32 位数据 rs 与 rt 相减产生一个 32 位数据存入目标寄存器 rd。

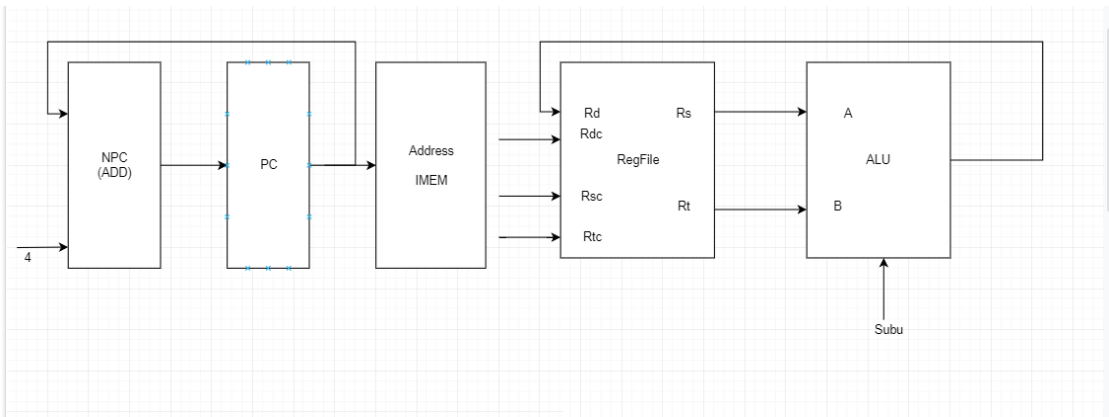
在任何情况下都不会有溢出的异常。

所用部件: PC,NPC,IM,RF,ALU

编号	指令	PC	NPC	IM	RF	ALU	
					Wdata	A	B

4	SUBU	NPC	PC	PC	ALU	Rs	Rt
---	------	-----	----	----	-----	----	----

数据通路:



(5) AND

格式: **AND** rd, rs, rt

目的: 按位逻辑与

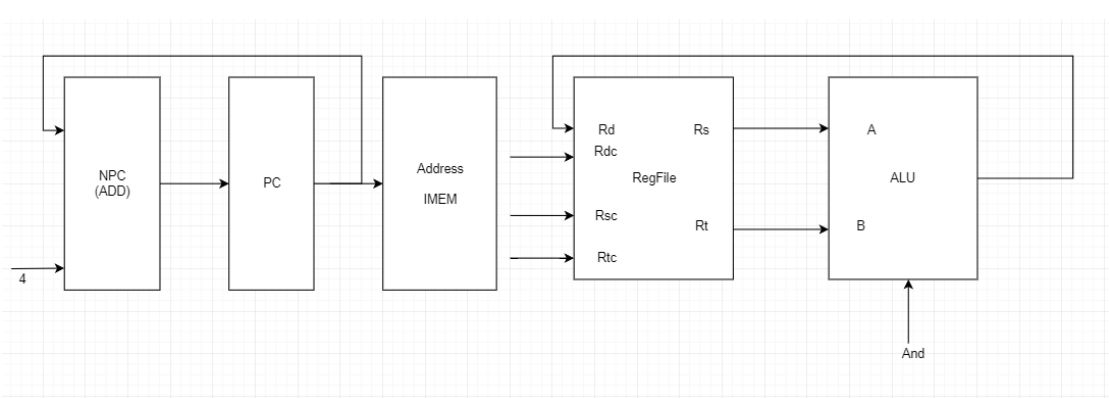
描述: $rd \leftarrow rs \text{ AND } rt$

将通用寄存器 rs 和 rd 中的数据每一位做按位与操作，将结果存入目标寄存器 rd 中。

所用部件: PC,NPC,IM,RF,ALU

编号	指令	PC	NPC	IM	RF	ALU	
					Wdata	A	B
5	AND	NPC	PC	PC	ALU	Rs	Rt

数据通路:



(6) OR

格式: **OR** rd, rs, rt

目的: 按位逻辑或

描述: $rd \leftarrow rs \text{ or } rt$

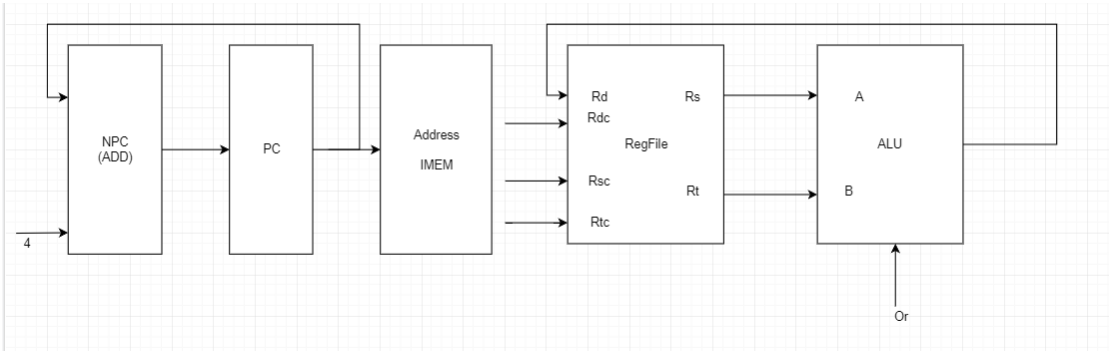
将通用寄存器 rs 和 rt 中的数据每一位做按位或操作，将结果存入目标寄存器 rd 中。

所用部件: PC,NPC,IM,RF,ALU

编号	指令	PC	NPC	IM	RF	ALU	
					Wdata	A	B

6	OR	NPC	PC	PC	ALU	Rs	Rt
---	----	-----	----	----	-----	----	----

数据通路:



(7) XOR

格式: XOR rd, rs, rt

目的: 按位逻辑异或

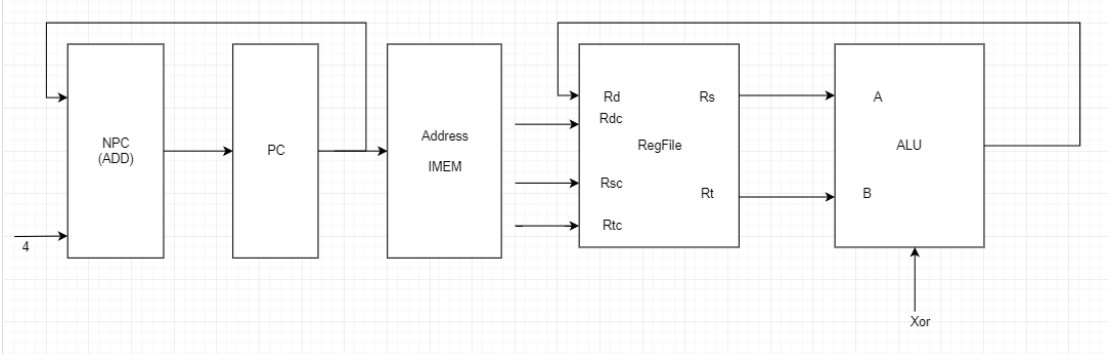
描述: $rd \leftarrow rs \text{ XOR } rt$

将通用寄存器 rs 和 rt 中的内容按位进行异或操作，将结果存入 rd 中。

所用部件:

编号	指令	PC	NPC	IM	RF	ALU	
					Wdata	A	B
7	XOR	NPC	PC	PC	ALU	Rs	Rt

数据通路:



(8) NOR

格式: NOR rd, rs, rt

目的: 按位逻辑或非

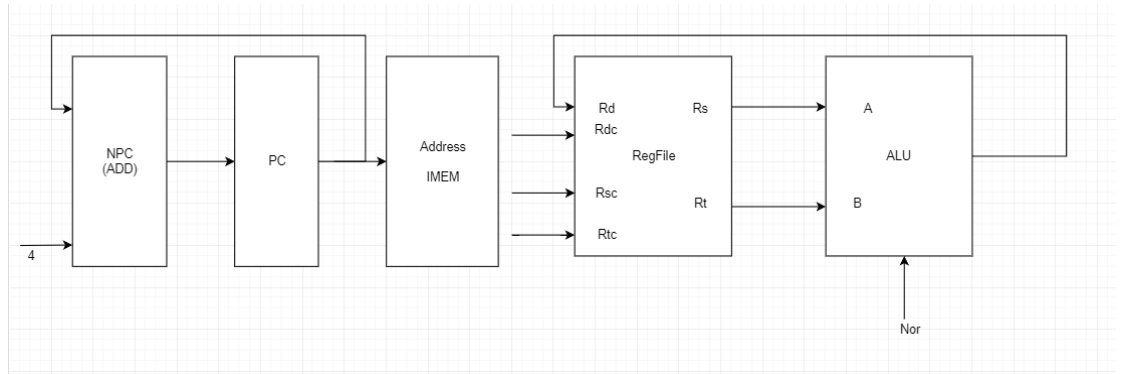
描述: $rd \leftarrow rs \text{ NOR } rt$

将通用寄存器 rs 和 rt 中的数据每一位做按位或非操作，将结果存入目标寄存器 rd 中。

所用部件:

编号	指令	PC	NPC	IM	RF	ALU	
					Wdata	A	B
8	NOR	NPC	PC	PC	ALU	Rs	Rt

数据通路:



(9) SLT

格式: SLT rd, rs, rt

目的: 通过小于的比较来记录结果

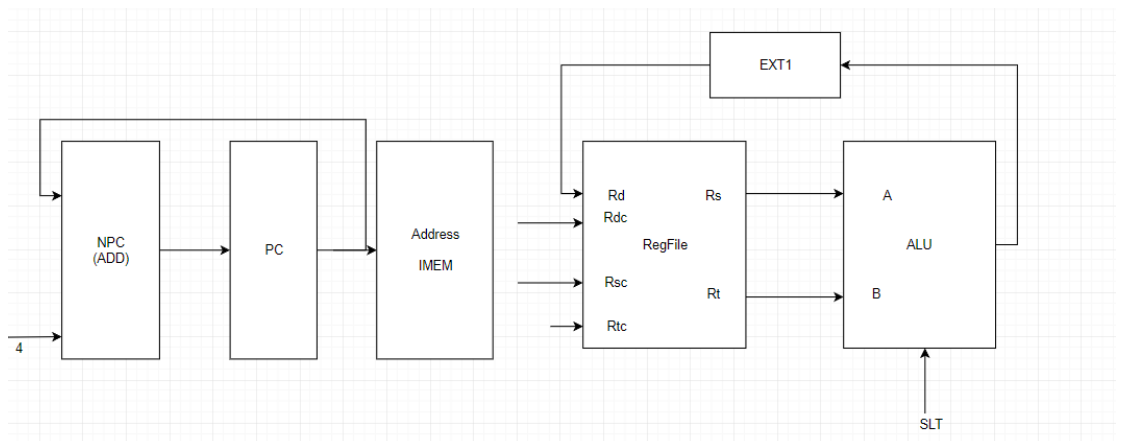
描述: $rd \leftarrow (rs < rt)$

比较在 *rs* 和 *rt* 寄存器中保存的有符号数, 用 **boolean** 值保存结果到 *rd* 寄存器中。如果 *rs* 小于 *rt*, 则结果为 1, 反之结果为 0。算数比较不会引起溢出异常。

所用部件:

编号	指令	PC	NPC	IM	RF Wdata EXT1	ALU A B Rs Rt	EXT16	EXT5	EXT18	DM Data in addr	II A B	ADD A B	EXT1
9	SLT	NPC	PC	PC									ALU

数据通路:



(10) SLTU

格式: SLTU rd, rs, rt

目的: 通过跟立即数无符号小于的比较来记录结果

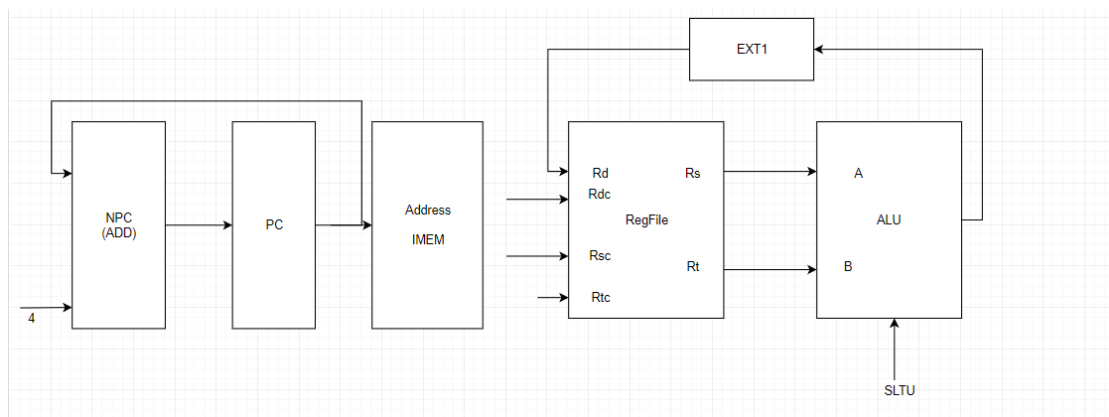
描述: $rd \leftarrow (rs < rt)$

比较在 *rs* 和 *rt* 寄存器中保存的无符号数, 用 **boolean** 值保存结果到 *rd* 寄存器中。如果 *rs* 小于 *rt*, 则结果为 1, 反之结果为 0。算数比较不会引起溢出异常。

所用部件:

编号	指令	PC	NPC	IM	RF Wdata EXT1	ALU A B Rs Rt	EXT16	EXT5	EXT18	DM Data in addr	II A B	ADD A B	EXT1
10	SLTU	NPC	PC	PC									ALU

数据通路:



(11) SLL

格式: SLL rd, rt, sa

目的: 通过数字填充逻辑左移

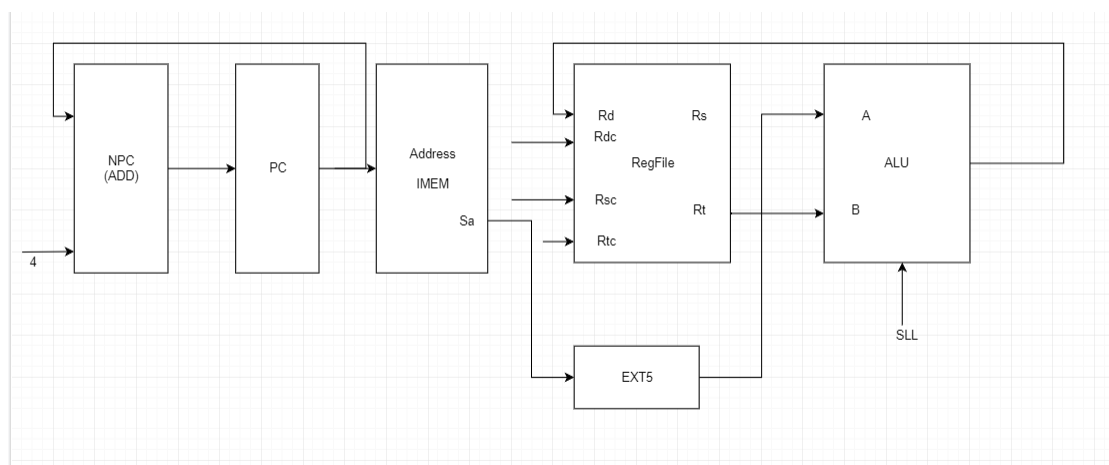
描述: $rd \leftarrow rt \ll sa$

将通用寄存器 rt 的内容左移 sa 位, 空余出来的位置用 0 来填充, 把结果存入 rd 寄存器。

所用部件:

编号	指令	PC	NPC	IM	RF Wdata	ALU		EXT16	EXT5
						A	B		
11	SLL	NPC	PC	PC	ALU	EXT5	Rt		sa

数据通路:



(12) SRL

格式: SRL rd, rt, sa

目的: 通过数字填充逻辑右移

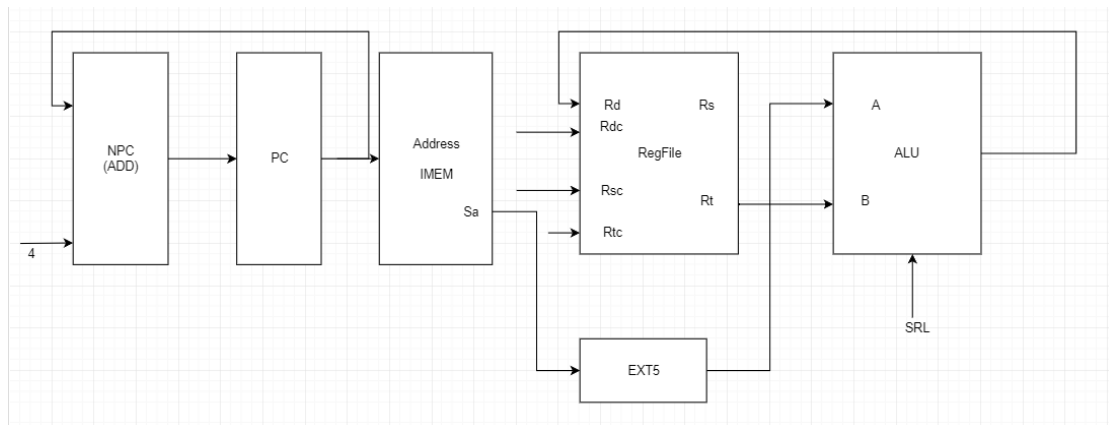
描述: $rd \leftarrow rt \gg sa \text{ (logical)}$

将通用寄存器 rt 中的 32 位内容右移 sa 位, 高位用 0 来填充, 结果存入通用寄存器 rd 。

所用部件:

编号	指令	PC	NPC	IM	RF Wdata	ALU		EXT16	EXT5
						A	B		
12	SRL	NPC	PC	PC	ALU	EXT5	Rt		sa

数据通路:



(13) SRA

格式: SRA rd, rt, sa

目的: 通过数字填充算术右移

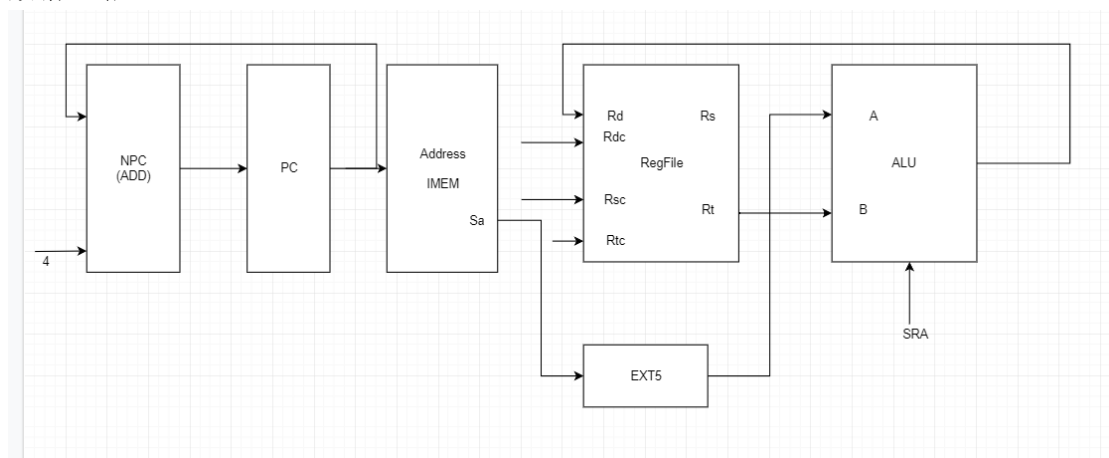
描述: $rd \leftarrow rt \gg sa$ (arithmetic)

将通用寄存器 rt 中的 32 位内容右移 sa 位，高位用 rt[31]来填充，结果存入通用寄存器 rd。

所用部件:

编号	指令	PC	NPC	IM	RF Wdata	ALU		EXT16	EXT5
						A	B		
13	SRA	NPC	PC	PC	ALU	EXT5	Rt		sa

数据通路:



(14) SLLV

格式: SLLV rd, rt, rs

目的: 通过数字填充逻辑左移

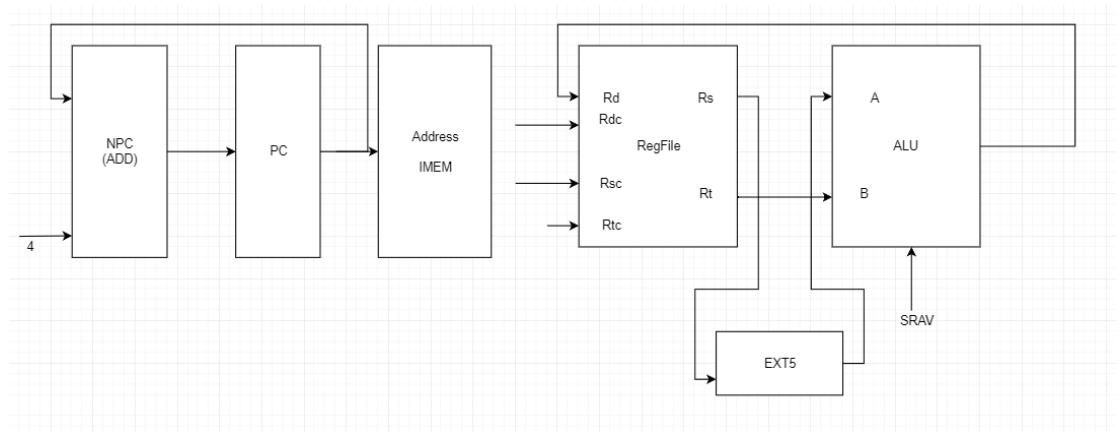
描述: $rd \leftarrow rt \ll rs$

将通用寄存器 rt 的内容逻辑左移，左移的位数保存在 rs 寄存器中，空余出来的位置用 0 来填充，把结果存入 rd 寄存器。

所用部件:

编号	指令	PC	NPC	IM	RF Wdata	ALU		EXT16	EXT5
						A	B		
14	SLLV	NPC	PC	PC	ALU	EXT5	Rt		Rs

数据通路:



(17) JR

格式: JR rs

目的: 使用寄存器的跳转指令

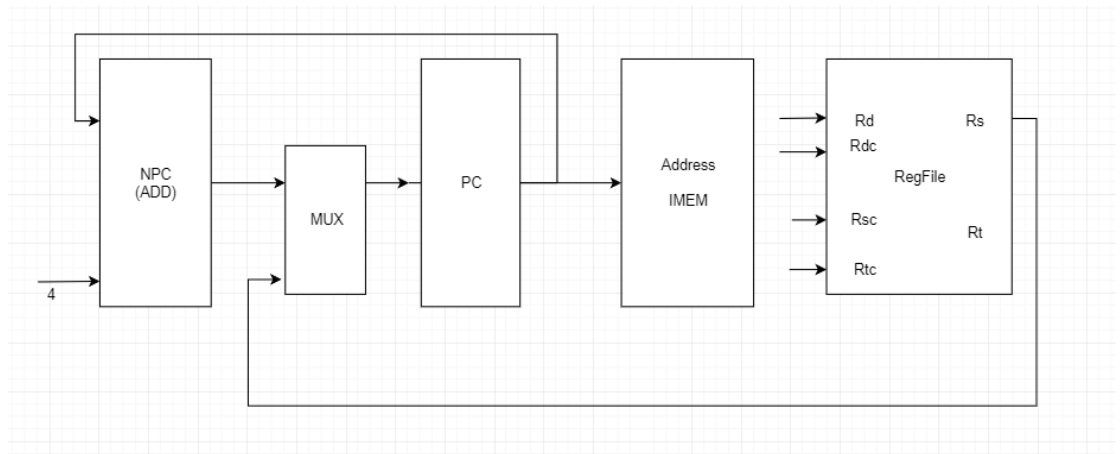
描述: $PC \leftarrow rs$

跳转地址存放在通用寄存器 rs 中，直接跳转到寄存器所存地址。

所用部件:

编号	指令	PC	NPC	IM
17	JR	Rs	PC	PC

数据通路:



(18) ADDI

格式: ADDI rt, rs, immediate

目的: 使 32 位数据与一个立即数相加

描述: $rt \leftarrow rs + \text{immediate}$

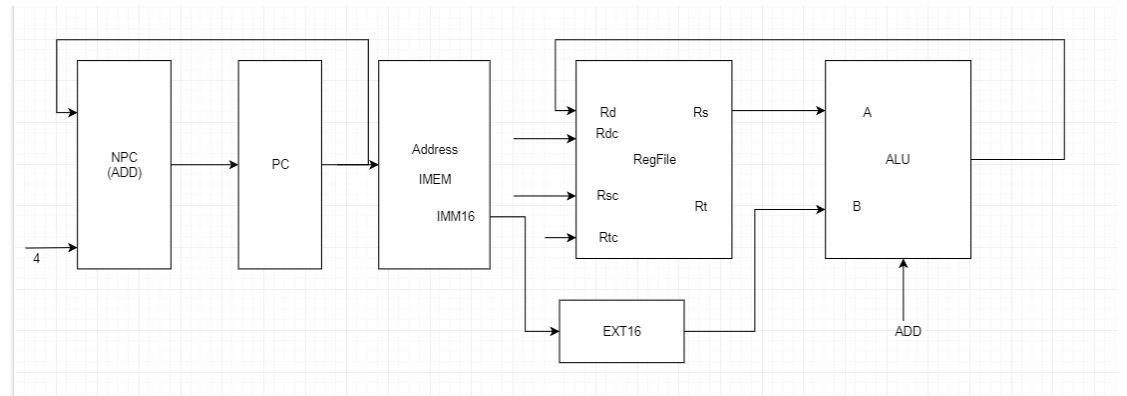
16 位有符号立即数与通用寄存器 rs 中的 32 位数相加产生一个 32 位的数存入目标寄存器 rt。

- 如果发生了溢出，则 rt 不改变并且产生一个溢出的异常。
- 如果相加不溢出，则结果存入目标寄存器 rt。

所用部件:

编号	指令	PC	NPC	IM	RF	Wdata	ALU		EXT16
18	ADDI	NPC	PC	PC	ALU	Rs	A	B	imm16

数据通路:



(19) ADDIU

格式: **ADDIU** *rt*, *rs*, *immediate*

目的: 使 32 位数据与一个立即数相加

描述: $rt \leftarrow rs + immediate$

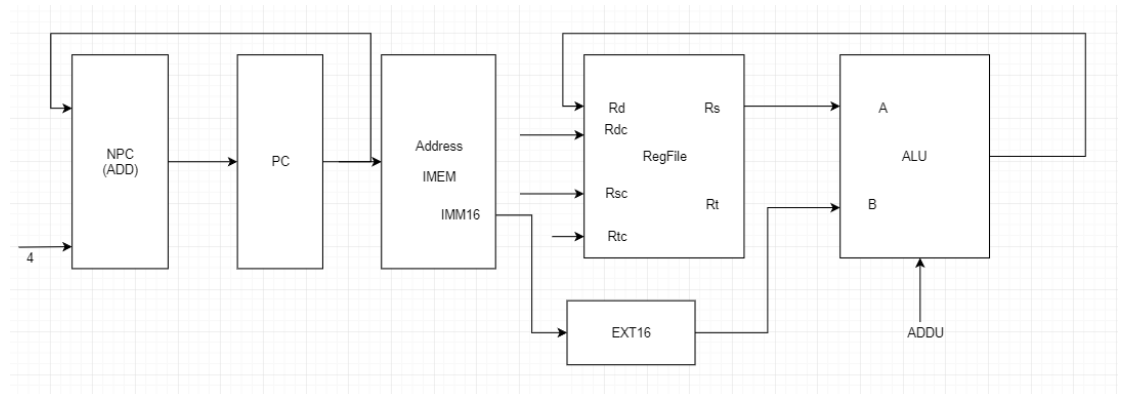
一个 16 位有符号的立即数与通用寄存器 *rs* 中的 32 位数相加产生一个 32 位的数存入目标寄存器 *rt*。

在任何情况下都不会有溢出的异常。

所用部件:

编号	指令	PC	NPC	IM	RF	ALU		EXT16
					Wdata	A	B	
19	ADDIU	NPC	PC	PC	ALU	Rs	EXT16	imm16

数据通路:



(20) ANDI

格式: **ANDI** *rt*, *rs*, *immediate*

目的: 与一个常数做按位逻辑与

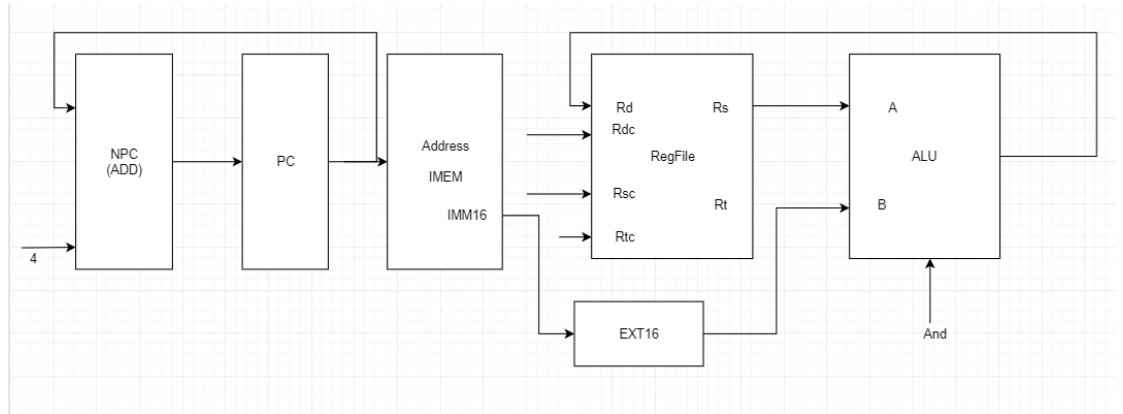
描述: $rt \leftarrow rs \text{ AND } immediate$

将 16 位立即数做 0 扩展后与通用寄存器 *rs* 中的 32 位数据做按位与，将结果存入目标寄存器 *rt*。

所用部件:

编号	指令	PC	NPC	IM	RF	ALU		EXT16
					Wdata	A	B	
20	ANDI	NPC	PC	PC	ALU	Rs	EXT16	imm16

数据通路:



(21) ORI

格式: ORI rt, rs, immediate

目的: 和一个常数做按位逻辑或

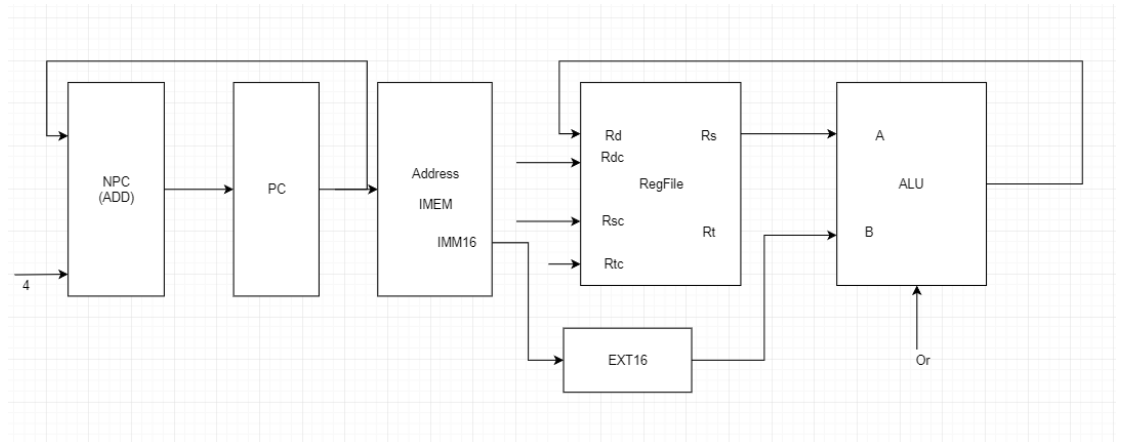
描述: $rt \leftarrow rs \text{ or immediate}$

将通用寄存器 rs 和经过 0 扩展的立即数每一位做按位或操作，将结果存入目标寄存器 rd 中。

所用部件:

编号	指令	PC	NPC	IM	RF Wdata	ALU		EXT16
						A	B	
21	ORI	NPC	PC	PC	ALU	Rs	EXT16	imm16

数据通路:



(22) XORI

格式: XORI rt, rs, immediate

目的: 和一个常数做按位逻辑异或

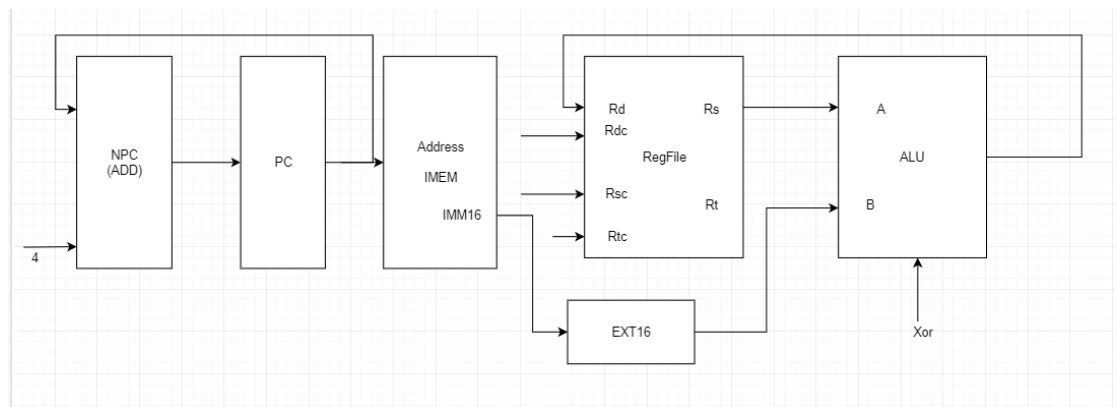
描述: $rt \leftarrow rs \text{ XOR immediate}$

将通用寄存器 rs 和经过 0 扩展的立即数每一位做按位异或操作，将结果存入目标寄存器 rd 中。

所用部件:

编号	指令	PC	NPC	IM	RF Wdata	ALU		EXT16
						A	B	
22	XORI	NPC	PC	PC	ALU	Rs	EXT16	imm16

数据通路:



(23) LW

格式: **LW** *rt*, *offset(base)*

目的: 从内存读取一个字的有符号数据

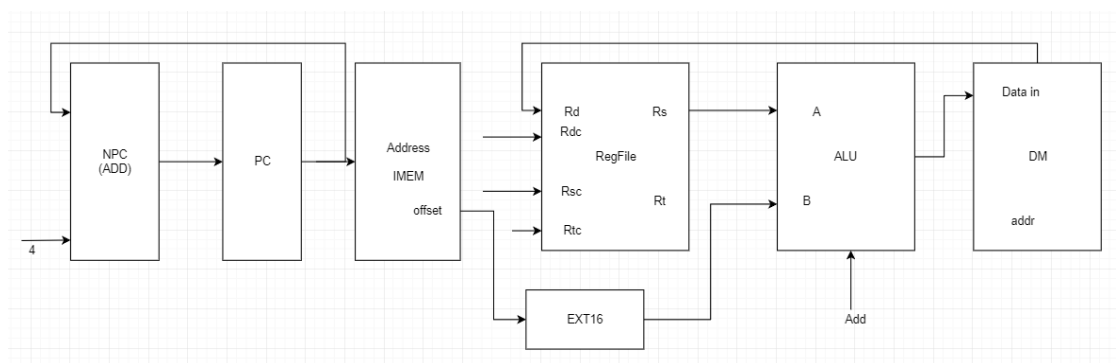
描述: $rt \leftarrow \text{memory}[\text{base} + \text{offset}]$

从内存中基地址加偏移量所得到的准确地址中的内容加载到通用寄存器 *rt* 中。

所用部件:

编号	指令	PC	NPC	IM	RF	ALU		EXT16	EXT5	EXT18	DM	
					Wdata	A	B				Data in	addr
23	LW	NPC	PC	PC	DM	Rs	EXT16	offset			ALU	

数据通路:



(24) SW

格式: **SW** *rt*, *offset(base)*

目的: 存一个字到内存

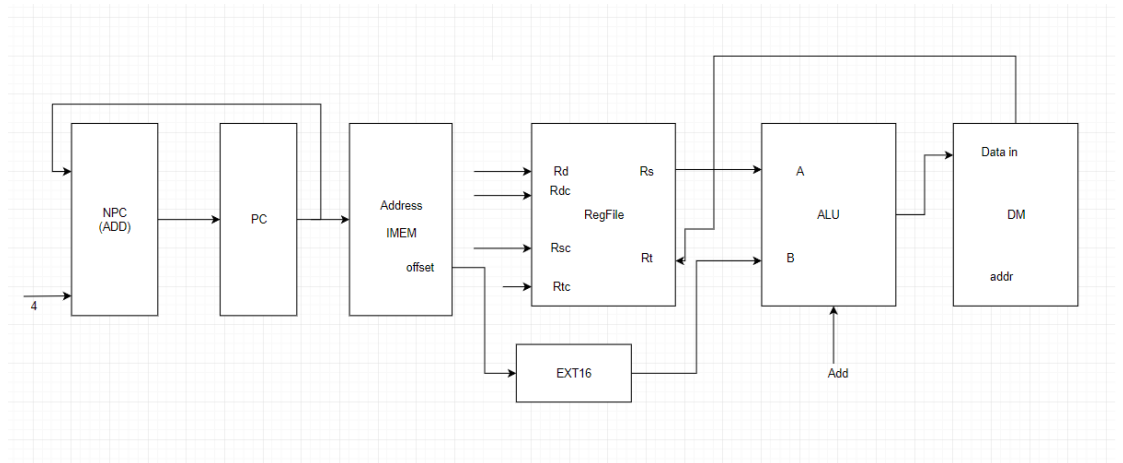
描述: $\text{memory}[\text{base} + \text{offset}] \leftarrow rt$

将通用寄存器 *rt* 中的 32 位数据存入内存中的有效地址，有效地址由基地址和 16 位偏移量相加所得。

所用部件:

编号	指令	PC	NPC	IM	RF	ALU		EXT16	EXT5	EXT18	DM	
					Wdata	A	B				Data in	addr
24	SW	NPC	PC	PC		Rs	EXT16	offset			ALU	Rt

数据通路:



(25) BEQ

格式: **BEQ rs, rt, offset**

目的: 比较通用寄存器的值, 然后做 **pc** 相关的分支跳转

描述: 比较通用寄存器的值, 然后做 **pc** 相关的分支跳转。

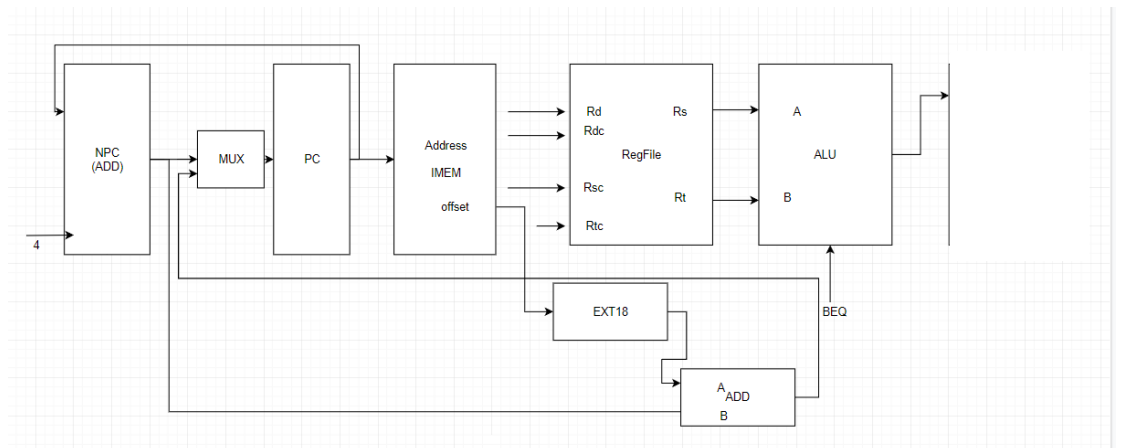
如果 **rs = rt** , 那么将 **offset** 左移两位, 再进行符号扩展到 32 位与当前 **pc** 相加, 形成有效转移地址, 转到该地址。

如果 **rs != rt** , 则继续执行下条指令。

所用部件:

编号	指令	PC	NPC	IM	RF	Wdata	A	B	EXT16	EXT5	EXT18	Data in	DM	addr	A	B	ADD	
25	BEQ	ADD	PC	PC			Rs	Rt			offset						NPC	EXT18

数据通路:



(26) BNE

格式: **BNE rs, rt, offset**

目的: 比较通用寄存器的值, 然后做 **pc** 相关的分支跳转

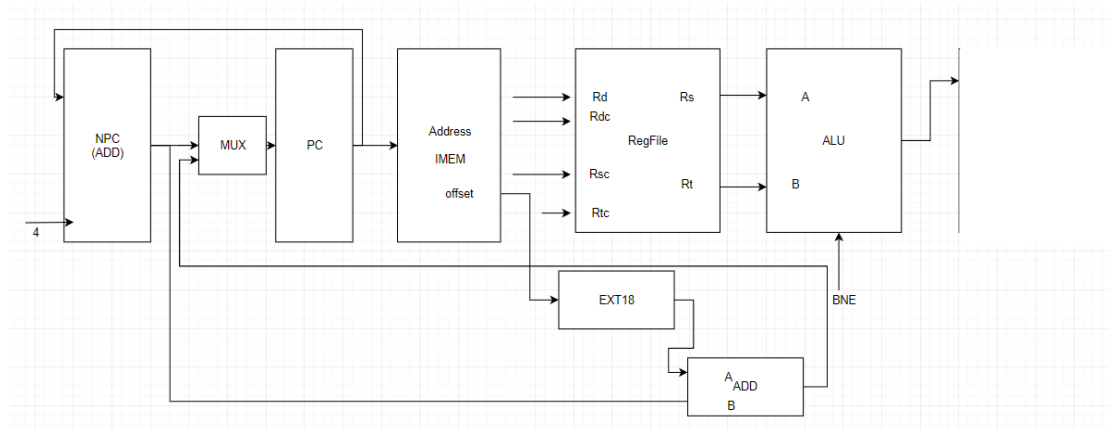
描述: 如果 **rs != rt** , 那么将会跳转到现在 **pc** 与偏移量 **offset** (如果是 16 位需扩展到 18 位) 相加后所得的指令。

如果 **rs = rt** , 则继续执行。

所用部件:

编号	指令	PC	NPC	IM	RF	Wdata	A	B	EXT16	EXT5	EXT18	Data in	DM	addr	A	B	ADD	
26	BNE	ADD	PC	PC			Rs	Rt			offset						NPC	EXT18

数据通路:



(27) SLTI

格式: SLTI rt, rs, immediate

目的: 通过跟立即数小于的比较来记录结果

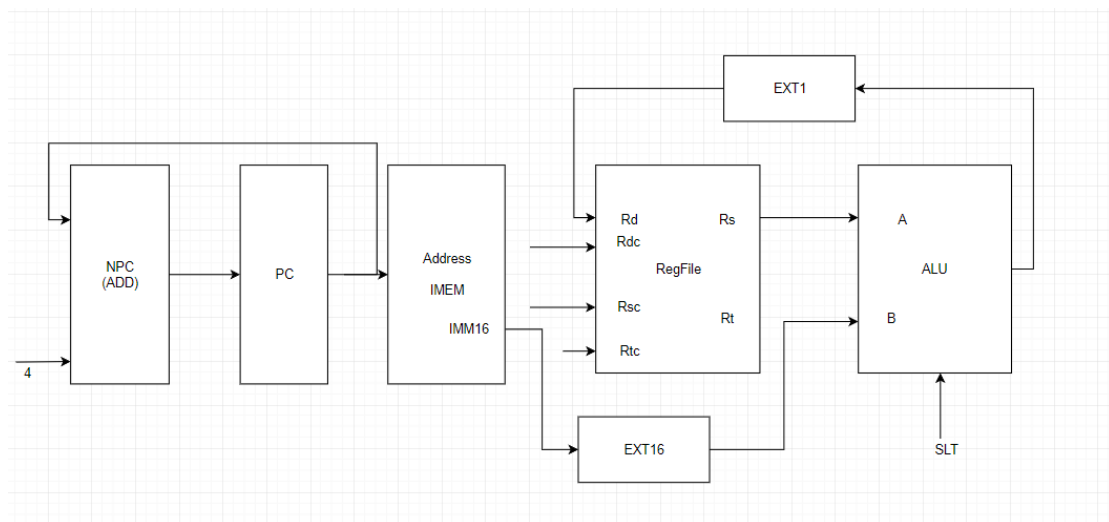
描述: $rt \leftarrow (rs < \text{immediate})$

比较在 rs 和经过符号扩展的 16 位立即数, 用 boolean 值保存结果到 rd 寄存器中。如果 rs 小于 rt, 则结果为 1, 反之结果为 0。算数比较不会引起溢出异常。

所用部件:

编号	指令	PC	NPC	IM	RF Wdata	ALU A	ALU B	EXT16	EXT5	EXT18	DM Data in	DM addr	II A	II B	ADD A	ADD B	EXT1
27	SLTI	NPC	PC	PC	EXT1	EXT16	Rt	imm16									ALU

数据通路:



(28) SLTIU

格式: SLTIU rt, rs, immediate

目的: 通过跟立即数无符号小于的比较来记录结果

描述: $rt \leftarrow (rs < \text{immediate})$

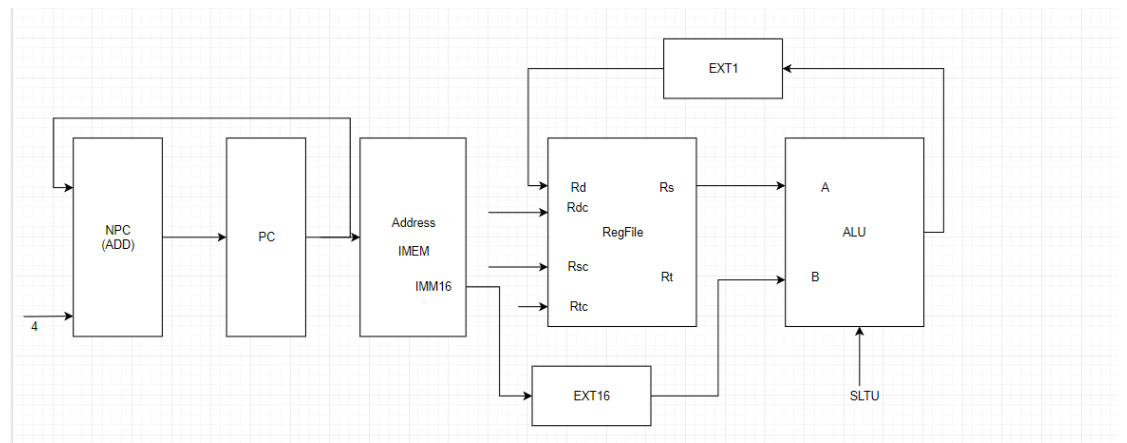
比较在 rs 和经过 0 扩展的 16 位立即数, 用 boolean 值保存结果到 rd 寄存器中。

如果 rs 小于 rt, 则结果为 1, 反之结果为 0。算数比较不会引起溢出异常。

所用部件:

编号	指令	PC	NPC	IM	RF Wdata	ALU A	ALU B	EXT16
28	SLTIU	NPC	PC	PC	EXT1	EXT16	Rt	imm16

数据通路:



(29) LUI

格式: LUI rt, immediate

目的: 把一个立即数载入到寄存器的高位, 低位补 0

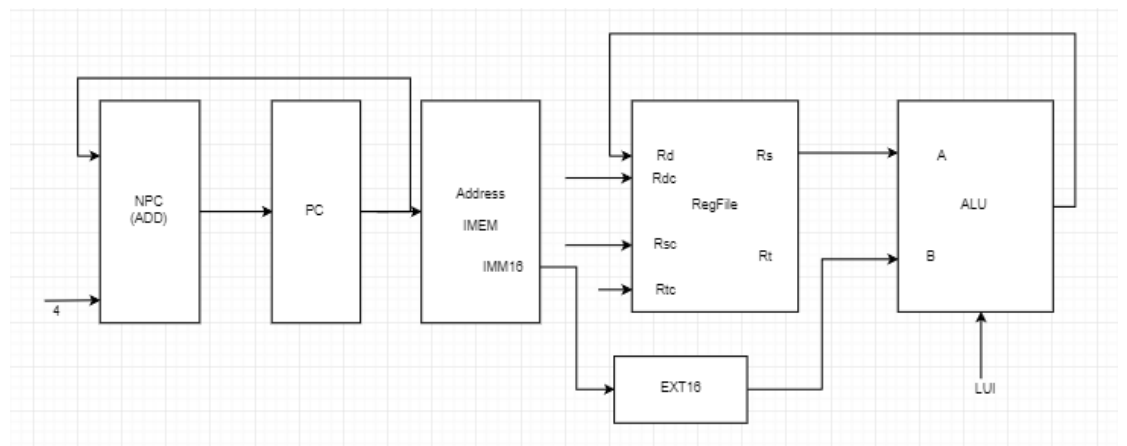
描述: $rt \leftarrow \text{immediate} \parallel 016$

将一个 16 位的立即数载入到通用寄存器 rt 的高位, 低 16 位补 0。

所用部件:

编号	指令	PC	NPC	IM	RF Wdata	ALU		EXT16
29	LUI	NPC	PC	PC	ALU	A	B	imm16

数据通路:



(30) J

格式: J target

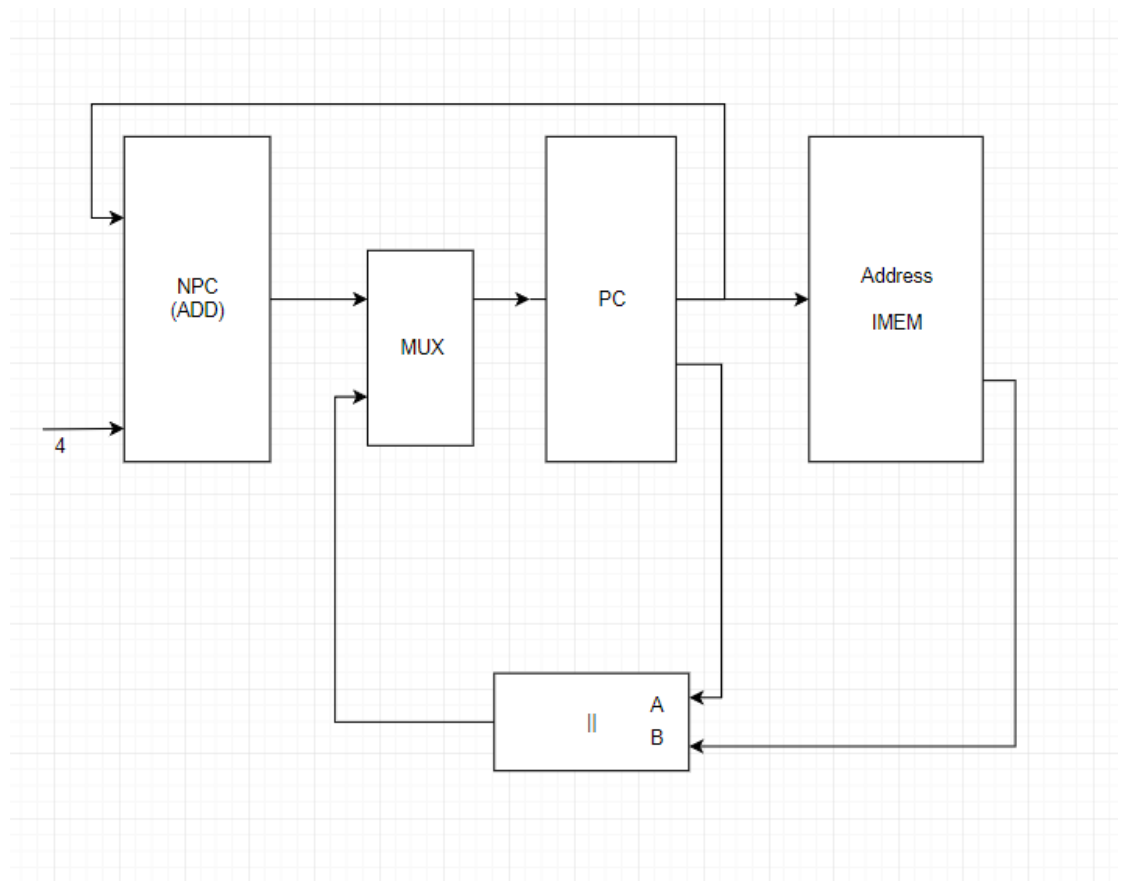
目的: 在 256MB 的范围内跳转

描述: 该指令无条件跳转到一个绝对地址, instr_index 有 26 位, 在左移过后访问空间能达到 228B, 既是 256M。

所用部件:

编号	指令	PC	NPC	IM	RF Wdata	ALU		EXT16	EXT5	EXT18	DM Data in	addr		
30	J		PC	PC		A	B						A	B

数据通路:



(31) JAL

格式: JAL target

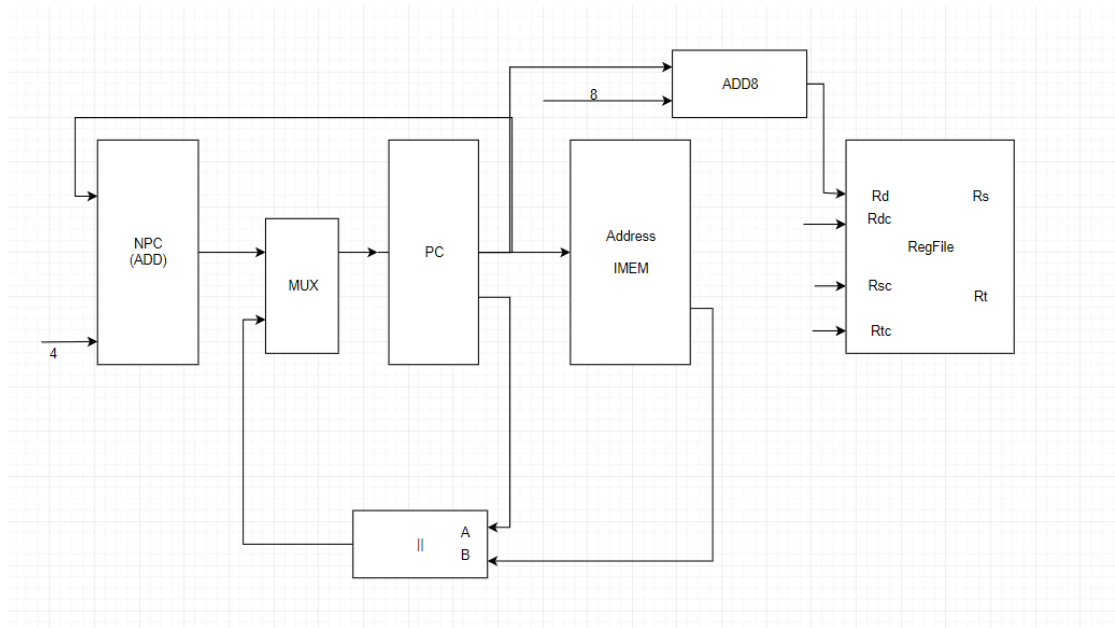
目的: 在 256MB 范围内执行一个过程调用

描述: 在跳转到指定地址执行子程序调用的同时, 在 31 号寄存器中存放返回地址 (当前地址后的第二条指令地址)。

所用部件:

编号	指令	PC	NPC	IM	RF Wdata ADD8	ALU A B	EXT16	EXT5	EXT18	DM Data in addr	IM A B	ADD A B	EXT1	ADD8 PC
31	JAL		PC	PC	ADD8						PC[31:28] IM[25:0]			PC

数据通路:

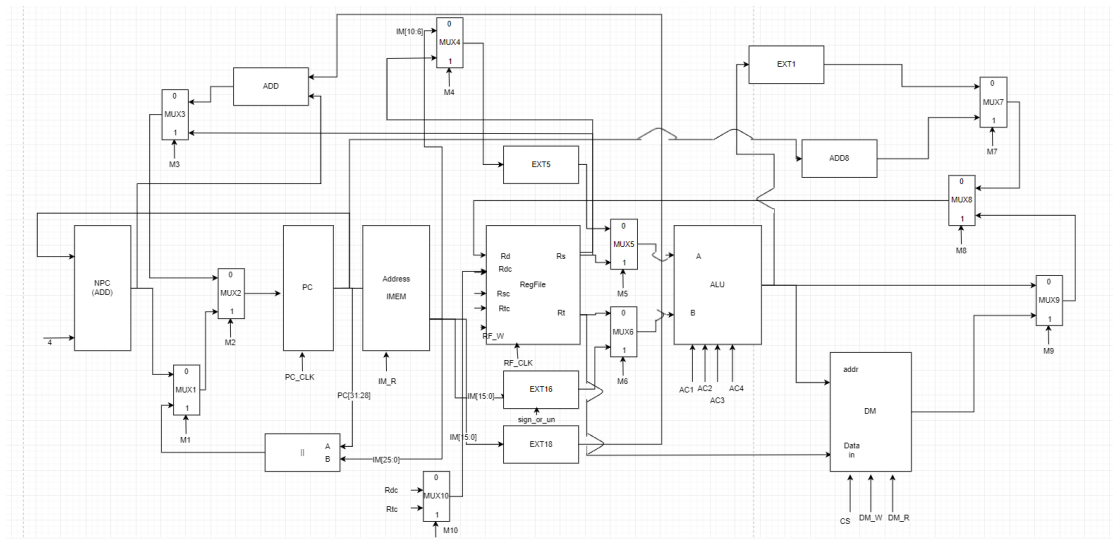


总体部件情况：

编号	指令	PC	NPC	IM	RF Wdata	A	ALU B	EXT16	EXT5	EXT18	Data in	DM addr	A	II B	A	ADD B	EXT1	ADD8
1	ADDU	NPC	PC	PC	ALU	Rs	Rt											
2	ADD	NPC	PC	PC	ALU	Rs	Rt											
3	SUB	NPC	PC	PC	ALU	Rs	Rt											
4	SUBU	NPC	PC	PC	ALU	Rs	Rt											
5	AND	NPC	PC	PC	ALU	Rs	Rt											
6	OR	NPC	PC	PC	ALU	Rs	Rt											
7	XOR	NPC	PC	PC	ALU	Rs	Rt											
8	NOR	NPC	PC	PC	ALU	Rs	Rt											
9	SLT	NPC	PC	PC	EXT1	Rs	Rt											
10	SLTU	NPC	PC	PC	EXT1	Rs	Rt										ALU	ALU
11	SLL	NPC	PC	PC	ALU	EXT5	Rt		sa									
12	SRL	NPC	PC	PC	ALU	EXT5	Rt		sa									
13	SRA	NPC	PC	PC	ALU	EXT5	Rt		sa									
14	SLLV	NPC	PC	PC	ALU	EXT5	Rt		Rs									
15	SRLV	NPC	PC	PC	ALU	EXT5	Rt		Rs									
16	SRAV	NPC	PC	PC	ALU	EXT5	Rt		Rs									
17	JR	Rs	PC	PC														
18	ADDI	NPC	PC	PC	ALU	Rs	EXT16	imm16										
19	ADDIU	NPC	PC	PC	ALU	Rs	EXT16	imm16										
20	ANDI	NPC	PC	PC	ALU	Rs	EXT16	imm16										
21	ORI	NPC	PC	PC	ALU	Rs	EXT16	imm16										
22	XORI	NPC	PC	PC	ALU	Rs	EXT16	imm16										
23	LW	NPC	PC	PC	DM	Rs	EXT16	offset				ALU						
24	SW	NPC	PC	PC		Rs	EXT16	offset			ALU	Rt						
25	BEQ	ADD	PC	PC		Rs	Rt			offset					NPC	EXT18		
26	BNE	ADD	PC	PC		Rs	Rt			offset					NPC	EXT18		
27	SLTI	NPC	PC	PC	EXT1	Rs	EXT16	imm16									ALU	
28	SLTIU	NPC	PC	PC	EXT1	Rs	EXT16	imm16									ALU	
29	LUI	NPC	PC	PC	ALU	Rs	EXT16	imm16										
30	J	II	PC	PC									PC[31:28] IM[25:0]					
31	JAL	II	PC	PC	ADD8								PC[31:28] IM[25:0]					PC

2.总体数据通路和控制信号表

总体数据通路：



CPU 控制信号表

	Rsc	Rtc	Rdc	AC[3:0]	M1	M2	M3	M4	M5	M6	M7	M8	M9	PC_CLK	IM_R	RF_CLK	RF_W	CS	DM_W	DM_R	M10
ADDU	IM[25:21]	IM[20:16]	IM[15:11]	00000	0	1			1	0		1	0	1	1	1	1	0	0	0	0
ADD	IM[25:21]	IM[20:16]	IM[15:11]	00001	0	1			1	0		1	0	1	1	1	1	0	0	0	0
SUB	IM[25:21]	IM[20:16]	IM[15:11]	00010	0	1			1	0		1	0	1	1	1	1	0	0	0	0
SUBU	IM[25:21]	IM[20:16]	IM[15:11]	00011	0	1			1	0		1	0	1	1	1	1	0	0	0	0
AND	IM[25:21]	IM[20:16]	IM[15:11]	00100	0	1			1	0		1	0	1	1	1	1	0	0	0	0
OR	IM[25:21]	IM[20:16]	IM[15:11]	00101	0	1			1	0		1	0	1	1	1	1	0	0	0	0
XOR	IM[25:21]	IM[20:16]	IM[15:11]	00110	0	1			1	0		1	0	1	1	1	1	0	0	0	0
NOR	IM[25:21]	IM[20:16]	IM[15:11]	00111	0	1			1	0		1	0	1	1	1	1	0	0	0	0
SLT	IM[25:21]	IM[20:16]	IM[15:11]	10010	0	1			1	0	0	0	0	1	1	1	1	0	0	0	0
SLTU	IM[25:21]	IM[20:16]	IM[15:11]	10011	0	1			1	0	0	0	0	1	1	1	1	0	0	0	0
SLL	0	IM[20:16]	IM[15:11]	01000	0	1		0	0	0		1	0	1	1	1	1	0	0	0	0
SRL	0	IM[20:16]	IM[15:11]	01001	0	1		0	0	0		1	0	1	1	1	1	0	0	0	0
SRA	0	IM[20:16]	IM[15:11]	01010	0	1		0	0	0		1	0	1	1	1	1	0	0	0	0
SRLV	IM[25:21]	IM[20:16]	IM[15:11]	01011	0	1		1	0	0		1	0	1	1	1	1	0	0	0	0
SRLV	IM[25:21]	IM[20:16]	IM[15:11]	01110	0	1		1	0	0		1	0	1	1	1	1	0	0	0	0
SRAV	IM[25:21]	IM[20:16]	IM[15:11]	01111	0	1		1	0	0		1	0	1	1	1	1	0	0	0	0
JR	IM[25:21]	0	0	0	0	0	1							1							
ADDI	IM[25:21]	IM[20:16]		00001	0	1			1	1		1	0	1	1	1	1	0	0	0	1
ADDIU	IM[25:21]	IM[20:16]		00000	0	1			1	1		1	0	1	1	1	1	0	0	0	1
ANDI	IM[25:21]	IM[20:16]		00100	0	1			1	1		1	0	1	1	1	1	0	0	0	1
ORI	IM[25:21]	IM[20:16]		00101	0	1			1	1		1	0	1	1	1	1	0	0	0	1
XORI	IM[25:21]	IM[20:16]		00110	0	1			1	1		1	0	1	1	1	1	0	0	0	1
LW	IM[25:21]	IM[20:16]	IM[20:16]	10001	0	1			1	1		1	0	1	1	1	1	0	1	1	1
SW	IM[25:21]	IM[20:16]	IM[20:16]	10000	0	1			1	1		1	0	1	1	1	0	1	1	0	1
BEQ	IM[25:21]	IM[20:16]		00010	0	0	0		1	0				1	1	0	0	0	0	0	0
BNE	IM[25:21]	IM[20:16]		00011	0	0	0		1	0				1	1	0	0	0	0	0	0
SLTI	IM[25:21]	IM[20:16]	IM[20:16]	10010	0	1			1	1	0	0		1	1	1	1	0	0	0	1
SLTIU	IM[25:21]	IM[20:16]	IM[20:16]	10011	0	1			1	1	0	0		1	1	1	1	0	0	0	1
LUI	IM[25:21]	0	IM[20:16]	01101	0	1			1	1		1	0	1	1	1	1	0	0	0	1
J					1	1								1	1	0	0	0	0	0	
JAL					1	1						1	0	1	1	1	1	0	0	0	0

3. 编写代码并下板

详见 仿真过程

三、 总体架构部件的解释说明

1、CPU 总体结构部件的解释说明

M[10:1]: 多路选择器

RF: 寄存器内容存储

ALU: 主要运算部件, 兼顾标志位的改变

EXT16:16 位拓展

EXT5:5 位拓展

EXT18: 18 位拓展

II: 主要用作 J,JAL 命令

ADD8:用作 JAL 命令

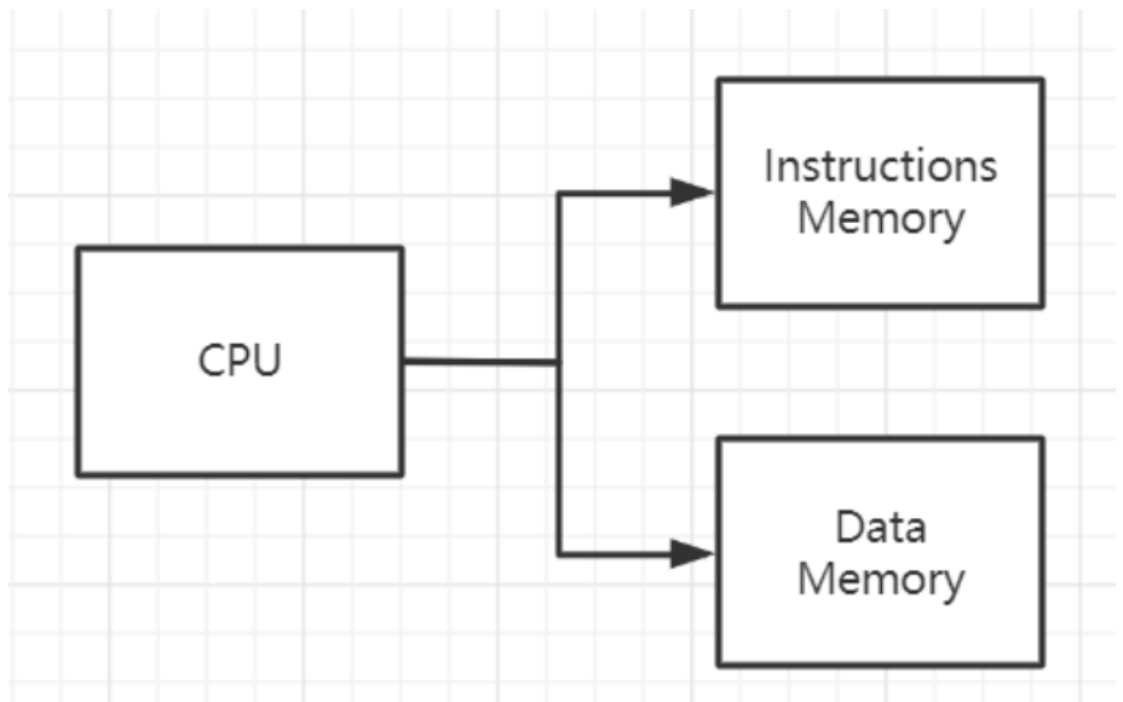
NPC:PC+4

IM:取指令

四、 实验仿真过程

1、 CPU 的仿真过程

本实验采取的是哈佛结构



首先绘制出数据通路，确定好 CPU 和 DM,IM 的关联，然后编写总体架构代码，结构如下所示：

```

module top_module
    IMEM imemory(
        .addr(temp_im_addr/4),
        .instr(instr)
    );
    assign temp_im_addr = pc - 32'h0040_0000;
    assign im_addr = temp_im_addr;

    DMEM dmemory(
        .clk(clk), .ena(1'b1), .DM_W(dw), .DM_R(dr), .DM_Addr(dm_addr[10:0]), .DM_WData(w_data),
        .DM_RData(r_data)
    );
    cpu31(
        .clk(clk), .reset(rst), .inst(instr), .rdata(r_data),
        .DM_CS(dena), .DM_W(dw), .DM_R(dr), .wdata(w_data), .pc(pc), .addr(res)
    );
endmodule
  
```

CPU 编写，我采用的是逐层实现，从底层一步步实现到总体。

首先通过 `input_decode.v`，将命令转换为容易处理的形式，主要代码如下：

```

1. begin
2.     casez(t)
3.         12'b000000100000 :i = 32'b00000000000000000000000000000001;
4.         12'b000000100001  :i = 32'b00000000000000000000000000000010;
5.         12'b000000100010  :i = 32'b000000000000000000000000000000100;
  
```



```

8.    assign M6 = i[17]|i[18]|i[19]|i[20]|i[21]|i[22]|i[23]|i[26]|i[27]|i[28];

9.    assign M7 = i[30];
10.   assign M8=i[8]|i[9]|i[16]|i[26]|i[27]|i[22];
11.   assign M9 =i[22];
12.   assign M10 = i[17] | i[18] | i[19] | i[20] | i[21] | i[22] | i[23] | i[26
    ] | i[27] | i[28];
13.
14.   assign ALUC[0] = i[0]|i[3]|i[5]|i[7]|i[9]|i[11]|i[15]|i[17]|i[20]|i[22]|i
    [23]|i[27]|i[28];
15.   assign ALUC[1]=i[2]|i[3]|i[6]|i[7]|i[8]|i[9]|i[12]|i[14]|i[15]|i[21]|i[24
    ]|i[25]|i[26]|i[27];
16.   assign ALUC[2]=i[4]|i[5]|i[6]|i[7]|i[14]|i[15]|i[19]|i[20]|i[21]|i[28];
17.   assign ALUC[3]=i[10]|i[11]|i[12]|i[13]|i[14]|i[15]|i[28];
18.   assign ALUC[4]=i[8]|i[9]|i[22]|i[23]|i[26]|i[27];
19.
20.   assign RF_W = ~(i[16] | i[23] | i[24] | i[25] | i[29]);
21.   assign RF_CLK = ~clk;
22.
23.   assign DM_W = i[23];
24.   assign DM_R = i[22];
25.   assign DM_CS = i[22] | i[23];
26.   assign C_EXT16 = ~(i[19] | i[20] | i[21]);

```

编写 ALU:

```

1.  //instruction set
2.      parameter ADDU = 5'b 00000;
3.      parameter ADD = 5'b 00001;
4.      parameter SUB = 5'b 00010;
5.      parameter SUBU = 5'b 00011;
6.      parameter AND = 5'b 00100;
7.      parameter OR = 5'b 00101;
8.      parameter XOR = 5'b 00110;
9.      parameter NOR = 5'b 00111;
10.     parameter SLL = 5'b 01000;
11.     parameter SRL = 5'b 01001;
12.     parameter SRA = 5'b 01010;
13.     parameter SLLV = 5'b 01011;
14.
15.     parameter LUI = 5'b 01101;
16.     parameter SRLV = 5'b 01110;
17.     parameter SRAV = 5'b 01111;

```

```

18.
19.     parameter SLT = 5'b 10010;
20.     parameter SLTU = 5'b 10011;
21.     parameter LW = 5'b 10001;//SW
22.
23.
24.     //easy to calculate
25.     wire signed [31:0]aSign,bSign ;
26.     assign aSign =a;
27.     assign bSign =b;
28.
29.     //store res
30.     //use [32:0] instead of [31:0] use [32] to change CF SF OF ZF
31.     reg [32:0]res;
32.
33.     always @(*) begin
34.         case(aluc)
35.             ADD:    res<=aSign+bSign;
36.             LW:     begin res<=aSign+bSign;
37.
38.                 end
39.             ADDU:   res<=a+b;
40.
41.             SUB:    res<=aSign-bSign;
42.             SLT:    res<=aSign<bSign?32'b1:32'b0;
43.
44.             SUBU:   res<=a-b;
45.             SLTU:   res<=a<b?1:0;
46.
47.             AND:    res<=a&b;
48.             OR:     res<=a|b;
49.             XOR:    res<=a^b;
50.             NOR:    res<=~(a|b);
51.
52.             SLL:    res<=b<<a;
53.             SRL:    res<=b>>a;
54.             SRA:    res<=bSign>>>aSign;
55.             SLLV:   res<=b<<a[4:0];
56.             SRLV:   res<=b>>a[4:0];
57.             SRAV:   res<=bSign>>>aSign[4:0];
58.
59.             LUI:    res<={b[15:0],16'b0};
60.
61.             default:

```

```

62.         res<=32'b0;
63.     endcase
64. end
65.
66. //process output
67. assign r=res[31:0];
68.
69. //process ZF,OF,CF,SF
70. assign zero =(res==32'b0)?1'b1:1'b0;//ZF
71. //only add sub is not process 明天写
72. assign overflow=(aluc==ADD)?(res[32]?((aSign>0&&bSign>0)?1'b1:1'b0):((aSign<0&&bSign<0)?1'b1:1'b0)):((aluc==SUB)?(res[32]?((aSign>0&&bSign<0)?1'b1:1'b0):((aSign<0&&bSign>0)?1'b1:1'b0)):1'b0);//OF
73. assign carry = (aluc==ADDU|aluc==SUBU|aluc==SLTU|aluc==SRA|aluc==SRL|aluc==SLL)?res[32]:1'bz;//CF
74. assign negative=(aluc==SUB?(aSign<bSign):((aluc==SUBU)?(a<b):1'b0));//SF

```

编写其余 CPU 部件，详细代码详见文档末尾，相关部件大多只有一两行。

其余部件包括：ADD,ADD8,EXT5,EXT18,EXT16,II,MUX,MUX5,PCreg, RegFile

编写代码结束后就可以开始仿真：

Testbench 代码：

```

1. `timescale 1ns / 1ps
2. module test_cpu(
3. );
4.     reg clk, rst;
5.     wire [31:0] inst, pc;
6.     reg [31:0] cnt;
7.     wire[10:0] dma, ima;
8.     integer file_open;
9.     initial begin
10.         clk = 1'b0;
11.         rst = 1'b1;
12.         #2 rst=1'b0;
13.         cnt = 0;
14.     end
15.
16.     always begin
17.         #1 clk = !clk;
18.     end
19. initial begin

```



```

20. file_open = $fopen("D:/shuzidianlu/Xilinx/CPU/CPU.sim/sim_1/behav/output.txt", "w+");//目标存放点
21. if (!file_open) begin
22.
23. $display("can not open \"888.txt\"");
24.
25. $finish;
26.
27. end
28. end
29.     always @(negedge clk) begin
30.         cnt <= cnt + 1'b1;
31.         if (cnt < 8'h99 - 1&&cnt>=1'b1 ) begin // 可自我调整, 这是所有 txt 测试文件都能够容纳的数据, 在此时钟下, 为 6000 ns
32.
33. $fdisplay(file_open, "pc: %h", sc.pc);
34.         $fdisplay(file_open, "instr: %h", sc.inst);
35.         // $fdisplay(file_open, "aluc: %h,a: %h,b:%h,res:%h", sc.cpu31.cpu_alu.aluc,sc.cpu31.cpu_alu.a,sc.cpu31.cpu_alu.b,sc.cpu31.cpu_alu.res);
36.         // $fdisplay(file_open, "wdata: %h,MUX9:%h,D_ALU:%h", sc.cpu31.D_Mux8,sc.cpu31.D_Mux9,sc.cpu31.D_ALU);
37.
38.         $fdisplay(file_open, "regfile0: %h", sc.cpu31.cpu_ref.array_reg[0]);
39.
40.         $fdisplay(file_open, "regfile1: %h", sc.cpu31.cpu_ref.array_reg[1]);
41.         $fdisplay(file_open, "regfile2: %h", sc.cpu31.cpu_ref.array_reg[2]);
42.         $fdisplay(file_open, "regfile3: %h", sc.cpu31.cpu_ref.array_reg[3]);
43.         $fdisplay(file_open, "regfile4: %h", sc.cpu31.cpu_ref.array_reg[4]);
44.         $fdisplay(file_open, "regfile5: %h", sc.cpu31.cpu_ref.array_reg[5]);
45.         $fdisplay(file_open, "regfile6: %h", sc.cpu31.cpu_ref.array_reg[6]);
46.         $fdisplay(file_open, "regfile7: %h", sc.cpu31.cpu_ref.array_reg[7]);
47.         $fdisplay(file_open, "regfile8: %h", sc.cpu31.cpu_ref.array_reg[8]);
48.         $fdisplay(file_open, "regfile9: %h", sc.cpu31.cpu_ref.array_reg[9]);
49.         $fdisplay(file_open, "regfile10: %h", sc.cpu31.cpu_ref.array_reg[10]);

```

```
49.          $fdisplay(file_open, "regfile11: %h", sc.cpu31.cpu_ref.arr
    ay_reg[11]);
50.          $fdisplay(file_open, "regfile12: %h", sc.cpu31.cpu_ref.arr
    ay_reg[12]);
51.          $fdisplay(file_open, "regfile13: %h", sc.cpu31.cpu_ref.arr
    ay_reg[13]);
52.          $fdisplay(file_open, "regfile14: %h", sc.cpu31.cpu_ref.arr
    ay_reg[14]);
53.          $fdisplay(file_open, "regfile15: %h", sc.cpu31.cpu_ref.arr
    ay_reg[15]);
54.          $fdisplay(file_open, "regfile16: %h", sc.cpu31.cpu_ref.arr
    ay_reg[16]);
55.          $fdisplay(file_open, "regfile17: %h", sc.cpu31.cpu_ref.arr
    ay_reg[17]);
56.          $fdisplay(file_open, "regfile18: %h", sc.cpu31.cpu_ref.arr
    ay_reg[18]);
57.          $fdisplay(file_open, "regfile19: %h", sc.cpu31.cpu_ref.arr
    ay_reg[19]);
58.          $fdisplay(file_open, "regfile20: %h", sc.cpu31.cpu_ref.arr
    ay_reg[20]);
59.          $fdisplay(file_open, "regfile21: %h", sc.cpu31.cpu_ref.arr
    ay_reg[21]);
60.          $fdisplay(file_open, "regfile22: %h", sc.cpu31.cpu_ref.arr
    ay_reg[22]);
61.          $fdisplay(file_open, "regfile23: %h", sc.cpu31.cpu_ref.arr
    ay_reg[23]);
62.          $fdisplay(file_open, "regfile24: %h", sc.cpu31.cpu_ref.arr
    ay_reg[24]);
63.          $fdisplay(file_open, "regfile25: %h", sc.cpu31.cpu_ref.arr
    ay_reg[25]);
64.          $fdisplay(file_open, "regfile26: %h", sc.cpu31.cpu_ref.arr
    ay_reg[26]);
65.          $fdisplay(file_open, "regfile27: %h", sc.cpu31.cpu_ref.arr
    ay_reg[27]);
66.          $fdisplay(file_open, "regfile28: %h", sc.cpu31.cpu_ref.arr
    ay_reg[28]);
67.          $fdisplay(file_open, "regfile29: %h", sc.cpu31.cpu_ref.arr
    ay_reg[29]);
68.          $fdisplay(file_open, "regfile30: %h", sc.cpu31.cpu_ref.arr
    ay_reg[30]);
69.          $fdisplay(file_open, "regfile31: %h", sc.cpu31.cpu_ref.arr
    ay_reg[31]);
70.
71.          end
```

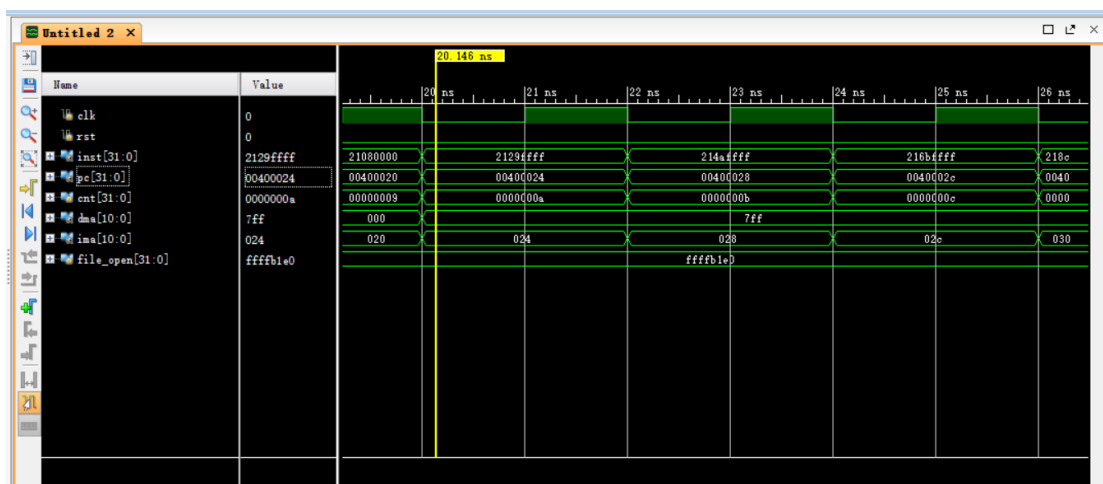
```

72.         if(cnt==400) begin
73.
74.             $fclose(file_open);
75.
76.         end
77.     end
78.     sccomp_dataflow sc(
79.         .clk(clk),
80.         .rst(rst),
81.         .inst(inst),
82.         .pc(pc),
83.         .dm_addr(dma), .im_addr(ima)
84.     );
85. endmodule

```

五、 实验仿真的波形图及某时刻寄存器值的物理意义

1、仿真波形图及某时刻寄存器值的物理意义



Pc 是当前指令值

Inst 是 IM 根据 ima 取出的对应的指令

Dma 是写入 DMEM 的地址

Ima 是将 PC-32'H0040_0000 后，适配本地 IMEM 的地址

六、 实验验算数学模型及算法程序

比萨塔摔鸡蛋游戏模型。两个同学在可变换层数的比萨塔上摔鸡蛋，一个同学秘密设定同一批鸡蛋耐摔值；另一个同学在指定层高的比萨塔拿着鸡蛋往下摔，用最少的摔次数和摔破的鸡蛋数求出鸡蛋的耐摔值。假定在耐摔值的楼层及其下面楼层，鸡蛋摔不破，可以重复使用，否则鸡蛋摔破。要求模型的算法输出包括：摔的总次数、摔的总鸡蛋数、最后摔的鸡蛋是否摔破。请使用 MIPS 指令汇编设计该验证模型的算法，并利用编译器生成 MIPS 指令集可执行目标程序。

对应算法：

```

1. .data
2. floor: .word 0
3. naishuaivalue: .word 0
4.
5. myMessage: .ascii "input the value of floor\n\0"
6. myMessage2: .ascii "input the egg firm\n\0"
7. mess: .ascii "trytime:\0"
8. mess1: .ascii "\nbrokenegg:\0"
9. mess2: .ascii "\nthe end egg is break:\0"
10. mess3: .ascii "egg firm value:\0"
11. yes: .ascii "Yes!\n\0"
12. no: .ascii "No!\n\0"
13. .text
14. main:
15. li $v0,4
16. la $a0,myMessage
17. syscall
18. li $v0,5
19. syscall
20. sw $v0,floor
21. li $v0,4
22. la $a0,myMessage2
23. syscall
24. li $v0,5
25. syscall
26. sw $v0,naishuaivalue
27.
28. label1:
29. la $s0,floor
30. lw $t0,($s0) #存储 floor 的数值
31. la $s0,naishuaivalue
32. lw $t7,($s0)
33. inital:
34. addi $t1,$zero,0
35. addi $t2,$zero,0
36. addi $t3,$zero,0
37. addi $t4,$zero,1
38. add $t5,$zero,$t0
39. addi $t6,$zero,0
40. addi $s2,$zero,1
41.
42. while:
43. add $t6,$t4,$t5
44. srl $t6,$t6,1

```

```
45. addi $t2,$t2,1
46. slt $s1,$t7,$t6
47. beq $s1,$s2,weak
48. j strong
49.
50. strong:
51. beq $t4,$t6,special
52. addi $t4,$t6,0
53. add $t3,$zero,1
54. j judge
55.
56. weak:
57. sub $t5,$t6,1
58. addi $t3,$zero,0
59. addi $t1,$t1,1
60. j judge
61.
62. judge:
63. beq $t5,$t4,end
64. j while
65.
66. special:
67. addi $t2,$t2,1
68. slt $s1,$t7,$t5
69. beq $s1,$s2,rstrong
70.
71. rweak:
72. add $t3,$zero,1
73. j end
74.
75. rstrong:
76. addi $t1,$t1,1
77. sub $t5,$t5,1
78. addi $t3,$zero,0
79. j end
80.
81. end:
82. li $v0,4
83. la $a0,mess
84. syscall
85. li $v0,1
86. add $a0,$t2,$zero
87. syscall
88.
```

```



















89. li $v0,4
90. la $a0,mess1
91. syscall
92. li $v0,1
93. add $a0,$t1,$zero
94. syscall
95.
96. li $v0,4
97. la $a0,mess2
98. syscall
99.
100. bne $t3,0,sure
101. sure:
102. li $v0,4
103. la $a0,yes
104. syscall
105. j final
106.
107. nosure:
108. li $v0,4
109. la $a0,no
110. syscall
111. j final
112.
113. final:
114. li $v0,4
115. la $a0,mess3
116. syscall
117. li $v0,1
118. add $a0,$t5,$zero
119. syscall
120.
121. addi $v0,$zero,10
122. syscall

```

七、实验验算程序下板测试过程与实现

由于本人的摔鸡蛋算法程序用到了 `li`, `la`, 在 31 条指令中没有对应的指令, 难以使用, 所以本人寻找到了另外一个班级的测试文件, 并进行了 31 条指令的全部测试, 下图是部分测试截图 (由于本人的 `testbench` 的截止代码与 `MARS` 存在差异, 所以结尾会多出一大截, 前面与 `MARS` 的输出完全相同, 可以判定本人 CPU 仿真不存在问题)

31 条指令测试所用 coe 文件:

 _1_addi.coe	2022/6/9 21:40	COE 文件
 _1_lui.coe	2022/6/9 21:40	COE 文件
 _2_add.coe	2022/6/9 21:41	COE 文件
 _2_addu.coe	2022/6/9 21:41	COE 文件
 _2_and.coe	2022/6/9 21:42	COE 文件
 _2_andi.coe	2022/6/9 21:42	COE 文件
 _2_lsw.coe	2022/6/9 23:21	COE 文件
 _2_lsw2.coe	2022/6/9 23:22	COE 文件
 _2_nor.coe	2022/6/9 23:23	COE 文件
 _2_or.coe	2022/6/9 23:23	COE 文件
 _2_ori.coe	2022/6/9 23:23	COE 文件
 _2_sll.coe	2022/6/9 23:24	COE 文件
 _2_sllv.coe	2022/6/9 23:24	COE 文件
 _2_slt.coe	2022/6/9 23:24	COE 文件
 _2_slti.coe	2022/6/9 23:25	COE 文件
 _2_sltiu.coe	2022/6/9 23:25	COE 文件
 _2_sltu.coe	2022/6/9 23:25	COE 文件
 _2_sra.coe	2022/6/9 23:26	COE 文件

_2_srav.coe
 _2_srl.coe
 _2_srlv.coe
 _2_sub.coe
 _2_subu.coe
 _2_xor.coe
 _2_xori.coe
 _2_xori.txt
 _3.5_beq.coe
 _3.5_beq.txt
 _3.5_bne.coe
 _3.5_bne.txt
 _3_j.coe
 _3_j.txt
 _3_jal.coe
 _3_jal.txt
 _4_jr.coe
 _4_jr.txt

部分测试结果：（所有的均测试过，但全贴过于冗余）

```

diff --git a/output.txt b/result.txt
index 00000000..00000000
--- output.txt
+++ result.txt
@@ -1,31 @@
Top line 2150
regfile5:~00005555
regfile6:~0000aaaa
regfile7:~00001234
regfile8:~00000000
regfile9:~ffff0000
regfile10:~ffff7fff
regfile11:~ffff0000
regfile12:~ffff00ff
regfile13:~ffffaaaa
regfile14:~ffff5555
regfile15:~ffffedcb
regfile16:~ffff00ff
regfile17:~ffff0000
regfile18:~ffff0000
regfile19:~00000000
regfile20:~ffff0000
regfile21:~00000000
regfile22:~ffff0000
regfile23:~00000000
regfile24:~00000000
regfile25:~0000ffff
regfile26:~0000ffff
regfile27:~ffff00ff
regfile28:~0000ffff
regfile29:~ffff00ff
regfile30:~0000ffff
regfile31:~ffff00ff
pc:~00400100
inst:~00000000
regfile0:~00000000
regfile1:~0000ffff
regfile2:~00000000
regfile3:~00007fff
regfile4:~0000f0f0
regfile5:~00005555
regfile6:~0000aaaa
regfile7:~00001234
regfile8:~00000000
regfile9:~ffff0000
regfile10:~ffff7fff
regfile11:~ffff0000
  
```


output.txt <-> result.txt - KDiff3

File Edit Directory Movement Diffview Merge Window Settings Help

A: d:\shuzidianlu\Xilinx\CPU\CPU.sia\sim_1\behav\output.txt ... B: D:\shuzidianlu\Xilinx\CPU\result.txt ...

Top line 1985 Encoding: System Line end style: DOS Top line 1985 Encoding: System Line end style: DOS

```
regfile10: .ffffffff
regfile11: .ffffffff
regfile12: .ffffffff
regfile13: .ffffffff
regfile14: .ffffffff
regfile15: .ffffffff
regfile16: .ffffffff
regfile17: .ffffffff
regfile18: .ffffffff
regfile19: .ffffffff
regfile20: .ffffffff
regfile21: .ffffffff
regfile22: .ffffffff
regfile23: .ffffffff
regfile24: .ffffffff
regfile25: .ffffffff
regfile26: .ffffffff
regfile27: .ffffffff
regfile28: .ffffffff
regfile29: .ffffffff
regfile30: .ffffffff
regfile31: .ffffffff
pc: .00400168
instr: .00000000
regfile0: .00000000
regfile1: .ffffffff
regfile2: .ffffffff
regfile3: .ffffffff
regfile4: .ffffffff
regfile5: .ffffffff
regfile6: .ffffffff
regfile7: .ffffffff
regfile8: .ffffffff
regfile9: .ffffffff
regfile10: .ffffffff
regfile11: .ffffffff
regfile12: .ffffffff
regfile13: .ffffffff
regfile14: .ffffffff
regfile15: .ffffffff
regfile16: .ffffffff
```

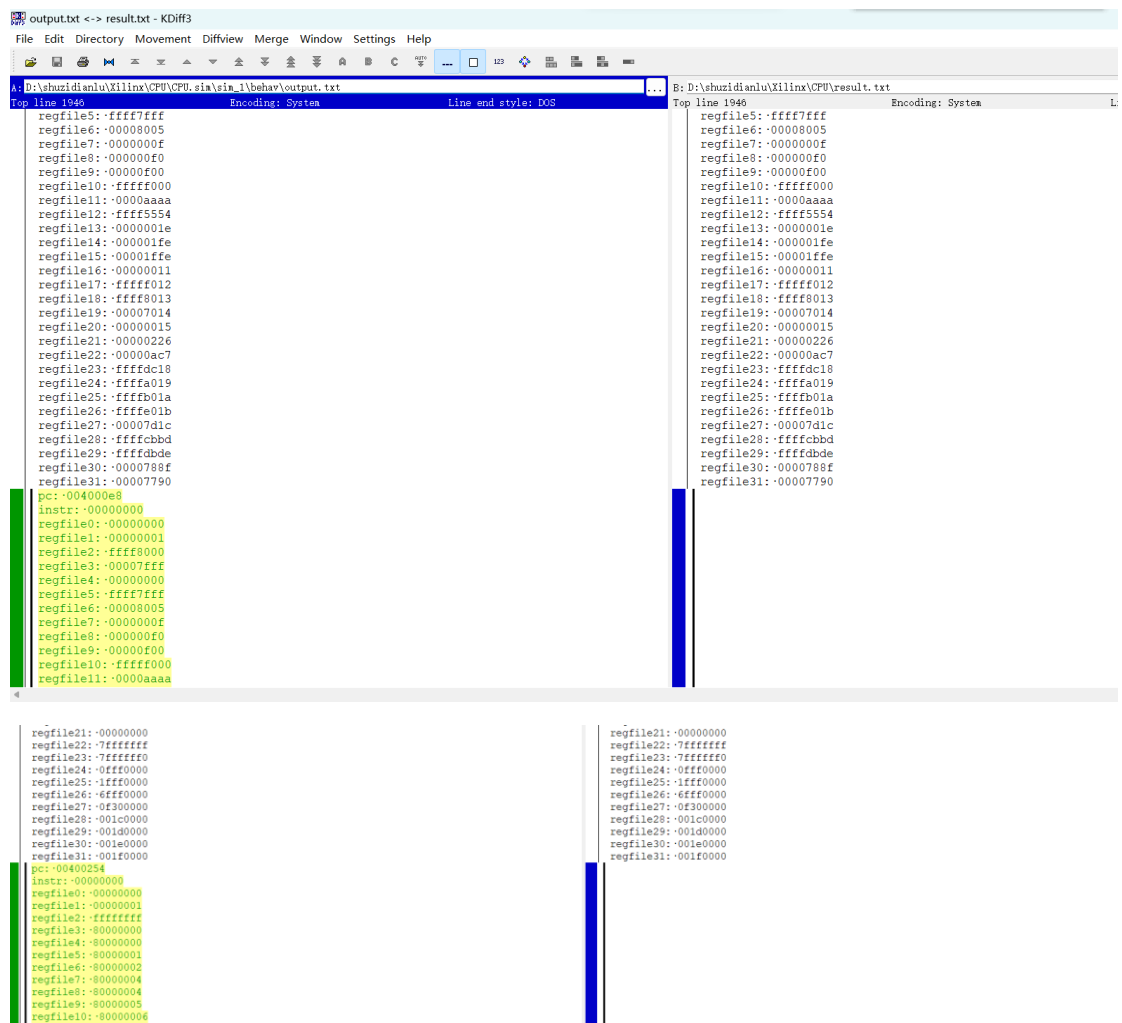
output.txt <-> result.txt - KDiff3

File Edit Directory Movement Diffview Merge Window Settings Help

A: d:\shuzidianlu\Xilinx\CPU\CPU.sia\sim_1\behav\output.txt ... B: D:\shuzidianlu\Xilinx\CPU\result.txt ...

Top line 3174 Encoding: System Line end style: DOS Top line 3174 Encoding: System Line end style: DOS

```
regfile9: .ffffffff
regfile10: .ffffffff
regfile11: .ffffffff
regfile12: .ffffffff
regfile13: .ffffffff
regfile14: .ffffffff
regfile15: .ffffffff
regfile16: .ffffffff
regfile17: .ffffffff
regfile18: .ffffffff
regfile19: .ffffffff
regfile20: .ffffffff
regfile21: .ffffffff
regfile22: .ffffffff
regfile23: .ffffffff
regfile24: .ffffffff
regfile25: .ffffffff
regfile26: .ffffffff
regfile27: .00000001
regfile28: .00000001
regfile29: .00000001
regfile30: .00000001
regfile31: .ffffffff
pc: .00400178
instr: .201bffff
regfile0: .00000000
regfile1: .ffffffff
regfile2: .ffffffff
regfile3: .ffffffff
regfile4: .ffffffff
regfile5: .ffffffff
regfile6: .ffffffff
regfile7: .ffffffff
regfile8: .ffffffff
regfile9: .ffffffff
regfile10: .ffffffff
regfile11: .ffffffff
regfile12: .ffffffff
regfile13: .ffffffff
regfile14: .ffffffff
regfile15: .ffffffff
```



八、 总结与体会

控制信号表的编写和数据通路的绘制，可以说是整个实验完成的关键，很多时候编写遇到了问题，就会把两者联系起来放在代码逻辑里跑，帮助我 debug 了很多错误除此之外

本次 CPU 实验遇到了一些问题，ALU 的标志位的编写，LW 信号要与 cpu 同级的 DM 建立联系，这些都给编写带来了困扰。

九、 附件（所有程序）

CPU 的设计程序

ALL.v （总体高层架构，哈佛结构）

```
1. module sccomp_dataflow(
2.     input clk,
3.     input rst,
4.     output [31:0] inst,
5.     output wire [31:0] pc, // to test
6.     output wire [10:0] dm_addr,
7.     output wire [10:0] im_addr
8. );
```

```

9.
10.    wire dw, dr, dena;
11.    wire [31:0] w_data, r_data;
12.    wire [31:0] instr;
13.    wire[10:0]temp_im_addr;
14.    wire [31:0] res;
15.
16.    assign inst = instr;
17.    assign dm_addr = (res - 32'h10010000)/4;
18.
19.    IMEM imemory(
20.        .addr(temp_im_addr/4),
21.        .instr(instr)
22.    );
23.    assign temp_im_addr = pc - 32'h0040_0000;
24.    assign im_addr =temp_im_addr;
25.
26.    DMEM dmemory(
27.        .clk(clk), .ena(1'b1), .DM_W(dw), .DM_R(dr), .DM_Addr(dm_addr[10:0])
28.        , .DM_WData(w_data),
29.        .DM_RData(r_data)
30.    );
31.    cpu cpu31(
32.        .clk(clk),.reset(rst), .inst(instr), .rdata(r_data),
33.        .DM_CS(dena), .DM_W(dw), .DM_R(dr), .wdata(w_data), .pc(pc), .addr(r
34.        es)
35.    );
36. endmodule

```

Add.v

```

1. `timescale 1ns / 1ns
2. module add(
3.    input [31:0] a,
4.    input [31:0] b,
5.    output [31:0] r,
6.    output overflow
7. );
8.    assign r=a+b;
9.    assign overflow=(a[31]==b[31]&&a[31]!=r[31])?1:0;
10. endmodule

```

add8.v

```
1. `timescale 1ns / 1ns
2. module add8(
3.     input [31:0] a,
4.     output [31:0] r
5. );
6.     assign r=a+4;
7. endmodule
```

ALU.v

```
1. `timescale 1ns / 1ns
2. module alu(
3.     input [31:0] a,          //OP1
4.     input [31:0] b,          //OP2
5.     input [4:0] aluc,        //controller
6.     output [31:0] r,         //result
7.     output zero,
8.     output carry,
9.     output negative,
10.    output overflow);
11.
12.    //instruction set
13.    parameter ADDU = 5'b 00000;
14.    parameter ADD = 5'b 00001;
15.    parameter SUB = 5'b 00010;
16.    parameter SUBU = 5'b 00011;
17.    parameter AND = 5'b 00100;
18.    parameter OR = 5'b 00101;
19.    parameter XOR = 5'b 00110;
20.    parameter NOR = 5'b 00111;
21.    parameter SLL = 5'b 01000;
22.    parameter SRL = 5'b 01001;
23.    parameter SRA = 5'b 01010;
24.    parameter SLLV = 5'b 01011;
25.
26.    parameter LUI = 5'b 01101;
27.    parameter SRLV = 5'b 01110;
28.    parameter SRAV = 5'b 01111;
29.
30.    parameter SLT = 5'b 10010;
31.    parameter SLTU = 5'b 10011;
```

```

32.         parameter LW = 5'b 10001;//SW
33.
34.
35.         //easy to calculate
36.         wire signed [31:0]aSign,bSign ;
37.         assign aSign =a;
38.         assign bSign =b;
39.
40.         //store res
41.         //use [32:0] instead of [31:0] use [32] to change CF SF OF ZF
42.         reg [32:0]res;
43.
44.         always @(*) begin
45.             case(aluc)
46.                 ADD:    res<=aSign+bSign;
47.                 LW:     begin res<=aSign+bSign;
48.
49.                 end
50.                 ADDU:   res<=a+b;
51.
52.                 SUB:    res<=aSign-bSign;
53.                 SLT:    res<=aSign<bSign?32'b1:32'b0;
54.
55.                 SUBU:   res<=a-b;
56.                 SLTU:   res<=a<b?1:0;
57.
58.                 AND:    res<=a&b;
59.                 OR:     res<=a|b;
60.                 XOR:    res<=a^b;
61.                 NOR:    res<=~(a|b);
62.
63.                 SLL:    res<=b<<a;
64.                 SRL:    res<=b>>a;
65.                 SRA:    res<=bSign>>>aSign;
66.                 SLLV:   res<=b<<a[4:0];
67.                 SRLV:   res<=b>>a[4:0];
68.                 SRAV:   res<=bSign>>>aSign[4:0];
69.
70.                 LUI:    res<={b[15:0],16'b0};
71.
72.                 default:
73.                     res<=32'b0;
74.                 endcase
75.             end

```

```

76.
77.    //process output
78.    assign r=res[31:0];
79.
80.    //process ZF,OF,CF,SF
81.    assign zero =(res==32'b0)?1'b1:1'b0;//ZF
82.    //only add sub is not process 明天写
83.    assign overflow=(aluc==ADD)?(res[32]?((aSign>0&&bSign>0)?1'b1:1'b0):((aSign<0&&bSign<0)?1'b1:1'b0)):
        (aluc==SUB?(res[32]?((aSign>0&&bSign<0)?1'b1:1'b0):((aSign<0&&bSign>0)?1'b1:1'b0)):1'b0);//OF
84.    assign carry = (aluc==ADDU|aluc==SUBU|aluc==SLTU|aluc==SRA|aluc==SRL|aluc==SLL)?res[32]:1'bz;//CF
85.    assign negative=(aluc==SUB?(aSign<bSign):((aluc==SUBU)?(a<b):1'b0));//SF
86. endmodule

```

control.v

```

1. `timescale 1ns / 1ns
2. module control(
3.     input clk,
4.     input zero,
5.     input [31:0] i,
6.
7.     output M1,
8.     output M2,
9.     output M3,
10.    output M4,
11.    output M5,
12.    output M6,
13.    output M7,
14.    output M8,
15.    output M9,
16.    output M10,
17.
18.    output PC_CLK,
19.    output IM_R,
20.    output [4:0] ALUC,
21.
22.    output RF_CLK,
23.    output RF_W,
24.    output DM_W,
25.    output DM_R,
26.    output DM_CS,

```

```

27.     output C_EXT16
28. );
29.     assign PC_CLK = clk;
30.     assign IM_R = 1;
31.     assign M1 = i[29]|i[30];
32.     assign M2 = ~(i[16]|( i[24] & zero) | (i[25] & ~zero));
33.     assign M3 = i[16];
34.     assign M4 = i[13] | i[14] | i[15];
35.     assign M5 = ~(i[10]|i[11]|i[12]|i[13]|i[14]|i[15]|i[16]);
36.     assign M6 = i[17]|i[18]|i[19]|i[20]|i[21]|i[22]|i[23]|i[26]|i[27]|i[28];

37.     assign M7 = i[30];
38.     assign M8=i[8]|i[9]|i[16]|i[26]|i[27]|i[22];
39.     assign M9 =i[22];
40.     assign M10 = i[17] | i[18] | i[19] | i[20] | i[21] | i[22] | i[23] | i[2
        6] | i[27] | i[28];
41.
42.     assign ALUC[0] = i[0]|i[3]|i[5]|i[7]|i[9]|i[11]|i[15]|i[17]|i[20]|i[22]|
        i[23]|i[27]|i[28];
43.     assign ALUC[1]=i[2]|i[3]|i[6]|i[7]|i[8]|i[9]|i[12]|i[14]|i[15]|i[21]|i[2
        4]|i[25]|i[26]|i[27];
44.     assign ALUC[2]=i[4]|i[5]|i[6]|i[7]|i[14]|i[15]|i[19]|i[20]|i[21]|i[28];

45.     assign ALUC[3]=i[10]|i[11]|i[12]|i[13]|i[14]|i[15]|i[28];
46.     assign ALUC[4]=i[8]|i[9]|i[22]|i[23]|i[26]|i[27];
47.
48.     assign RF_W = ~(i[16] | i[23] | i[24] | i[25] | i[29]);
49.     assign RF_CLK = ~clk;
50.
51.     assign DM_W = i[23];
52.     assign DM_R = i[22];
53.     assign DM_CS = i[22] | i[23];
54.     assign C_EXT16 = ~(i[19] | i[20] | i[21]);
55. endmodule

```

cpu.v

```

1. `timescale 1ns / 1ps
2. module cpu(
3.     input      clk,
4.     input      reset,
5.     input  [31:0] inst,
6.     input  [31:0] rdata,
7.     output [31:0] pc,

```

```

8.    output [31:0] addr,
9.    output [31:0] wdata,
10.   output      IM_R,
11.   output      DM_CS,
12.   output      DM_R,
13.   output      DM_W
14.   );
15.   //控制信号(除有关存储器)
16.   wire PC_CLK;           //
17.   wire PC_ENA;           //
18.   wire M1;               //
19.   wire M2;               //
20.   wire M3;               //
21.   wire M4;               //
22.   wire M5;               //
23.   wire M6;               //
24.   wire M7;               //
25.   wire M8;               //
26.   wire M9;               //
27.   wire M10;              //
28.   wire [4:0] ALUC;        //
29.   wire RF_W;              //
30.   wire RF_CLK;            //
31.   wire C_EXT16;           //
32.   //运算标志位
33.   wire zero;              //
34.   wire carry;             //
35.   wire negative;          //
36.   wire overflow;          //
37.   wire add_overflow;      //
38.   //
39.   wire [31:0] INS;         //译码后指令
40.   //数据通路(除有关存储器)
41.   wire [31:0] D_ALU;        //
42.   wire [31:0] D_PC;        //
43.   wire [31:0] D_RF;        //
44.   wire [31:0] D_Rs;        //
45.   wire [31:0] D_Rt;        //
46.   wire [31:0] D_IM;        //
47.   wire [31:0] D_DM;        //
48.   wire [31:0] D_Mux1;      //
49.   wire [31:0] D_Mux2;      //
50.   wire [31:0] D_Mux3;      //
51.   wire [4:0] D_Mux4;       //

```



```

52.    wire [31:0] D_Mux5;           //
53.    wire [31:0] D_Mux6;           //
54.    wire [31:0] D_Mux7;           //
55.    wire [31:0] D_Mux8;           //
56.    wire [31:0] D_Mux9;           //
57.    wire [5:0] D_Mux10;           //
58.
59.    wire [31:0] D_EXT1;            //
60.    wire [31:0] D_EXT5;            //
61.    wire [31:0] D_EXT16;           //
62.    wire [31:0] D_EXT18;           //
63.    wire [31:0] D_ADD;             //
64.    wire [31:0] D_ADD8;            //
65.    wire [31:0] D_NPC;             //
66.    wire [31:0] D_ii;              //
67.    assign PC_ENA = 1;
68.    //外部通路连接
69.    assign pc = D_PC;
70.    assign addr = D_ALU;
71.    assign wdata = D_Rt;
72.    assign D_DM=rdata;
73.    //指令译码
74.    instr_dec cpu_ins (inst, INS);
75.    control cpu_control (.clk(clk),
76.        .zero(zero),
77.        .i(INS),
78.        .M1(M1),
79.        .M2(M2),
80.        .M3(M3),
81.        .M4(M4),
82.        .M5(M5),
83.        .M6(M6),
84.        .M7(M7),
85.        .M8(M8),
86.        .M9(M9),
87.        .M10(M10),
88.
89.        .PC_CLK(PC_CLK),
90.        .IM_R(IM_R),
91.        .ALUC(ALUC),
92.
93.        .RF_CLK(RF_CLK),
94.        .RF_W(RF_W),
95.        .DM_W(DM_W),

```

```

96.         .DM_R(DM_R),
97.         .DM_CS(DM_CS),
98.         .C_EXT16(C_EXT16));
99.     // 部件
100.    pcreg  pc_out      (PC_CLK,      reset,      PC_ENA,      D_Mux2,      D_P
        C);
101.    alu     cpu_alu     (D_Mux5,      D_Mux6,      ALUC[4:0],   D_ALU,      zero
        ,          carry,          negative, overflow);
102.    regfile cpu_ref     (RF_CLK,      reset,      RF_W,          overflow, ins
        t[25:21], inst[20:16], D_Mux10,   D_Mux8,D_Rs, D_Rt);
103.    mux     cpu_mux1    (D_NPC,      D_ii,      M1,          D_Mux1);
104.    mux     cpu_mux2    (D_Mux3,      D_Mux1,      M2,          D_Mux2);
105.    mux     cpu_mux3    (D_ADD,      D_Rs,      M3,          D_Mux3);
106.    mux5    cpu_mux4    (inst[10:6], D_Rs[4:0],   {INS[30],M4},D_Mux4);
107.    mux     cpu_mux5    (D_EXT5,      D_Rs,M5,D_Mux5);
108.    mux     cpu_mux6    (D_Rt,      D_EXT16,      M6,          D_Mux6);
109.    mux     cpu_mux7    (D_ALU,      D_ADD8,      M7,          D_Mux7);
110.    mux     cpu_mux8    (D_Mux7,D_Mux9,M8,D_Mux8);
111.    mux     cpu_mux9    (D_ALU,D_DM,      M9,          D_Mux9);
112.    mux5    cpu_mux10   (inst[15:11],inst[20:16],{INS[30],M10},D_Mux10);
113.
114.    extend5 cpu_ext5     (D_Mux4,      D_EXT5);
115.    extend16 cpu_ext16   (inst[15:0], C_EXT16,      D_EXT16);
116.    extend18 cpu_ext18   (inst[15:0], D_EXT18);
117.    add     cpu_add     (D_EXT18,      D_NPC,      D_ADD,      add_overflow)
        ;
118.    add8    cpu_add8    (D_PC,      D_ADD8);
119.    npc     cpu_npc     (D_PC,      reset,      D_NPC);
120.    II      cpu_ii      (D_PC[31:28],inst[25:0], D_ii);
121.
122. endmodule

```

dmem.v

```

1. `timescale 1ns/1ps
2. module DMEM(
3.     input clk,
4.     input ena,
5.     input DM_W,
6.     input DM_R,
7.     input [10:0] DM_Addr,
8.     input [31:0]DM_WData,
9.     output [31:0]DM_RData
10. );

```

```

11. reg[31:0]D_mem[0:31];
12.
13. assign DM_RData=(DM_R&&ena)?D_mem[DM_Addr]:32'bz;
14. always@(posedge clk)begin
15.     if(DM_W && ena)begin
16.         D_mem[DM_Addr]<=DM_WData;
17.     end
18. end
19.
20.
21.
22. endmodule

```

extend5.v

```

1. `timescale 1ns / 1ns
2. module extend5 #(parameter WIDTH = 5)(
3.     input [WIDTH - 1:0] a,
4.     output [31:0] b
5. );
6.     assign b = {(32 - WIDTH){1'b0}},a};
7. endmodule

```

extend16.v

```

1. `timescale 1ns / 1ns
2. module extend16 #(parameter WIDTH = 16)(
3.     input [WIDTH - 1:0] a,
4.     input sext,          //1 表示有符号
5.     output [31:0] b
6. );
7.     assign b = sext ? {(32 - WIDTH){a[WIDTH - 1]}},a : {(32 - WIDTH){1'b0
    }},a};
8. endmodule

```

extend18.v

```

1. `timescale 1ns / 1ns
2. module extend18 (
3.     input [15:0] a,
4.     output [31:0] b
5. );

```

```

6.     assign b = {(32 - 18){a[15]}},a,2'b00;
7. endmodule

```

II.v

```

1. `timescale 1ns / 1ns
2. module II(
3.     input [3:0] a,
4.     input [25:0] b,
5.     output [31:0] r
6. );
7.     assign r = {a, b<<2};
8. endmodule

```

IMEM_test.v

```

1. module test_imem(
2. );
3.     reg [10:0] a;
4.     wire [31:0] spo;
5.     IMEM imem(
6.         .addr(a),
7.         .instr(spo)
8.     );
9.
10.    initial begin
11.        a = 11'b0;
12.    end
13.
14.    always begin
15.        #20 a = a + 1'b1;
16.    end
17. endmodule

```

IMEM.v

```

1. `timescale 1ns / 1ps
2. //////////////////////////////////////
   //
3. // Company:
4. // Engineer:
5. //

```

```

6. // Create Date: 2022/06/07 10:46:00
7. // Design Name:
8. // Module Name: IMEM
9. // Project Name:
10. // Target Devices:
11. // Tool Versions:
12. // Description:
13. //
14. // Dependencies:
15. //
16. // Revision:
17. // Revision 0.01 - File Created
18. // Additional Comments:
19. //
20. //////////////////////////////////////
   //
21.
22.
23. module IMEM(
24.     input[10:0]addr,
25.     output[31:0]instr
26. );
27.     dist_mem_gen_1 insrt_mem(
28.         .a(addr),
29.         .spo(instr)
30.     );
31. endmodule

```

input_decode.v

```

1. `timescale 1ns / 1ns
2. module instr_dec(
3.     input [31:0] instr_code,
4.     output reg [31:0] i
5. );
6.     wire [11:0] t;
7.     assign t = {instr_code[31:26],instr_code[5:0]};
8.     always @ (*)
9.     begin
10.         casez(t)
11.             12'b000000100000 :i = 32'b00000000000000000000000000000001;
12.             12'b000000100001 :i = 32'b00000000000000000000000000000010;
13.             12'b000000100010 :i = 32'b00000000000000000000000000000100;
14.             12'b000000100011 :i = 32'b00000000000000000000000000000100;

```



```

10.     case(choose)
11.         1'b1:z <= b;
12.         1'b0:z <= a;
13.     endcase
14. end
15. endmodule

```

mux5.v

```

1. `timescale 1ns / 1ns
2. module mux5(
3.     input [4:0] a,
4.     input [4:0] b,
5.     input [1:0] choose,
6.     output [4:0] z
7. );
8.     reg [4:0] t_z;
9.     always @(*)
10.     begin
11.         case(choose)
12.             2'b01:t_z <= b;
13.             2'b00:t_z <= a;
14.             2'b10:t_z <= 5'b11111;
15.             2'b11:t_z <= 5'b11111;
16.             default:t_z <= 5'bz;
17.         endcase
18.     end
19.     assign z = t_z;
20. endmodule

```

npc.v

```

1. `timescale 1ns / 1ns
2. module npc(
3.     input [31:0] a,
4.     input rst,
5.     output [31:0] r
6. );
7.     assign r = rst ? a : a+4;
8. endmodule

```

pcreg.v

```

1. `timescale 1ns / 1ns
2. module pcreg(
3.     input clk, //1 位输入, 寄存器时钟信号, 下降沿时为 PC 寄存器赋值
4.     input rst, //1 位输入, 异步重置信号, 高电平时将 PC 寄存器清零
5.         //注: 当 ena 信号无效时, rst 也可以重置寄存器
6.     input ena, //1 位输入, 有效信号高电平时 PC 寄存器读入 data_in 的值, 否则保持
        原有输出
7.     input [31:0] data_in, //32 位输入, 输入数据将被存入寄存器内部
8.     output [31:0] data_out //32 位输出, 工作时始终输出 PC 寄存器内部存储的值
9. );
10.     reg [31:0] pc_regis;
11.     always @( posedge rst or negedge clk ) begin
12.         if (ena) begin
13.             if (rst) begin
14.                 pc_regis <= 32'h00400000;
15.             end
16.             else begin
17.                 pc_regis <= data_in;
18.             end
19.         end
20.     end
21.     assign data_out = (ena && !rst) ? pc_regis : 32'h00400000;
22. endmodule

```

regfile.v

```

1. `timescale 1ns / 1ns
2. module regfile(
3.     input clk, //寄存器组时钟信号, 下降沿写入数据
4.     input rst, //reset 信号, 异步复位, 高电平时全部寄存器置零
5.     input we, //寄存器读写有效信号, 高电平时允许寄存器写入数据, 低电平时允许寄存器
        读出数据
6.     input ov, //overflow
7.     input [4:0] raddr1, //所需读取的寄存器的地址
8.     input [4:0] raddr2, //所需读取的寄存器的地址
9.     input [4:0] waddr, //写寄存器的地址
10.    input [31:0] wdata, //写寄存器数据, 数据在 clk 下降沿时被写入
11.    output [31:0] rdata1, //raddr1 所对应寄存器的输出数据
12.    output [31:0] rdata2 //raddr2 所对应寄存器的输出数据
13. );
14.
15.    reg [31:0] array_reg [31:0];
16.    reg c_o;
17.    always@(ov)

```



```

18.     begin
19.     case(ov)
20.     1'bz:c_o = 1;
21.     1'b1:c_o = 0;
22.     1'b0:c_o = 1;
23.     default:c_o=1;
24.     endcase
25.     end
26.
27.
28.
29.     assign rdata1 = rst?0:array_reg[raddr1];
30.     assign rdata2 = rst?0:array_reg[raddr2];
31.
32.     //清零
33.     integer i;
34.     always@(posedge rst)begin
35.         for(i=0;i<32;i=i+1)begin
36.             array_reg[i]<=0;
37.         end
38.     end
39.     //写入
40.     always@(negedge clk)begin
41.         if(we&&~c_o)begin
42.             array_reg[waddr]<=wdata;
43.             array_reg[0]<=0;
44.         end
45.     end
46.
47. endmodule

```

test.v

```

1. `timescale 1ns / 1ps
2. module test_cpu(
3. );
4.     reg clk, rst;
5.     wire [31:0] inst, pc;
6.     reg [31:0] cnt;
7.     wire[10:0] dma, ima;
8.     integer file_open;
9.     initial begin
10.         clk = 1'b0;
11.         rst = 1'b1;

```

```

12.         #2 rst=1'b0;
13.         cnt = 0;
14.     end
15.
16.     always begin
17.         #1 clk = !clk;
18.     end
19. initial begin
20.     file_open = $fopen("D:/shuzidianlu/Xilinx/CPU/CPU.sim/sim_1/behav/output.txt", "w+"); //目标存放点
21.     if (!file_open) begin
22.
23.         $display("can not open \"888.txt\");
24.
25.     $finish;
26.
27.     end
28. end
29.     always @(negedge clk) begin
30.         cnt <= cnt + 1'b1;
31.         if (cnt < 8'h99 - 1&&cnt>=1'b1 ) begin // 可自我调整, 这是所有 txt 测试文件都能够容纳的数据, 在此时钟下, 为 6000 ns
32.
33.             $fdisplay(file_open, "pc: %h", sc.pc);
34.             $fdisplay(file_open, "instr: %h", sc.inst);
35.             // $fdisplay(file_open, "aluc: %h,a: %h,b:%h,res:%h", sc.cpu31.cpu_alu.aluc,sc.cpu31.cpu_alu.a,sc.cpu31.cpu_alu.b,sc.cpu31.cpu_alu.res);
36.             // $fdisplay(file_open, "wdata: %h,MUX9:%h,D_ALU:%h", sc.cpu31.D_Mux8,sc.cpu31.D_Mux9,sc.cpu31.D_ALU);
37.
38.             $fdisplay(file_open, "regfile0: %h", sc.cpu31.cpu_ref.array_reg[0]);
39.
40.             $fdisplay(file_open, "regfile1: %h", sc.cpu31.cpu_ref.array_reg[1]);
41.             $fdisplay(file_open, "regfile2: %h", sc.cpu31.cpu_ref.array_reg[2]);
42.             $fdisplay(file_open, "regfile3: %h", sc.cpu31.cpu_ref.array_reg[3]);
43.             $fdisplay(file_open, "regfile4: %h", sc.cpu31.cpu_ref.array_reg[4]);
44.             $fdisplay(file_open, "regfile5: %h", sc.cpu31.cpu_ref.array_reg[5]);
45.             $fdisplay(file_open, "regfile6: %h", sc.cpu31.cpu_ref.array_reg[6]);

```

```
45.          $fdisplay(file_open, "regfile7: %h", sc.cpu31.cpu_ref.array_reg[7]);
46.          $fdisplay(file_open, "regfile8: %h", sc.cpu31.cpu_ref.array_reg[8]);
47.          $fdisplay(file_open, "regfile9: %h", sc.cpu31.cpu_ref.array_reg[9]);
48.          $fdisplay(file_open, "regfile10: %h", sc.cpu31.cpu_ref.array_reg[10]);
49.          $fdisplay(file_open, "regfile11: %h", sc.cpu31.cpu_ref.array_reg[11]);
50.          $fdisplay(file_open, "regfile12: %h", sc.cpu31.cpu_ref.array_reg[12]);
51.          $fdisplay(file_open, "regfile13: %h", sc.cpu31.cpu_ref.array_reg[13]);
52.          $fdisplay(file_open, "regfile14: %h", sc.cpu31.cpu_ref.array_reg[14]);
53.          $fdisplay(file_open, "regfile15: %h", sc.cpu31.cpu_ref.array_reg[15]);
54.          $fdisplay(file_open, "regfile16: %h", sc.cpu31.cpu_ref.array_reg[16]);
55.          $fdisplay(file_open, "regfile17: %h", sc.cpu31.cpu_ref.array_reg[17]);
56.          $fdisplay(file_open, "regfile18: %h", sc.cpu31.cpu_ref.array_reg[18]);
57.          $fdisplay(file_open, "regfile19: %h", sc.cpu31.cpu_ref.array_reg[19]);
58.          $fdisplay(file_open, "regfile20: %h", sc.cpu31.cpu_ref.array_reg[20]);
59.          $fdisplay(file_open, "regfile21: %h", sc.cpu31.cpu_ref.array_reg[21]);
60.          $fdisplay(file_open, "regfile22: %h", sc.cpu31.cpu_ref.array_reg[22]);
61.          $fdisplay(file_open, "regfile23: %h", sc.cpu31.cpu_ref.array_reg[23]);
62.          $fdisplay(file_open, "regfile24: %h", sc.cpu31.cpu_ref.array_reg[24]);
63.          $fdisplay(file_open, "regfile25: %h", sc.cpu31.cpu_ref.array_reg[25]);
64.          $fdisplay(file_open, "regfile26: %h", sc.cpu31.cpu_ref.array_reg[26]);
65.          $fdisplay(file_open, "regfile27: %h", sc.cpu31.cpu_ref.array_reg[27]);
66.          $fdisplay(file_open, "regfile28: %h", sc.cpu31.cpu_ref.array_reg[28]);
```

```
67.             $fdisplay(file_open, "regfile29: %h", sc.cpu31.cpu_ref.array_reg[29]);
68.             $fdisplay(file_open, "regfile30: %h", sc.cpu31.cpu_ref.array_reg[30]);
69.             $fdisplay(file_open, "regfile31: %h", sc.cpu31.cpu_ref.array_reg[31]);
70.
71.         end
72.         if(cnt==400) begin
73.
74.             $fclose(file_open);
75.
76.         end
77.     end
78.     sccomp_dataflow sc(
79.         .clk(clk),
80.         .rst(rst),
81.         .inst(inst),
82.         .pc(pc),
83.         .dm_addr(dma), .im_addr(ima)
84.     );
85. endmodule
```