同济大学计算机系

计算机组成原理课程综合实验报告

学　　号 2351753

姓　　名 黄保翔

专　　业 计算机科学与技术

授课老师 张冬冬

一、实验内容

在本次实验中，使用 Verilog HDL 语言实现 54 条 MIPS 指令的 CPU 的设计，前仿真，后仿真和下板调试运行。

二、CPU 数据通路设计

（1）根据每条指令所涉及部件和部件的数据输入来源，画出每条指令的数据通路。

（2）根据每条指令功能，在已形成的数据通路下，画出每条指令从取指到执行过程的指令流程图

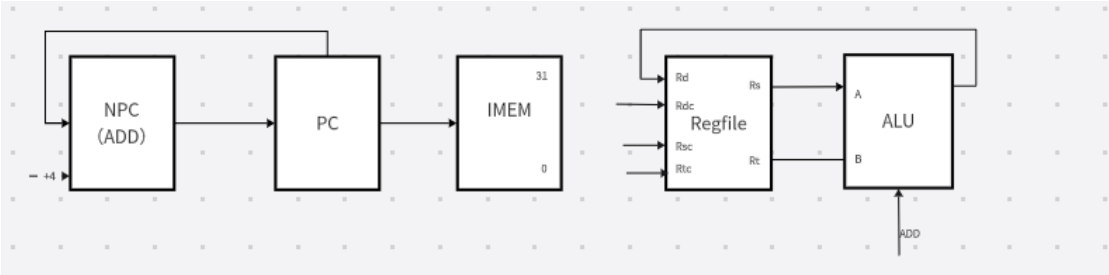
R type：

1.add

rd ← rs + r

功能：将两个寄存器的值相加，结果存入目标寄存器（R 型）。

格式：ADD rd, rs, rt （rd = rs + rt）

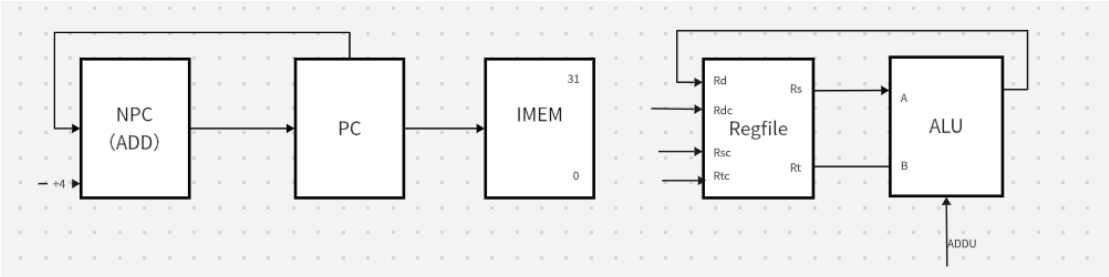


2.addu

rd ← rs + r

功能：类似 ADD，但不检查溢出（R 型）。

格式：ADDU rd, rs, rt

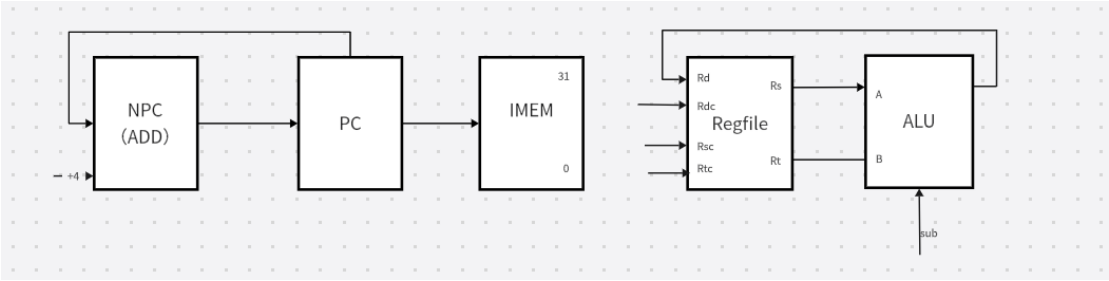


3.sub

rd ← rs - rt

功能：从一个寄存器值减去另一个寄存器值，结果存入目标寄存器（R 型）。

格式：SUB rd, rs, rt （rd = rs - rt）

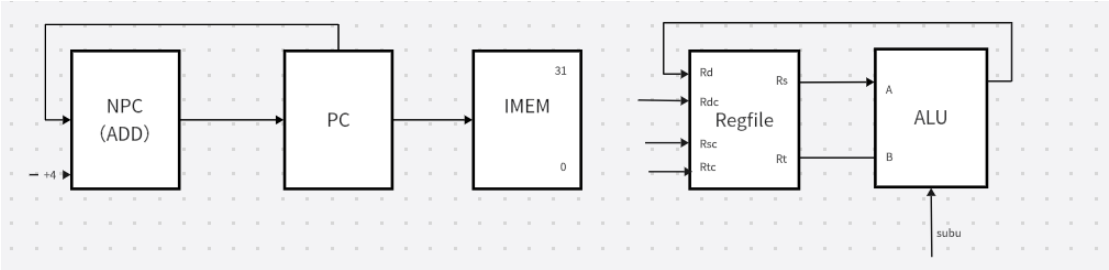


4.subu

rd ← rs - rt

功能：类似 SUB，但不检查溢出（R 型）。

格式：SUBU rd, rs, rt

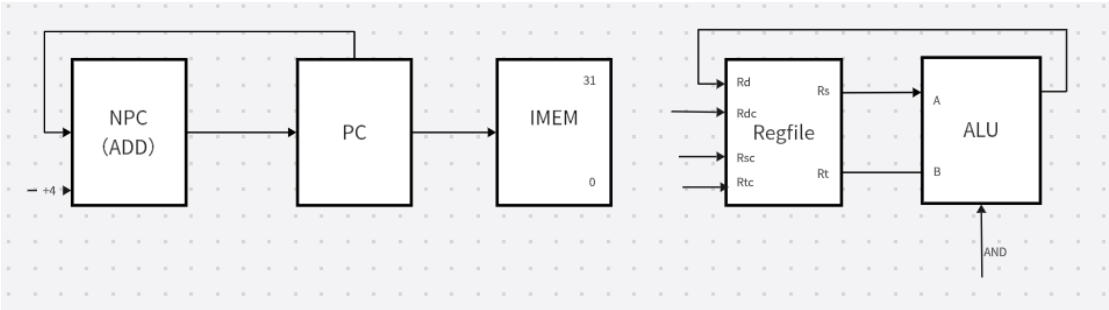


5. AND

rd ← rs AND rt

功能：对两个寄存器值按位与，结果存入目标寄存器（R 型）。

格式：AND rd, rs, rt （rd = rs & rt）

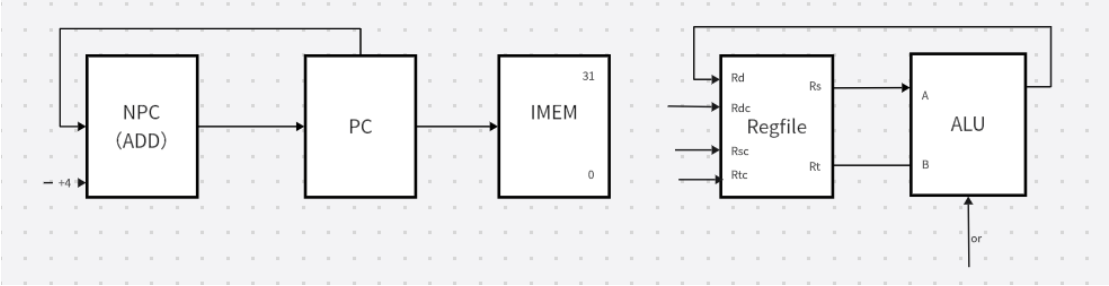


6. or

rd ← rs or rt

功能：对两个寄存器值按位或，结果存入目标寄存器（R 型）。

格式：OR rd, rs, rt （rd = rs | rt）

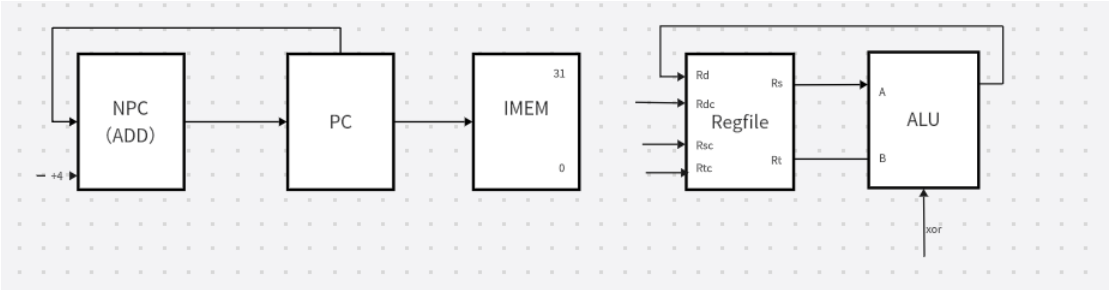


7.xor

rd ←rs XOR rt

功能：对两个寄存器值按位异或，结果存入目标寄存器（R 型）。

格式：XOR rd, rs, rt （rd = rs ^ rt）

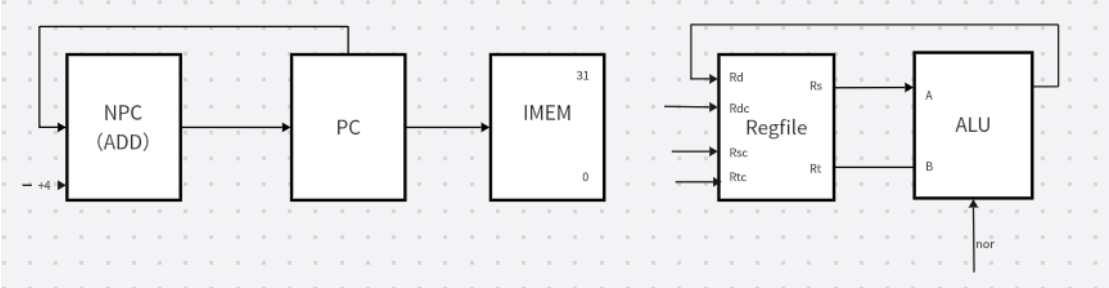


8.nor

rd ← rs NOR r

功能：对两个寄存器值按位或后取反，结果存入目标寄存器（R 型）。

格式：NOR rd, rs, rt （rd = ~(rs | rt)）



指令通路 add/addu/sub/subu/and/or/xor/nor

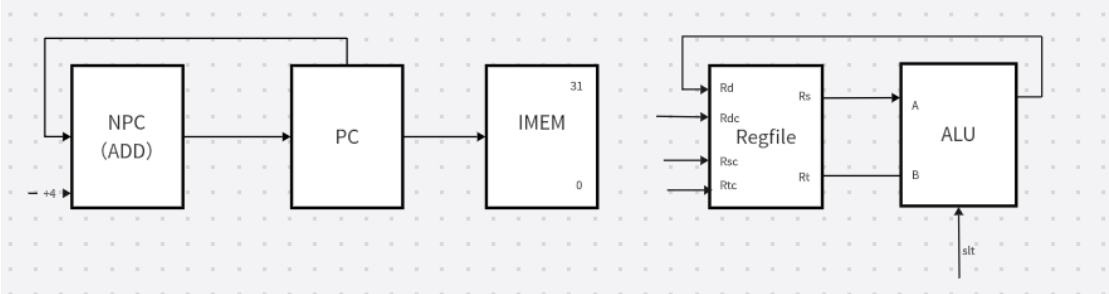
PC
↓
IMEM (读取指令)
↓
NPC (PC + 4 -> NPC)
↓
PC (NPC -> PC)
↓
寄存器文件
Rs -> A
Rt -> B
↓
ALU (执行运算)
↓
RES (结果 -> Rd)

9.slt

rd ←(rs < rt)

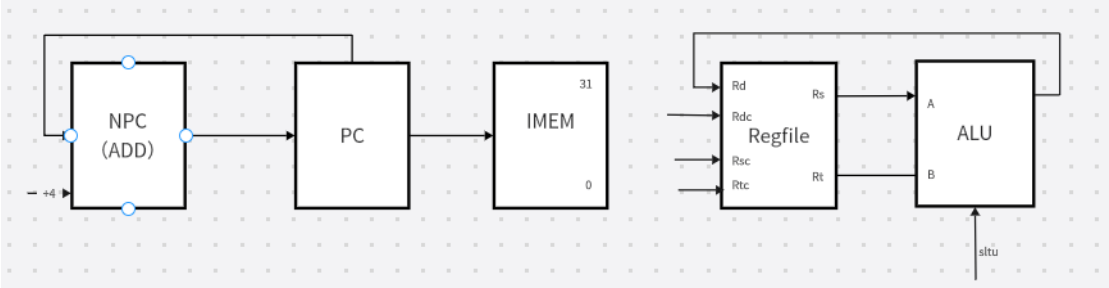
功能：比较两个寄存器值，若 rs < rt，则目标寄存器置 1，否则置 0（R 型）。

格式：SLT rd, rs, rt （rd = (rs < rt) ? 1 : 0）



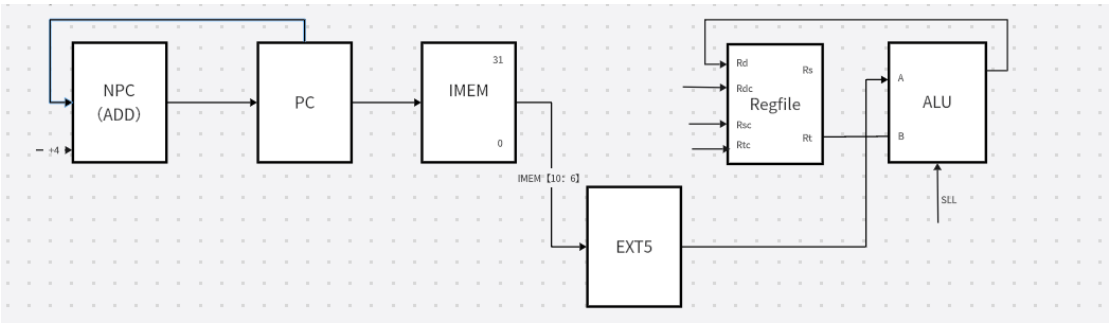
指令
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
Rs -> A
Rt -> B
A - B -> RES #这里其实做的是减法操作
SF ->Rd

10.sltu
rd ←(rs < rt)
功能：类似 SLT，但进行无符号比较（R 型）。
格式：SLTU rd, rs, rt



PC -> IMEM
PC + 4 -> NPC
NPC -> PC
Rs -> A
Rt -> B
A - B -> RES #这里其实做的是减法操作
SF ->Rd

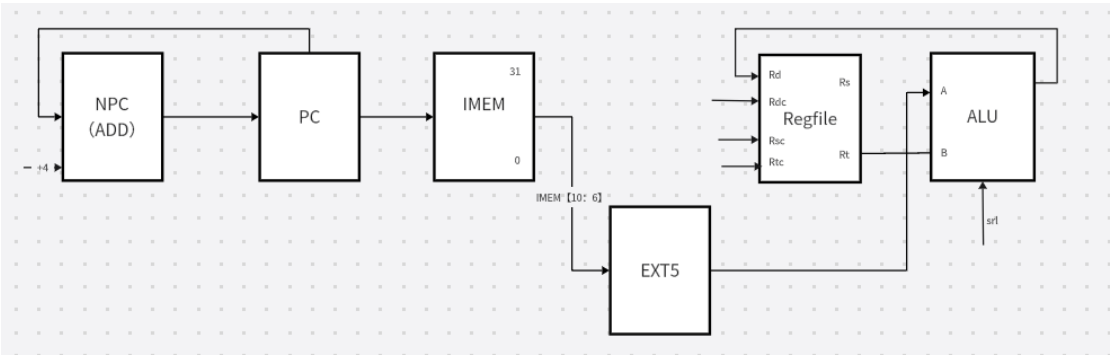
11.sll
rd ← rt << s
功能：将寄存器值逻辑左移指定位数，结果存入目标寄存器（R 型）。
格式：SLL rd, rt, shamt （rd = rt << shamt）



指令流程图

PC -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[10:6] -> EXT5
EXT5_out -> A
Rt -> B
B << A -> RES
Res -> Rd

12.srl
rd ←rt >> sa (logical)
功能：将寄存器值逻辑右移指定位数，结果存入目标寄存器（R 型）。
格式：SRL rd, rt, shamt （rd = rt >> shamt）

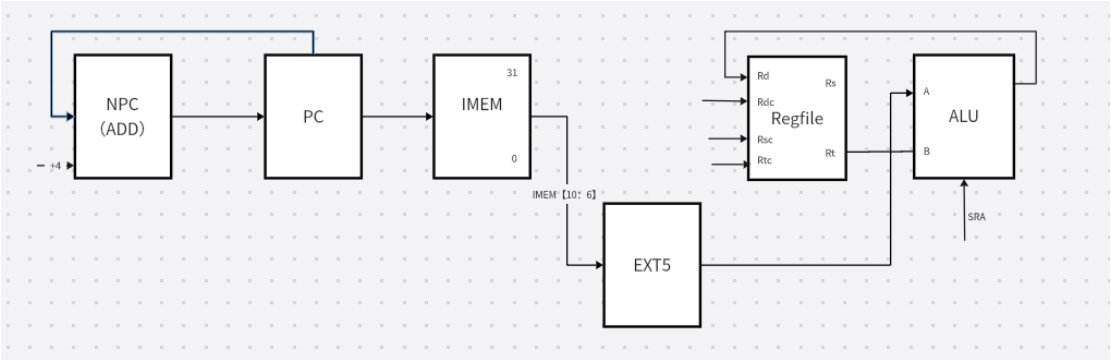


指令流程图

C -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[10:6] -> EXT5
EXT5_out -> A
Rt -> B
B >> A -> RES
Res -> Rd

13.sra
rd ←rt >> sa
功能：将寄存器值算术右移指定位数，结果存入目标寄存器（R 型）。

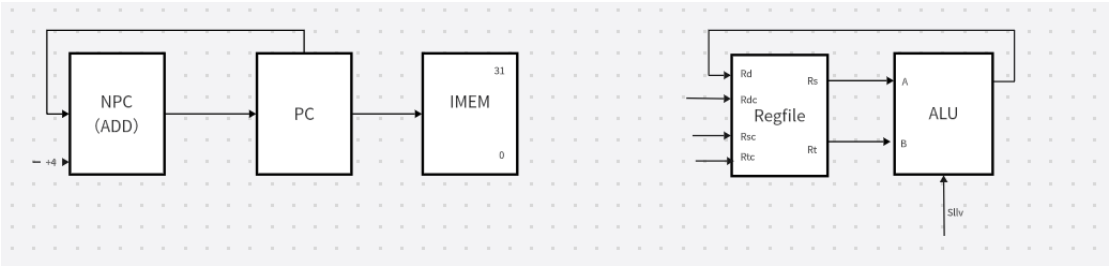
格式：SRA rd, rt, shamt （rd = rt >> shamt）



指令流程图

PC -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[10:6] -> EXT5
EXT5_out -> A
Rt -> B
B >> A -> RES
Res -> Rd

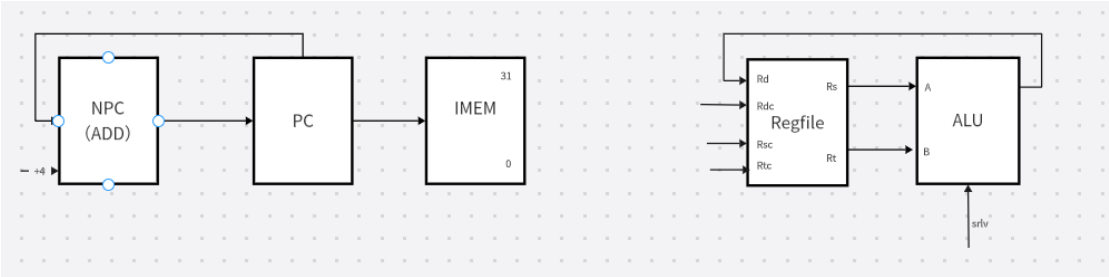
14.sllv
rd ← rt << rs
功能：根据另一寄存器值指定位数逻辑左移，结果存入目标寄存器（R 型）。
格式：SLLV rd, rt, rs （rd = rt << rs）



指令流程图
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
Rs[4:0] -> EXT5
EXT5_out -> A
Rt -> B
B << A -> RES
Res -> Rd

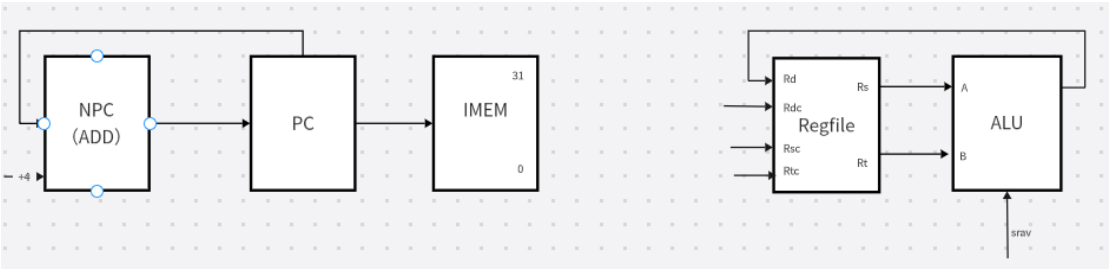
15.srlv

rd ←rt >> rs (logical)
功能：根据另一寄存器值指定位数逻辑右移，结果存入目标寄存器（R 型）。
格式：SRLV rd, rt, rs （rd = rt >> rs）



指令流程图
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
Rs[4:0] -> EXT5
EXT5_out -> A
Rt -> B
B >> A -> RES
Res -> Rd

16.srav
rd ←rt >> rs (arithmet)
功能：根据另一寄存器值指定位数算术右移，结果存入目标寄存器（R 型）。
格式：SRAV rd, rt, rs （rd = rt >> rs）

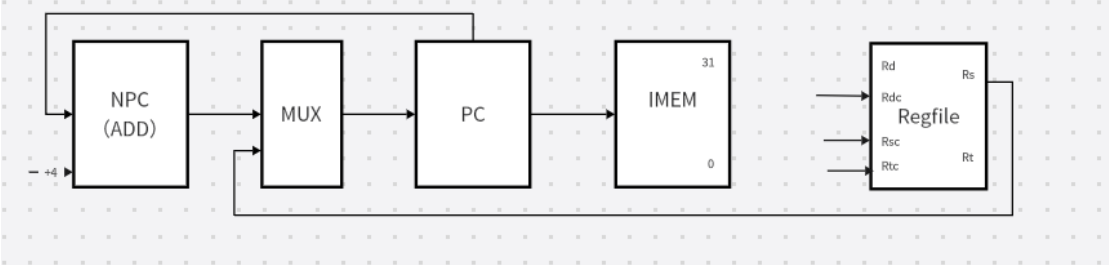


指令流程图

PC -> IMEM
PC + 4 -> NPC
NPC -> PC
Rs[4:0] -> EXT5
EXT5_out -> A
Rt -> B
B >> A -> RES
Res -> R

17.jr
PC ← rs

功能：无条件跳转到由寄存器指定的地址（R 型）。
格式：JR rs
rs：源寄存器，包含目标跳转地址。

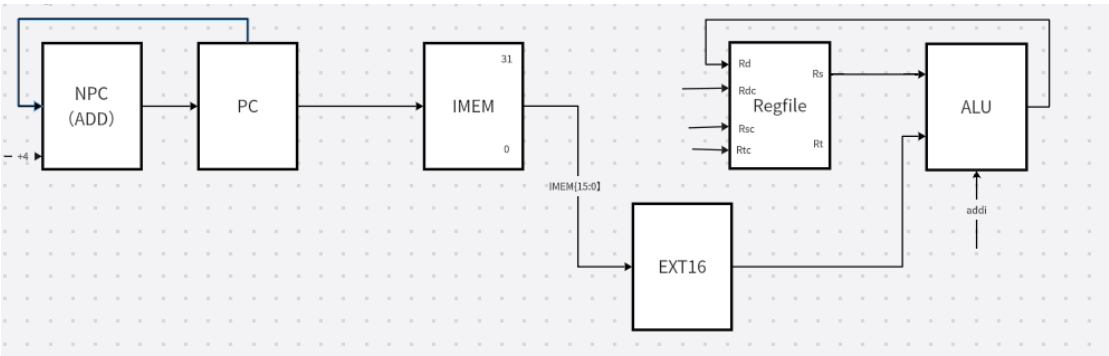


指令流程图

PC -> IMEM
PC + 4 -> NPC
Rs -> MUX
MUX -> PC
NPC -> MUX

I type:

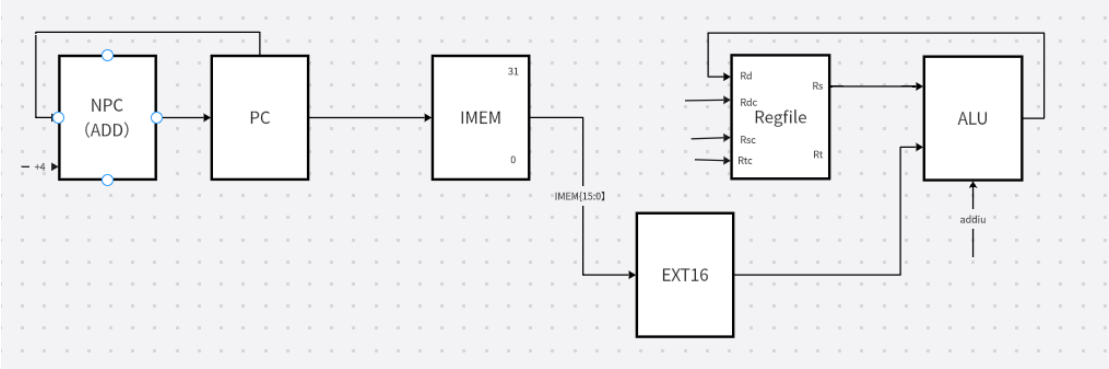
18.addi
rt ← rs + immediate
功能：将寄存器值与立即数相加，结果存入目标寄存器（I 型）。
格式：ADDI rt, rs, imm （rt = rs + imm）



指令流程图

PC -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[15:0] -> EXT16
EXT16_out -> B
Rs -> A
A + B -> RES
Res -> Rd

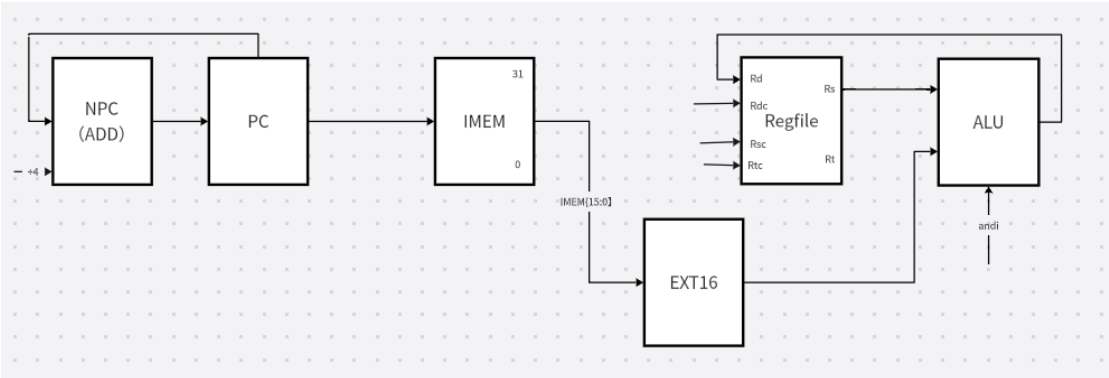
19.addiu
rt ← rs + immediate
功能：类似 ADDI，但不检查溢出（I 型）。
格式：ADDIU rt, rs, imm



指令流程图

PC -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[15:0] -> EXT16
EXT16_out -> B
Rs -> A
A + B -> RES
Res -> Rd

20.andi
rt ← rs AND immediate
功能：寄存器值与立即数按位与，结果存入目标寄存器（I 型）。
格式：ANDI rt, rs, imm （rt = rs & imm）

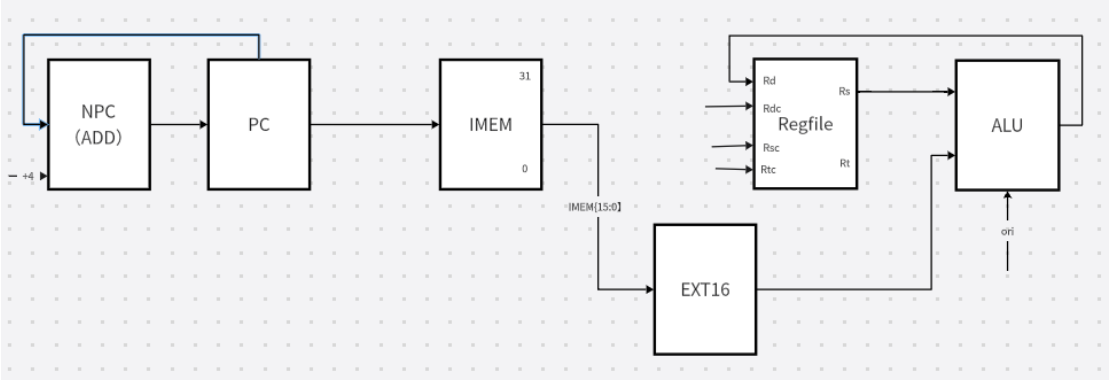


指令流程图

PC -> IMEM
PC + 4 -> NPC
NPC -> PC
Rs -> A

Rt -> B
A & B -> RES
RES -> Rd

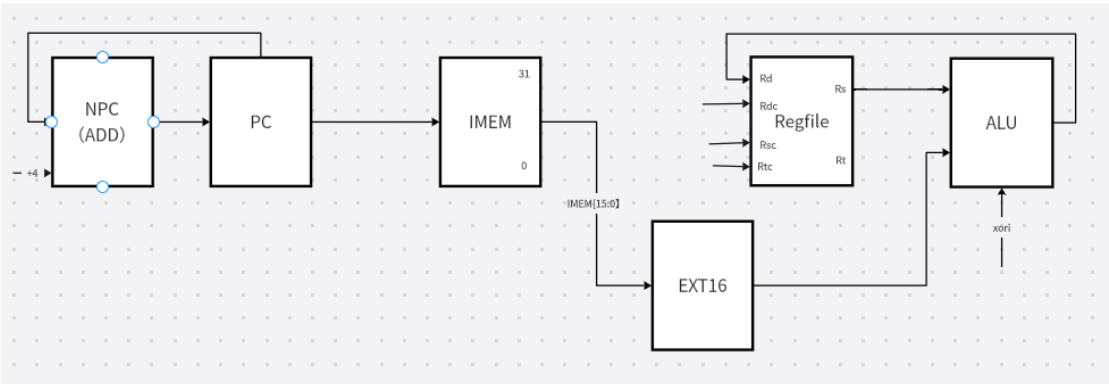
21.ori
rt ← rs or immediate
功能：寄存器值与立即数按位或，结果存入目标寄存器（I 型）。
格式：ORI rt, rs, imm （rt = rs | imm）



指令流程图

PC -> IMEM
PC + 4 -> NPC
NPC -> PC
Rs -> A
Rt -> B
A | B -> RES
RES -> Rd

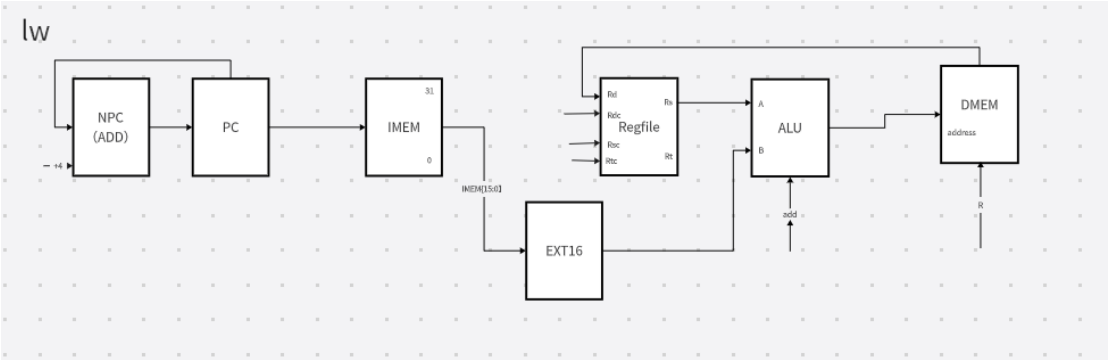
22.xori
rt ←rs XOR immediate
功能：对两个寄存器值按位异或，结果存入目标寄存器（R 型）。
格式：XOR rd, rs, rt （rd = rs ^ rt）



指令流程图

PC -> IMEM
PC + 4 -> NPC
NPC -> PC
Rs -> A
Rt -> B
A ⊕ B -> RES
RES -> Rd

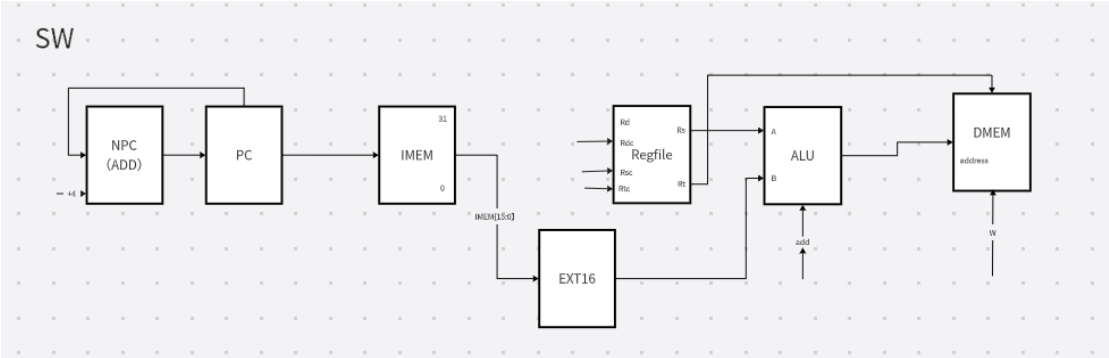
23.lw
rt ← memory[base+offset]
功能：从内存加载 32 位字到寄存器（I 型）。
格式：LW rt, offset(rs) （rt = memory[rs + offset]）



指令流程图

PC -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[15:0] -> EXT16
EXT16_out -> B
Rs -> A
A + B -> RES
Res -> DMEM_addr
DMEM_out -> Rd

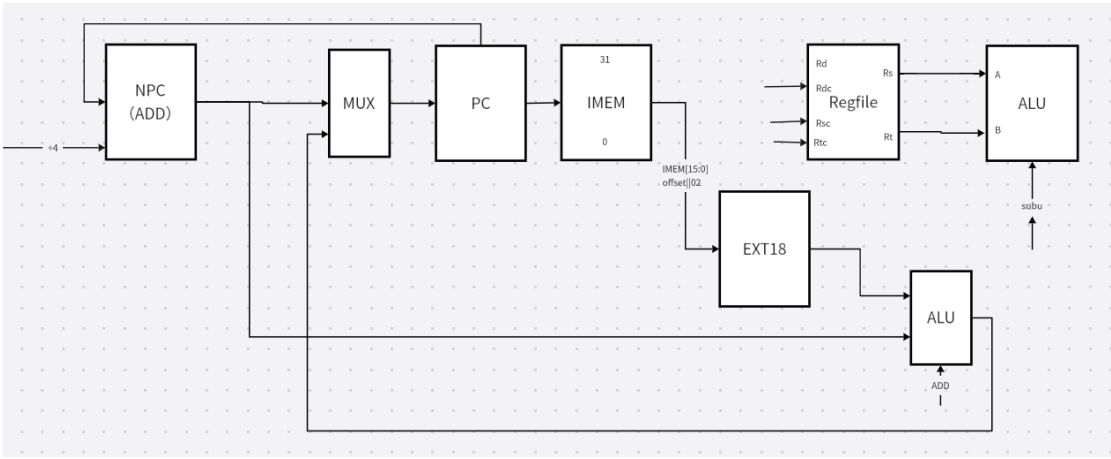
24.sw
memory[base+offset] ←r
功能：将寄存器中的 32 位字存储到内存（I 型）。
格式：SW rt, offset(rs) （memory[rs + offset] = rt



指令流程图

PC -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[15:0] -> EXT16
EXT16_out -> B
Rs -> A
A + B -> RES
Rt -> DMEM
Res -> DMEM_addr

25.beq
功能：如果两个寄存器值相等，则跳转到目标地址（I 型）。
格式：BEQ rs, rt, offset （if rs == rt, PC += offset << 2）

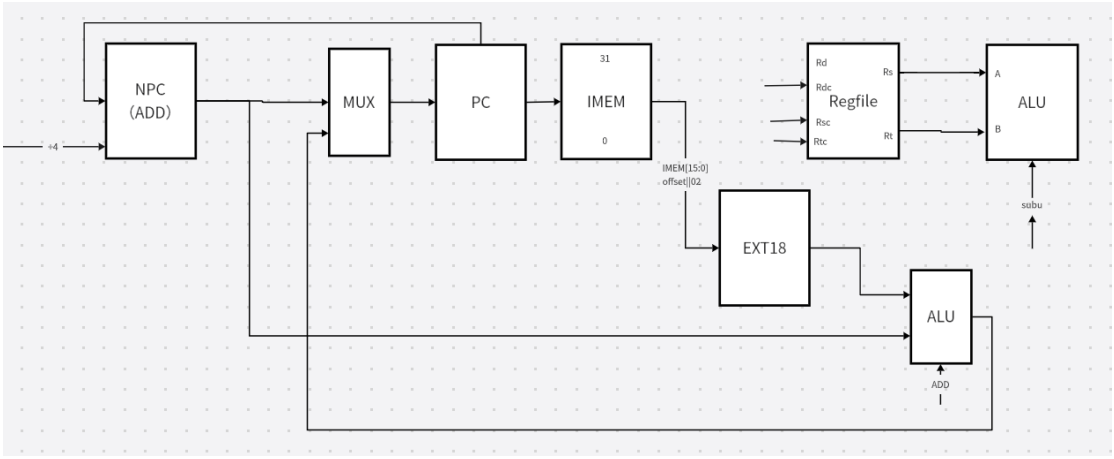


指令流程图

PC -> IMEM
PC + 4 -> NPC
NPC -> MUX
IMEM[15:0] || 02 -> EXT18
EXT18_out -> ADD
NPC -> ADD
ADD_out -> MUX

MUX_out -> PC
Rs -> A
Rt -> B
A + B -> RES
Z -> MUX
MUX -> PC

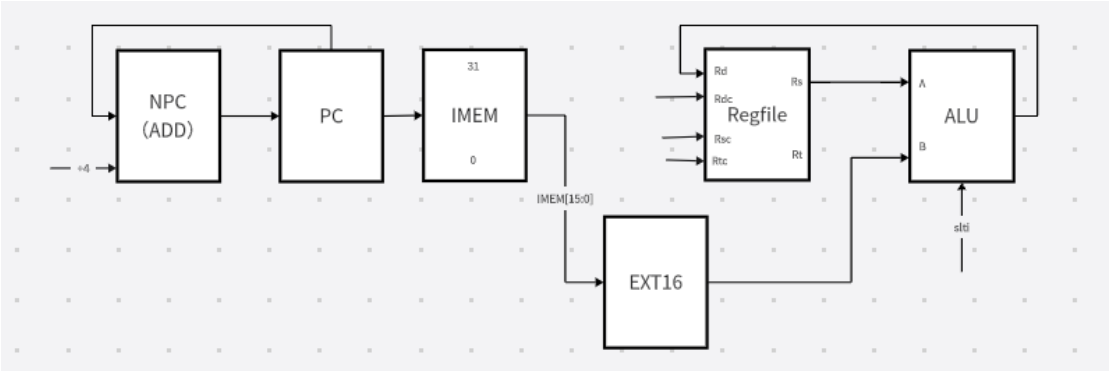
26.bne
功能：如果两个寄存器值不相等，则跳转到目标地址（I 型）。
格式：BNE rs, rt, offset



指令流程图

C -> IMEM
PC + 4 -> NPC
NPC -> MUX
IMEM[15:0] || 02 -> EXT18
EXT18_out -> ADD
NPC -> ADD
ADD_out -> MUX
Rs -> A
Rt -> B
A + B -> RES
Z -> MUX
MUX -> PC

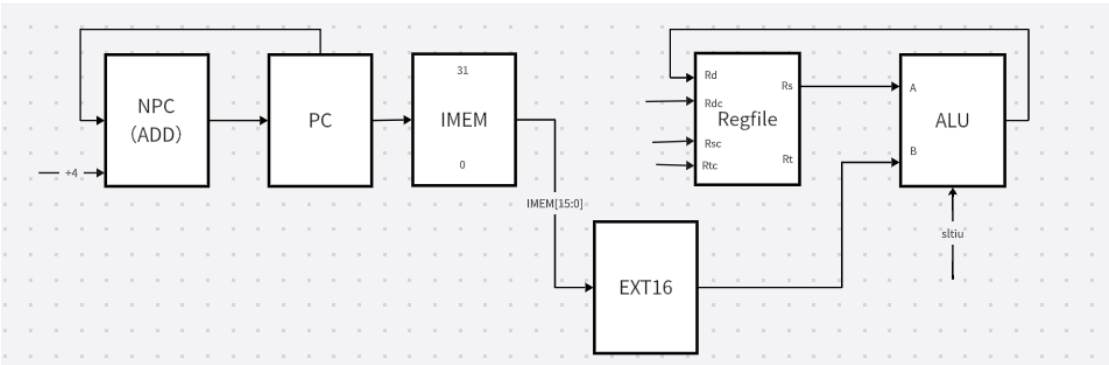
27.slti
rt ←(rs < immediate)
功能：寄存器值与立即数比较，若 rs < imm，则目标寄存器置 1，否则置 0（I 型）。
格式：SLTI rt, rs, imm （rt = (rs < imm) ? 1 : 0）



指令流程图

PC -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[15:0] -> EXT16
EXT16_out -> B
Rs -> A
CF -> EXT32
EXT32_out -> Rd

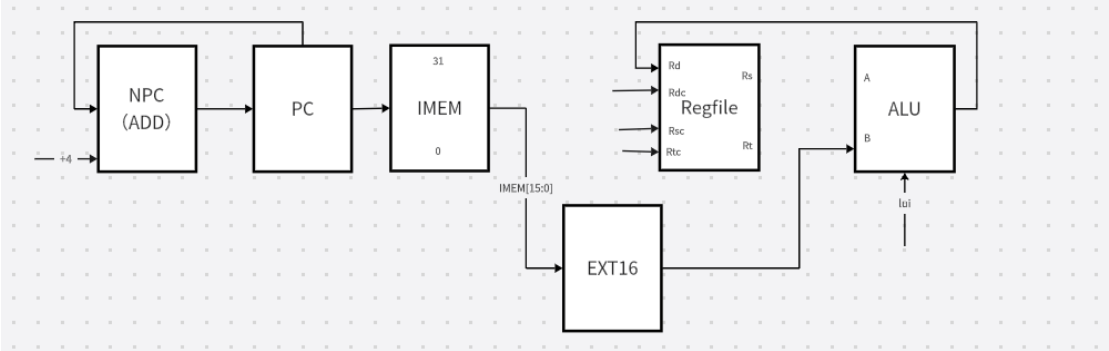
28.sltiu
rt ←(rs < immediate)
功能：类似 SLTI，但进行无符号比较（I 型）。
格式：SLTIU rt, rs, imm



指令流程图

C -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[15:0] -> EXT16
EXT16_out -> B
Rs -> A
CF -> EXT32
EXT32_out -> Rd

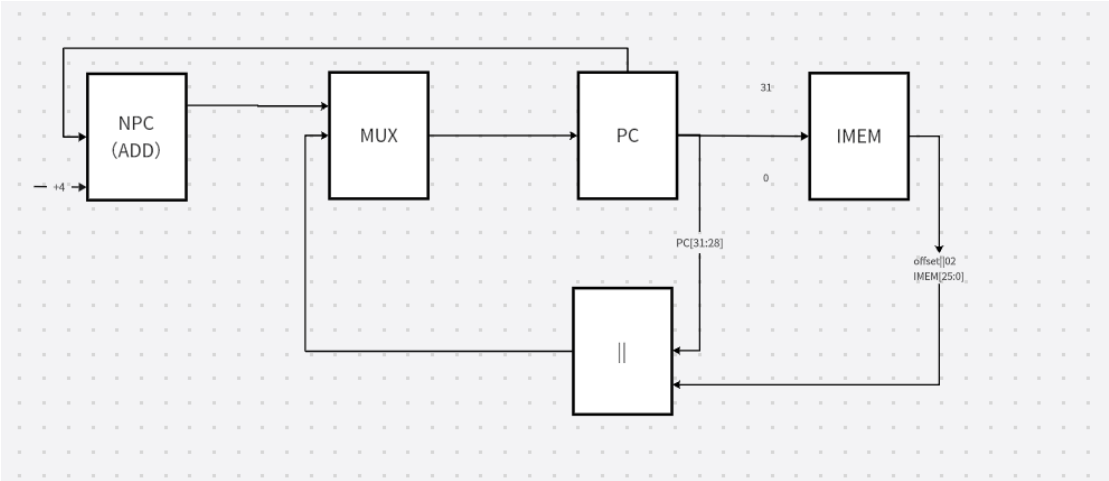
29.lui
rt ← immediate || 016
功能：将 16 位立即数加载到寄存器的高 16 位，低 16 位清零（I 型）。
格式：LUI rt, imm



指令流程图

C -> IMEM
PC + 4 -> NPC
NPC -> PC
IMEM[15:0] -> EXT16
EXT16_out -> B
Res -> Rd

J type
30.j
功能：无条件跳转到目标地址（J 型）。
格式：J target （PC = target << 2）

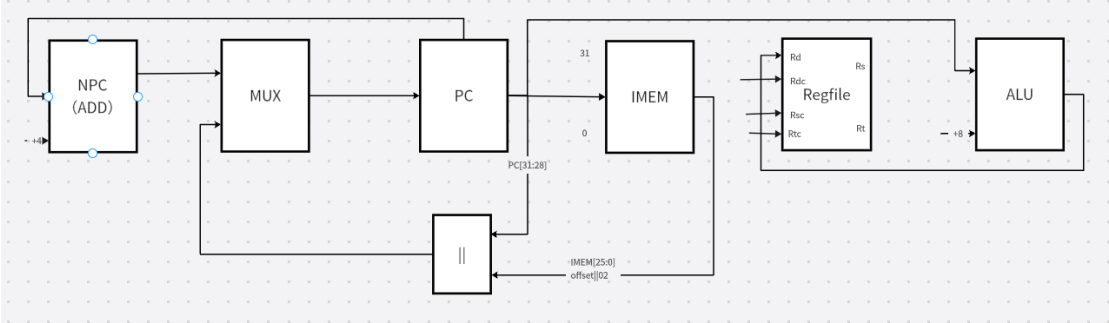


指令流程图

PC -> IMEM
PC[31:28] -> ||_A
IMEM[25,0] || 02 -> ||_B
||_out -> MUX

PC + 4 -> NPC
NPC -> MUX
MUX_out -> PC

31.jal
功能：无条件跳转到目标地址，并保存返回地址（J 型）。
格式：JAL target
target：26 位目标地址（左移 2 位后与 PC 高 4 位组合）。

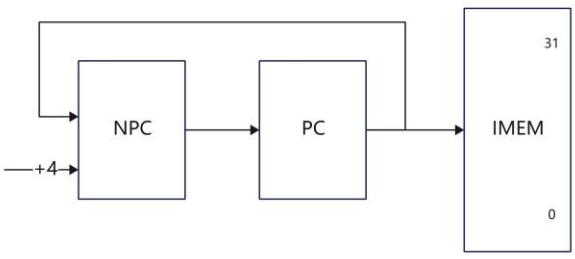


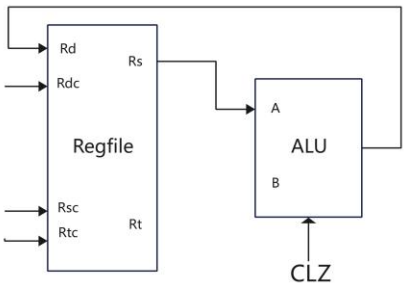
指令流程图

PC -> IMEM
PC[31:28] -> ||_A
IMEM[25,0] || 02 -> ||_B
||_out -> MUX
PC + 4 -> NPC
NPC -> MUX
MUX_out -> PC
PC -> ADD
+8 -> ADD
ADD_out -> Rd

以上为 31 条指令 CPU 的数据通路
下面是 23 条扩展指令

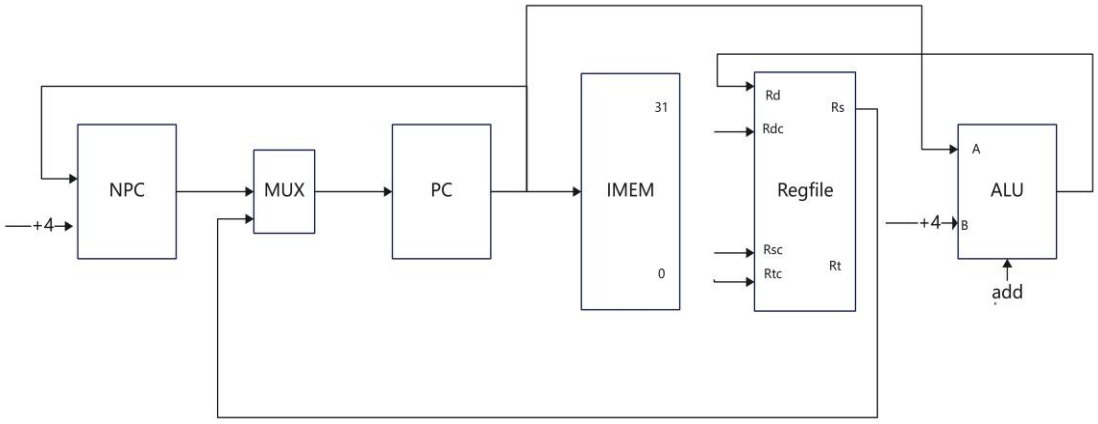
32.clz
功能：计算 32 位字中的前导零个数





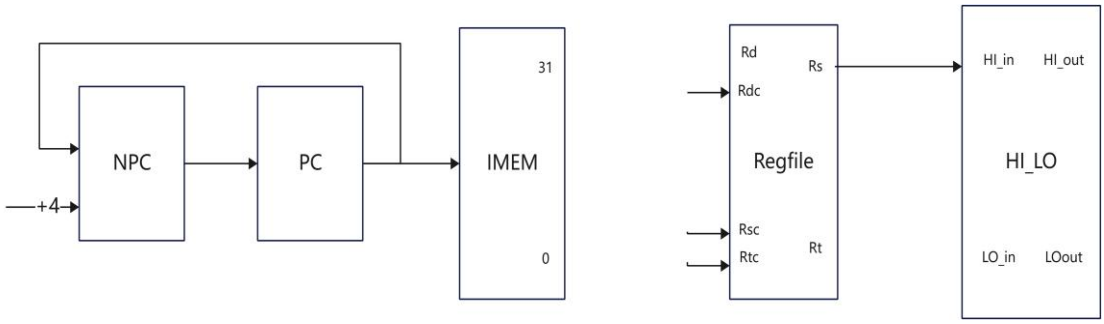
指令流程图
PC -> IMEM
PC + 4 -> NPC

NPC -> PC
Rs -> ALU(clz)->Rd
33. jalr
功能：执行一个在寄存器中存放指令地址的过程调用



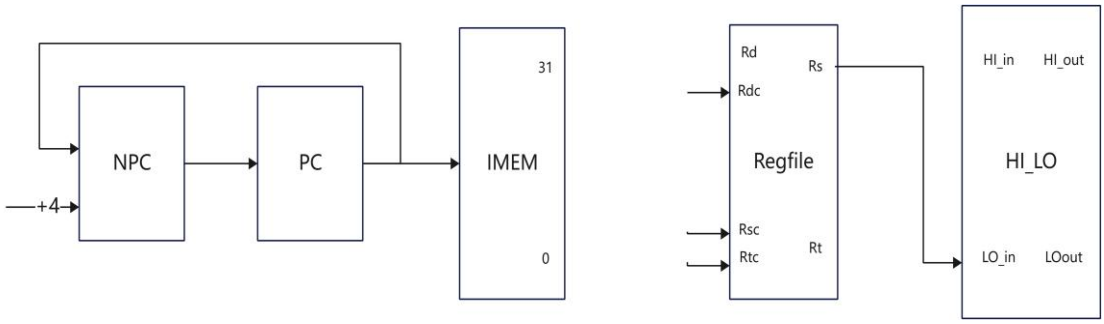
指令流程图
PC -> IMEM
PC -> ALU_A
4->ALU_B
PC + 4 -> NPC
NPC -> MUX
Rs -> MUX
MUX -> PC
ALU_out -> Rd

34.mthi
功能：复制通用寄存器的内容到特殊寄存器 HI 中



指令流程图
PC -> IMEM PC + 4 -> NPC NPC -> PC Rs -> HI_in

35.mtlo
功能：复制通用寄存器的内容到特殊寄存器 LO 中

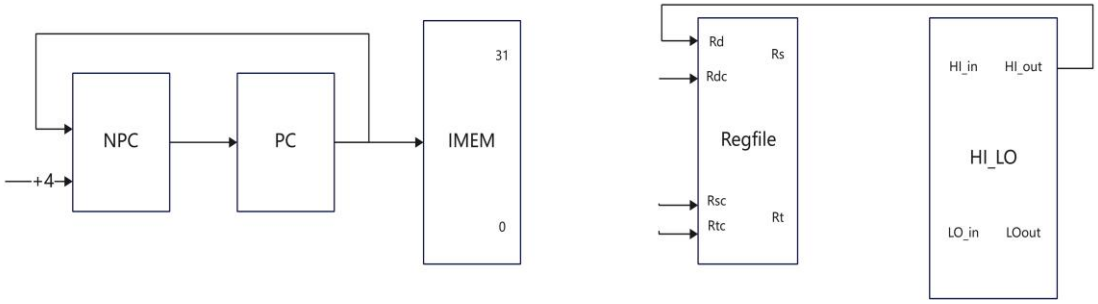


指令流程图

PC -> IMEM PC + 4 -> NPC NPC -> PC Rs> LO_in

36.mfhi

功能：复制特殊寄存器 HI 的内容到通用寄存器

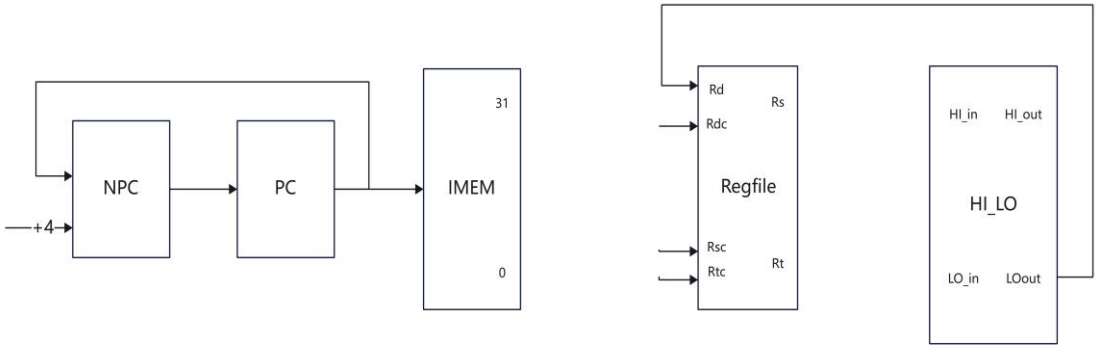


指令流程图：

PC -> IMEM PC + 4 -> NPC NPC -> PC HI_out -> Rd

37.mflo

功能：复制通用寄存器的内容到特殊寄存器 LO 中

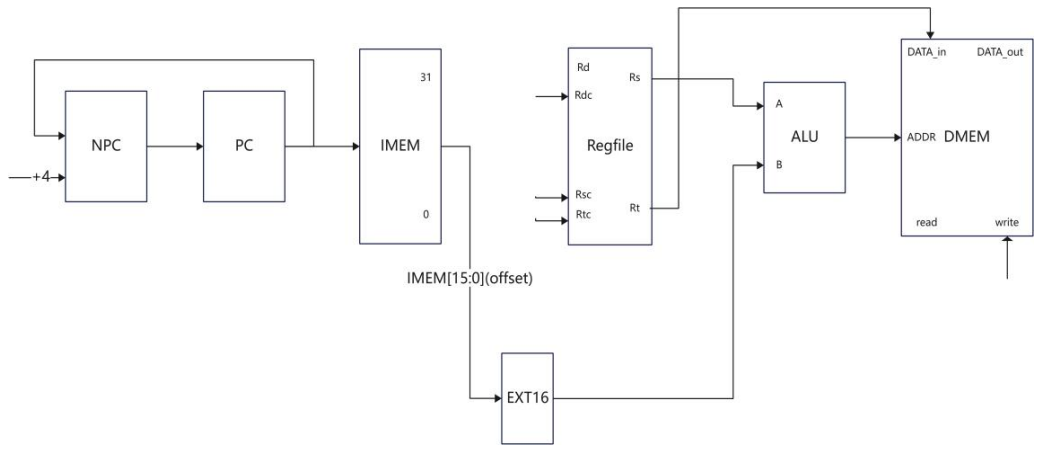


指令流程图：

PC -> IMEM PC + 4 -> NPC NPC -> PC LO_out -> Rd

38.sb

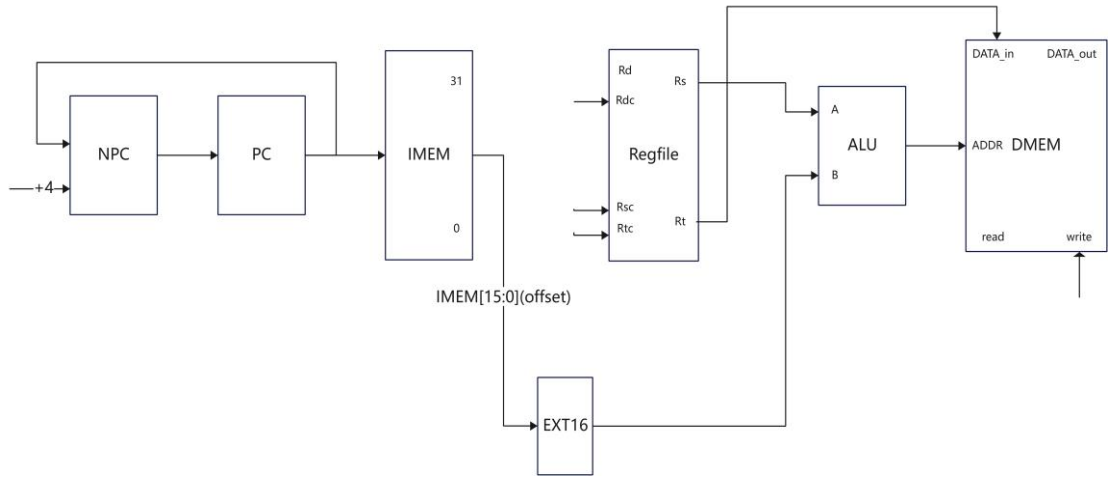
功能：存一字节到内存



指令流程图：

PC -> IMEM PC + 4 -> NPC NPC -> PC
IMEM[15:0] -> EXT16 Rs -> ALU_A EXT16 -> ALU_B
ALU_out -> ADDR Rt -> DATA_IN

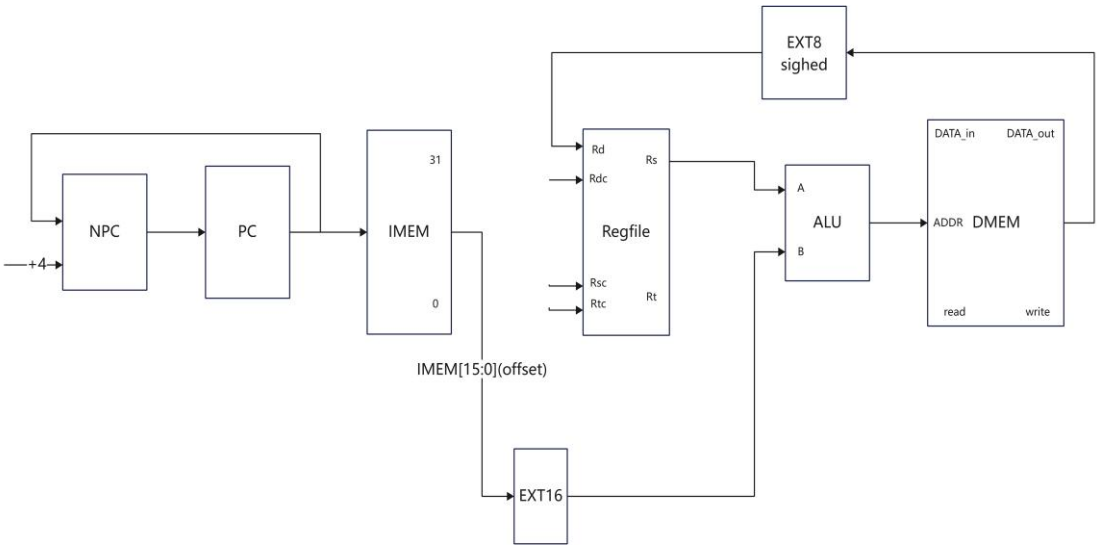
39.sh
功能：存一个半字到内存



指令流程图：

PC -> IMEM PC + 4 -> NPC NPC -> PC
IMEM[15:0] -> EXT16 Rs -> ALU_A EXT16 -> ALU_B
ALU_out -> ADDR Rt -> DATA_IN

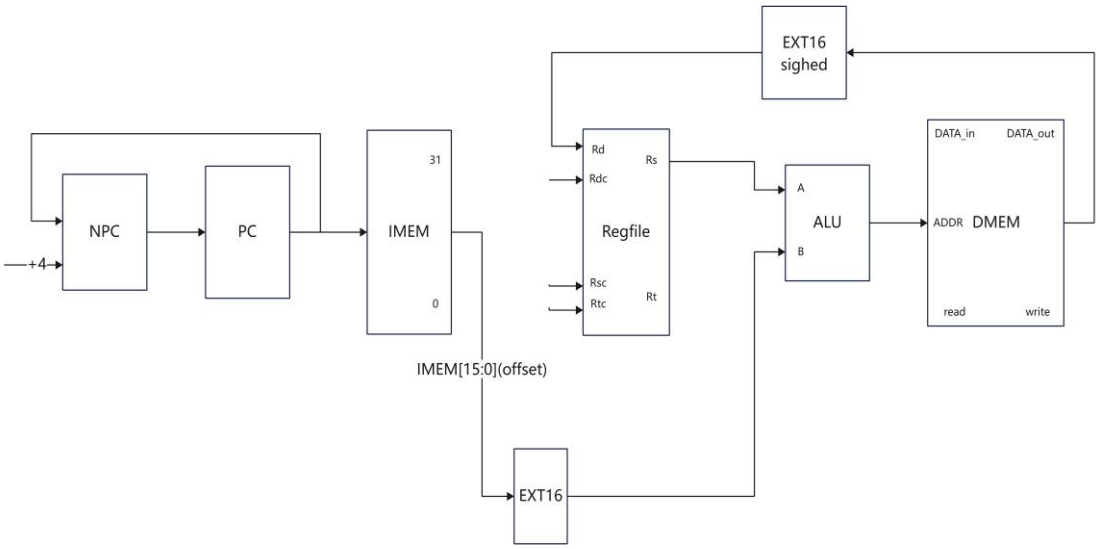
40.lb
功能：从内存加载一个有符号字节



指令流程图：

PC -> IMEM PC + 4 -> NPC NPC -> PC
IMEM[15:0] -> EXT16
Rs -> ALU_A EXT16 -> ALU_B
ALU_out -> ADDR
DATA_OUT -> EXT8signed
EXT8-> Rd

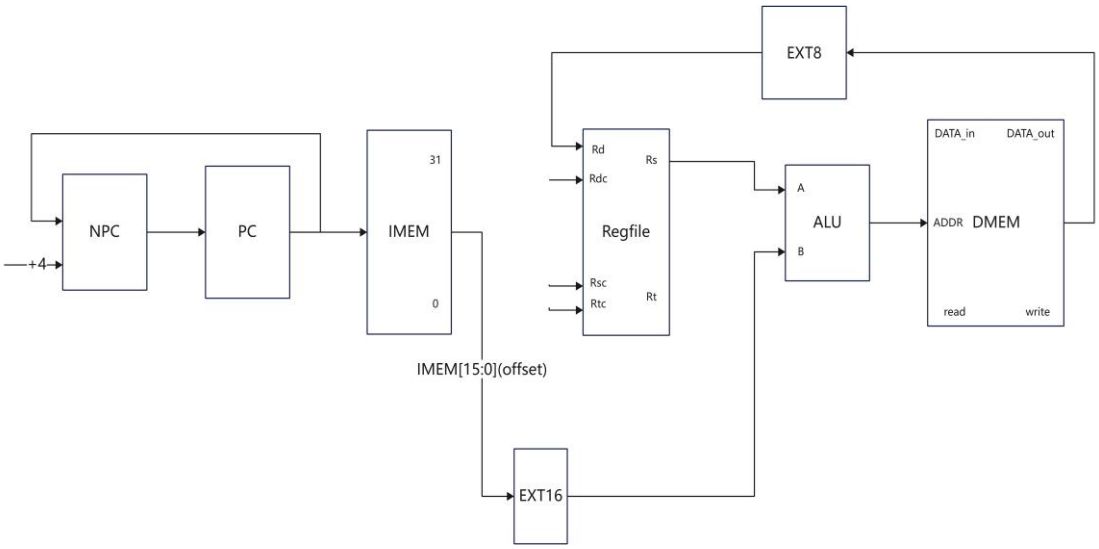
41.lh
功能：从内存加载一个有符号半字



指令流程图：

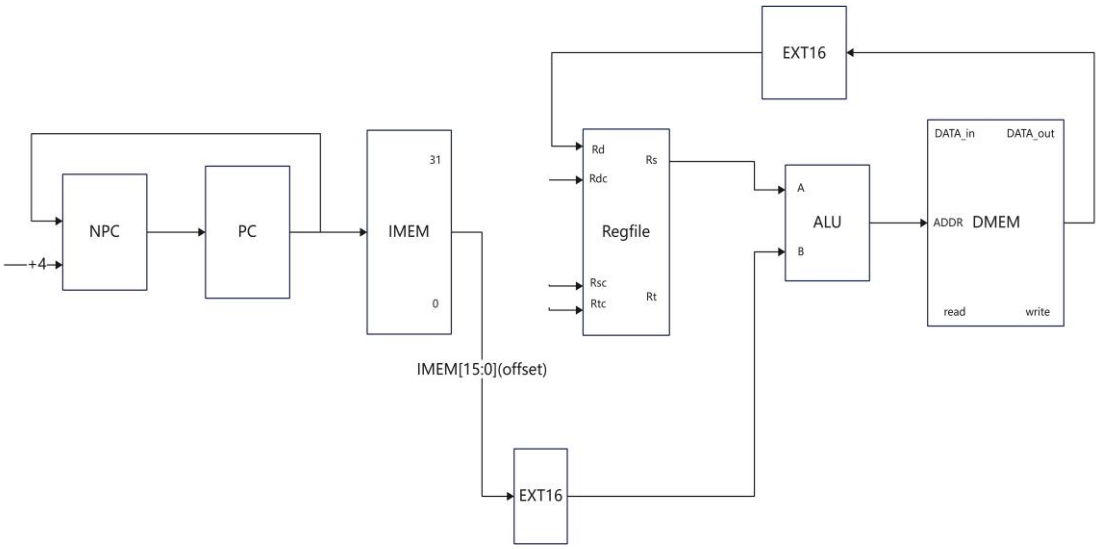
PC -> IMEM PC + 4 -> NPC NPC -> PC
IMEM[15:0] -> EXT16
Rs -> ALU_A EXT16 -> ALU_B
ALU_out -> ADDR
DATA_OUT -> EXT16signed
EXT16-> Rd

42.lbu
功能：从内存加载一个无符号字节



指令流程图：
PC -> IMEM PC + 4 -> NPC NPC -> PC
IMEM[15:0] -> EXT16
Rs -> ALU_A EXT16 -> ALU_B
ALU_out -> ADDR
DATA_OUT -> EXT8signed
EXT8-> Rd

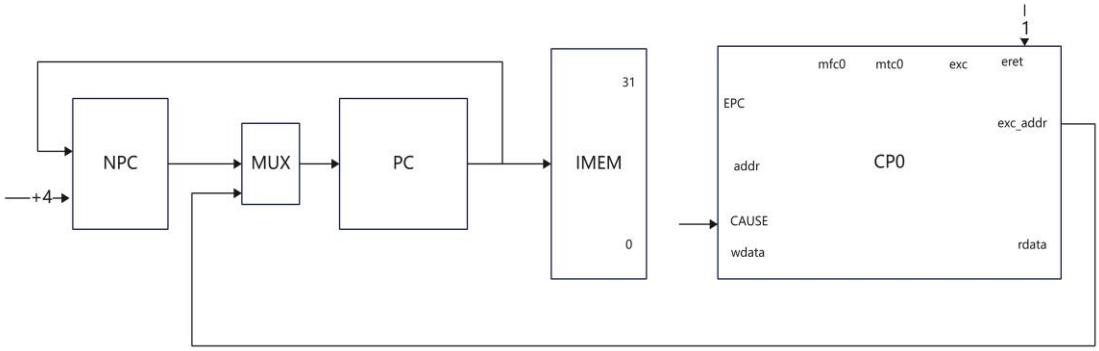
43.lhu
功能：从内存加载一个无符号半字



指令流程图：
PC -> IMEM PC + 4 -> NPC NPC -> PC
IMEM[15:0] -> EXT16
Rs -> ALU_A EXT16 -> ALU_B
ALU_out -> ADDR
DATA_OUT -> EXT16

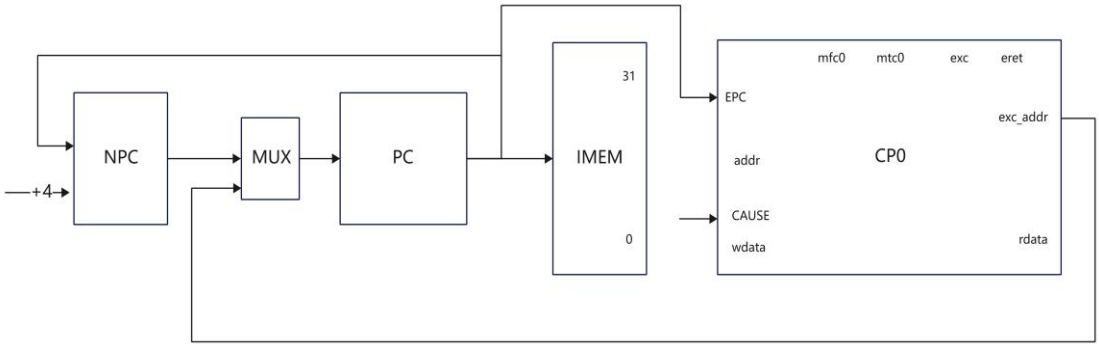
EXT16-> Rd

44.eret
功能：从异常中断返回



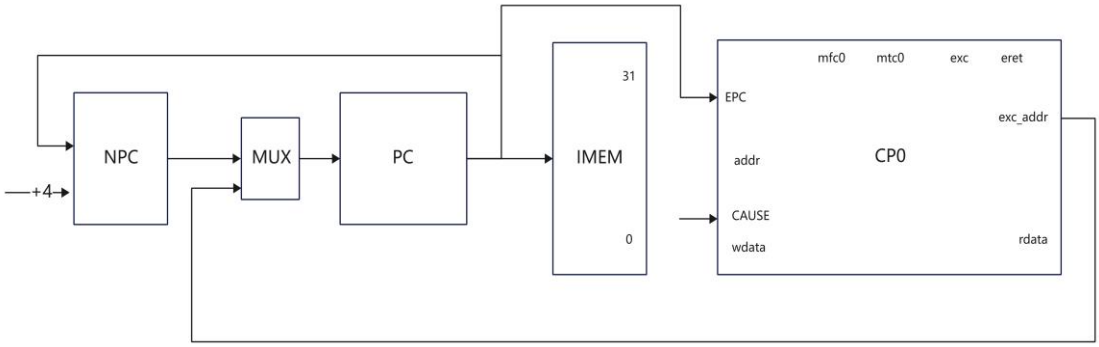
指令流程图：
PC -> IMEM PC + 4 -> NPC
NPC -> MUX EPC_OUT -> MUX
MUX-> PC

45.break
功能：引起一个断点异常



指令流程图：
PC -> IMEM PC + 4 -> NPC
NPC -> MUX
PC -> CP0
0x4 -> MUX
MUX -> PC

46.syscall
功能：引发一个系统调用异常

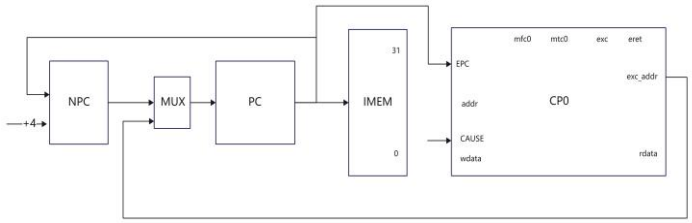


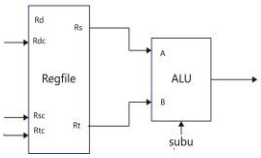
指令流程图：

PC -> IMEM PC + 4 -> NPC
NPC -> MUX
PC -> CP0
0x4 -> MUX
MUX -> PC

47.teq

功能：比较寄存器值根据条件引发异常



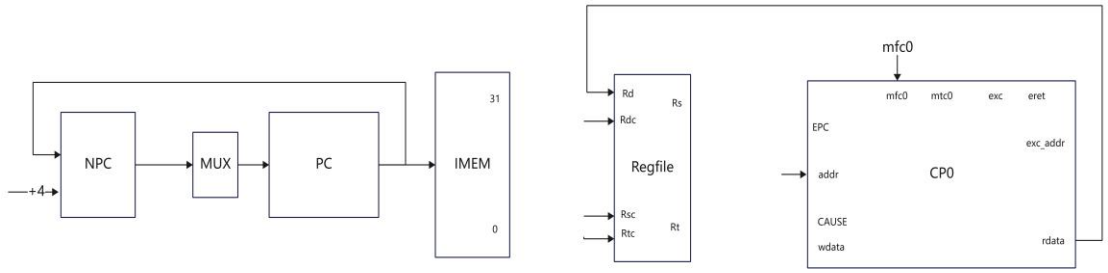


指令流程图：

PC -> IMEM PC + 4 -> NPC
NPC -> MUX
Exc_addr -> MUX
Rs -> ALU_A Rt -> ALU_B
ZERO -> Rs – Rt == 0
Exc_addr -> PC if ZERO == 0

48.mfc0

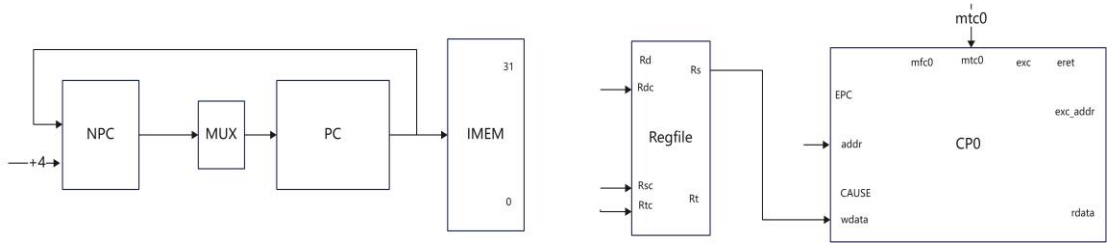
功能：把一个数据从特殊寄存器移到通用寄存器



指令流程图：

PC -> IMEM PC + 4 -> NPC NPC -> PC
Rt -> ADDR rdata -> Rd

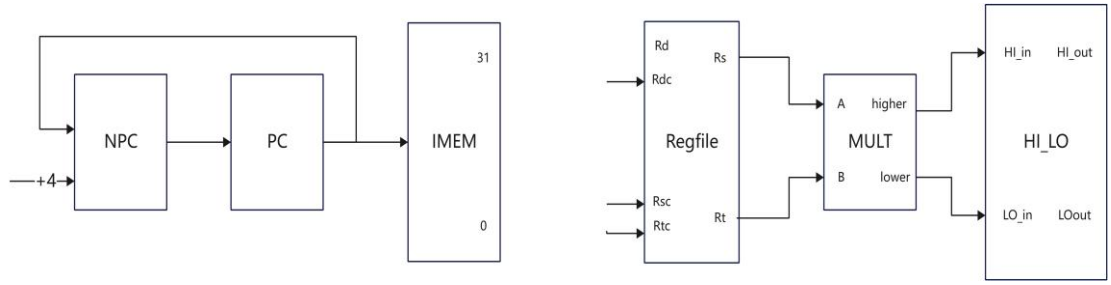
49.mtc0
功能：把一个数据从通用寄存器移到特殊寄存器



指令流程图：

PC -> IMEM PC + 4 -> NPC NPC -> PC Rt -> ADDR Rs -> wdata

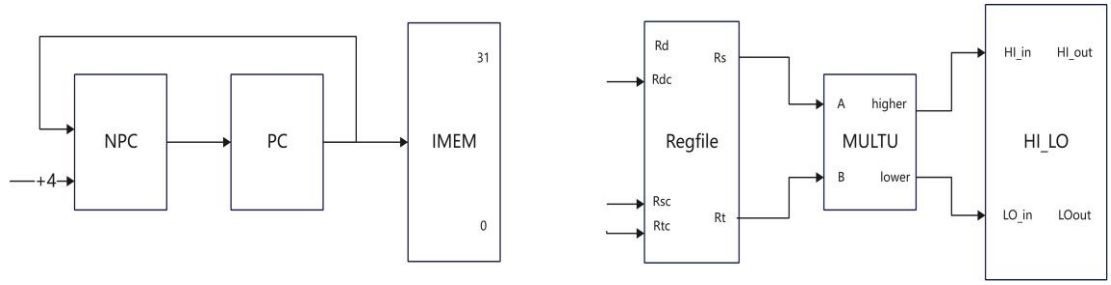
50.mult
功能：32 位有符号数乘法



指令流程图：

PC -> IMEM PC + 4 -> NPC NPC -> PC
Rs -> MULT_A
Rt -> MULT_B
MULT -> res
res[31:0] -> LO_in
res[63:32] -> HI_in

51.multu
功能：32 位无符号数乘法

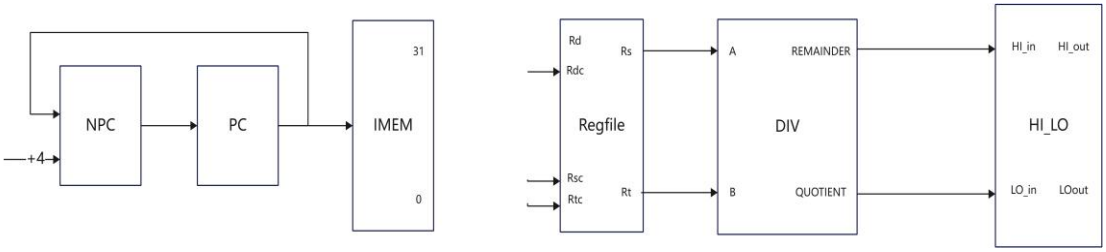


指令流程图：

PC -> IMEM PC + 4 -> NPC NPC -> PC
Rs -> MULTU_A
Rt -> MULTU_B
MULTU -> res
res[31:0] -> LO_in
res[63:32] -> HI_in

52.div

功能：有符号除法

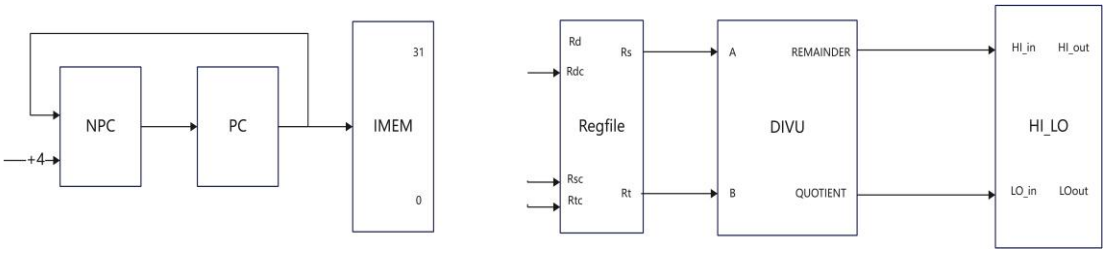


指令流程图：

PC -> IMEM PC + 4 -> NPC NPC -> PC
Rs -> DIV_A
Rt -> DIV_B
DIV-> res
res[31:0] -> LO_in
res[63:32] -> HI_in

53.divu

功能：32 位无符号数除法

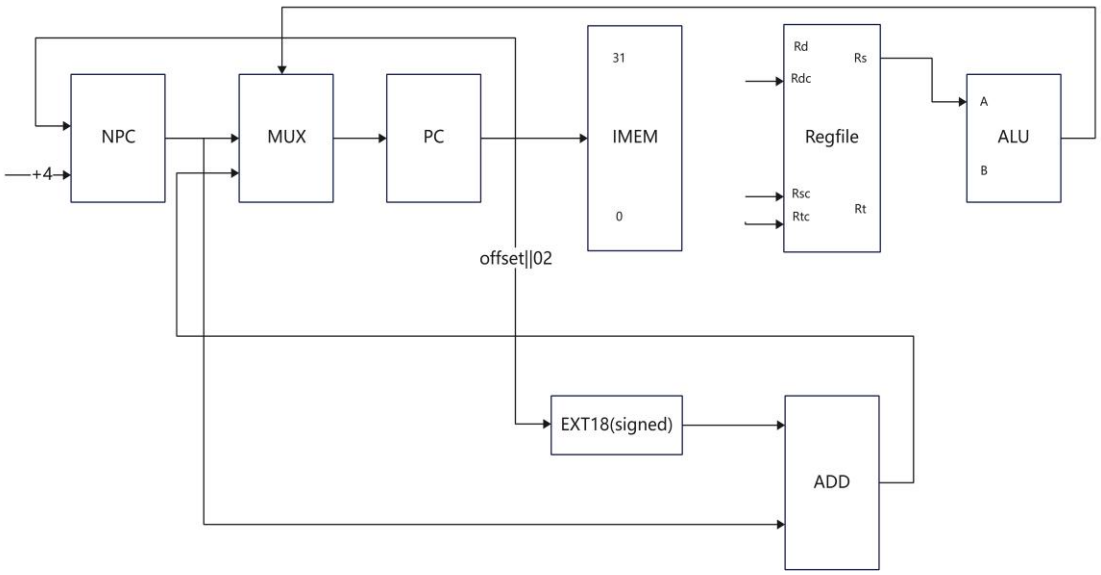


指令流程图：

PC -> IMEM PC + 4 -> NPC NPC -> PC
Rs -> DIVU_A
Rt -> DIVU_B
DIVU-> res
res[31:0] -> LO_in
res[63:32] -> HI_in

54.bgez

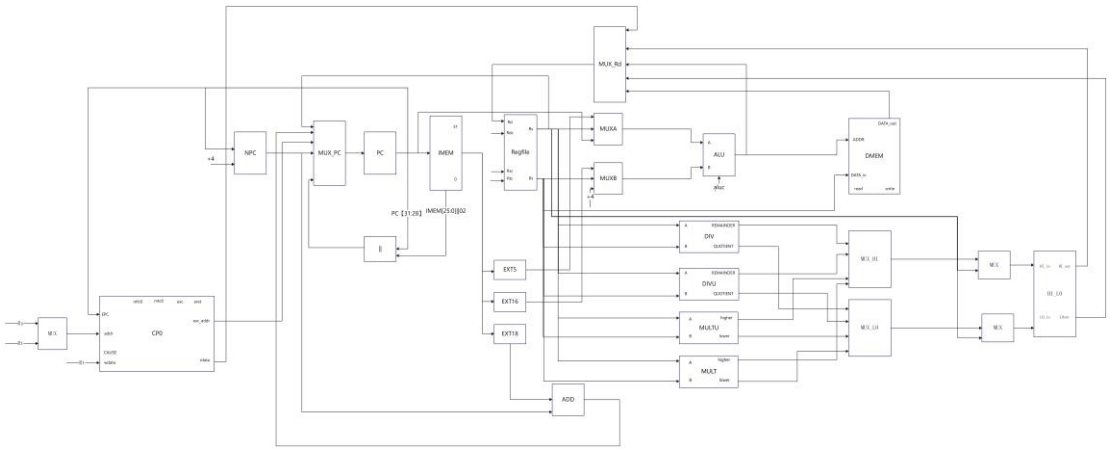
功能：检测通用寄存器的值然后进行 pc 相关的跳转



指令流程图：

PC -> IMEM PC + 4 -> NPC NPC -> MUX
Rs -> ALU_A
0 -> ALU_B
Rs – 0 -> res
IMEM[15:0] || 02 -> EXT18
EXT18 -> ADD_A
NPC -> ADD_B
ADD -> MUX
MUX->PC

（3）数据通路总图



三、CPU 控制部件设计

报告要求：
（1）根据指令流程图，编排指令取指到执行的操作时间表。

（2）根据指令操作时间表，写出每个控制信号的逻辑表达式。
（3）根据逻辑表达式，完成控制部件设计。

部件输入输出表

		PC	NPC	IMEM	Regfile		ALU		Ext5	Ext16	Ext18	DMEM		ADD		HI_LO		CP0					MUL		DIV	
					rd	rdc	A	B				Addr	Data	A	B	HI_in	LO_in	Cause	EPC	status	addr	wdata	A	B	A	B
1	add	NPC	PC	PC	ALU	IMEM[15:11]	RS	rt																		
2	addu	NPC	PC	PC	ALU	IMEM[15:11]	RS	rt																		
3	sub	NPC	PC	PC	ALU	IMEM[15:11]	rs	rt																		
4	subu	NPC	PC	PC	ALU	IMEM[15:11]	rs	rt																		
5	and	NPC	PC	PC	ALU	IMEM[15:11]	rs	rt																		
6	or	NPC	PC	PC	ALU	IMEM[15:11]	rs	rt																		
7	xor	NPC	PC	PC	ALU	IMEM[15:11]	rs	rt																		
8	nor	NPC	PC	PC	ALU	IMEM[15:11]	rs	rt																		
9	slt	NPC	PC	PC	ALU	IMEM[15:11]	rs	rt																		
10	sltu	NPC	PC	PC	ALU	IMEM[15:11]	rs	rt																		
11	sll	NPC	PC	PC	ALU	IMEM[15:11]	Ext5	rt	sa																	
12	srl	NPC	PC	PC	ALU	IMEM[15:11]	Ext5	rt	sa																	
13	sra	NPC	PC	PC	ALU	IMEM[15:11]	Ext5	rt	sa																	
14	sllv	NPC	pc	PC	ALU	IMEM[15:11]	Rs	Rt																		
15	srlv	NPC	PC	PC	ALU	IMEM[15:11]	Rs	Rt																		
16	srav	NPC	PC	PC	ALU	IMEM[15:11]	RS	Rt																		
17	jr	Rs	PC																							
18	addi	NPC	PC	PC	ALU	IMEM[15:11]	Rs	Ext16		IMEM[15:0]																
19	addiu	NPC	PC	PC	ALU	IMEM[15:11]	Rs	Ext16		IMEM[15:0]																
20	andi	NPC	PC	PC	ALU	IMEM[15:11]	Rs	Ext16		IMEM[15:0]																
21	ori	NPC	PC	PC	ALU	IMEM[15:11]	Rs	Ext16		IMEM[15:0]																
22	xori	NPC	PC	PC	ALU	IMEM[15:11]	Rs	Ext16		IMEM[15:0]																
23	lw	NPC	PC	PC	DMEM	IMEM[20:16]	Rs	Ext16		IMEM[15:0]		ALU														
24	sw	NPC	PC	PC			RS	Ext16		IMEM[15:0]		ALU	Rt													
25	beq	MUX	PC	PC			Rs	Rt			offset||02			Ext18	NPC											
26	bne	MUX	PC	PC			Rs	Rt			offset||02			Ext18	NPC											
27	slti	NPC	PC	PC	ALU	IMEM[20:16]	Rs	Ext16		IMEM[15:0]																
28	sltiu	NPC	PC	PC	ALU	IMEM[20:16]	Rs	Ext16		IMEM[15:0]																
29	lui	NPC	PC	PC	ALU	IMEM[20:16]		Ext16		IMEM[15:0]																
30	j	MUX	PC	PC																						
31	jal	MUX	PC	PC	ALU		PC	8																		
32	clz	NPC	PC	PC	ALU(CLZ)	IMEM[15:11]	RS	/																		
33	jalr	MUX	PC	PC	NPC	SRa																				
34	mthi	NPC	PC	PC												RS										
35	mtlo	NPC	PC	PC													RS									
36	mfhi	NPC	PC	PC	HI_out	IMEM[15:11]																				
37	mflo	NPC	PC	PC	LO_out	IMEM[15:11]																				
38	sb	NPC	PC	PC			Rs	EXT16		IMEM[15:0]		ALU	Rt													
39	sh	NPC	PC	PC			Rs	EXT16		IMEM[15:0]		ALU	Rt													
40	lb	NPC	PC	PC	DMEM	IMEM[20:16]	Rs	EXT8(signed)		IMEM[15:0]		ALU														
41	lbu	NPC	PC	PC	DMEM	IMEM[20:16]	Rs	EXT8		IMEM[15:0]		ALU														
42	lh	NPC	PC	PC	DMEM	IMEM[20:16]	Rs	EXT16(signed)		IMEM[15:0]		ALU														
43	lhu	NPC	PC	PC	DMEM	IMEM[20:16]	Rs	EXT16		IMEM[15:0]		ALU														
44	eret	exc_addr	PC	PC																status>>5						
45	break	exc_addr	PC	PC														01001	PC	status<<5						
46	syscall	exc_addr	PC	PC														01000	PC	status<<5						
47	teq	MUX	PC	PC			Rs	Rt										01101	PC	status<<5						
48	mfc0	NPC	PC	PC	CP0,rdata	IMEM[20:16]															Rt					
49	mtc0	NPC	PC	PC		IMEM[20:16]															Rd	Rt				
50	mult	NPC	PC	PC												MUL[63:32]	MUL[31:0]						Rs	Rt		
51	multu	NPC	PC	PC												MUL[63:32]	MUL[31:0]						Rs	Rt		
52	div	NPC	PC	PC												DIV.R	DIV.Q								Rs	Rt
53	divu	NPC	PC	PC												DIV.R	DIV.Q								Rs	Rt
54	bgez	MUX	PC	PC			Rs	Rt			IMEM[15:0]||02			Ext18	NPC											

控制信号表

		IMEM_r	data_w	data_r	ALUC	reg_w	MUX_PC	MUX_Rd	MUX_A	MUX_B	MUX_HI	MUX_LO
1	add	1	0	0	00000	1	000	000	00	00		
2	addu	1	0	0	00001	1	000	000	00	00		
3	sub	1	0	0	00010	1	000	000	00	00		
4	subu	1	0	0	00011	1	000	000	00	00		
5	and	1	0	0	00100	1	000	000	00	00		
6	or	1	0	0	00101	1	000	000	00	00		
7	xor	1	0	0	00110	1	000	000	00	00		
8	nor	1	0	0	00111	1	000	000	00	00		
9	slt	1	0	0	01000	1	000	000	00	00		
10	sltu	1	0	0	01001	1	000	000	00	00		
11	sll	1	0	0	01010	1	000	000	10	00		
12	srl	1	0	0	01011	1	000	000	10	00		
13	sra	1	0	0	01100	1	000	000	10	00		
14	sllv	1	0	0	01101	1	000	000	00	00		
15	srlv	1	0	0	01110	1	000	000	00	00		
16	srav	1	0	0	01111	1	000	000	00	00		
17	jr	1	0	0			001					
18	addi	1	0	0	00000	1	000	000	00	01		
19	addiu	1	0	0	00001	1	000	000	00	01		
20	andi	1	0	0	00100	1	000	000	00	01		
21	ori	1	0	0	00101	1	000	000	00	01		
22	xori	1	0	0	00110	1	000	000	00	01		
23	lw	1	0	1	00000	1	000	001	00	01		
24	sw	1	1	0	00000	0	000		00	01		
25	beq	1	0	0	00011	0	zero?11:00		00	00		
26	bne	1	0	0	00011	0	~zero?11:00		00	00		
27	slti	1	0	0	01000	1	000	000	00	01		
28	sltiu	1	0	0	01001	1	000	000	00	01		
29	lui	1	0	0	10000	1	000	000		01		
30	j	1	0	0		0	010					
31	jal	1	0	0	00000	1	010	000	01	10		
32	clz	1	0	0	10001	1	000	000	00			
33	divu	1	0	0		0	000				000	000
34	div	1	0	0		0	000				001	001
35	mult	1	0	0		0	000				100	100
36	multu	1	0	0		0	000				010	010
37	break	1	0	0		0	100					
38	syscall	1	0	0		0	100					
39	teq	1	0	0			100		00	00		
40	eret	1	0	0		0	100					
41	mfc0	1	0	0		1	000	100				
42	mtc0	1	0	0		0	000					
43	mfhi	1	0	0		1	000	010				
44	mflo	1	0	0		1	000	011				
45	mthi	1	0	0		0	000				011	
46	mtlo	1	0	0		0	000					011
47	jalr	1	0	0	00000	1	001	000	01	10		
48	lb	1	0	1	00000	1	000	001	00	01		
49	lbu	1	0	1	00000	1	000	001	00	01		
50	lh	1	0	1	00000	1	000	001	00	01		
51	lhu	1	0	1	00000	1	000	001	00	01		
52	sb	1	1	0	00000	0	000		00	01		
53	sh	1	1	0	00000	0	000		00	01		
54	bgez	1	0	0		0	~rs[31]?000:011					

MUX 选择信号表

MUX_PC	000NPC	001Rs	010拼接器	100CP0	011add	
MUX_Rd	000ALU	001DMEM		100CP0	010HI_out	011LO_out
MUX_A	00Rs	10EXT5	01PC			
MUX_B	00Rt	01EXT16	10 +4			
MUX_HI	000DIVU.R	001DIV.R	100MULT.H	010MULTU.H	011Rs	
MUX_LO	000DIVU.Q	001DIV.Q	100MULT.L	010MULTU.L	011Rd	

四、CPU 前仿真测试结果

报告要求：
（1）对 CPU 前仿真的实验内容和结果进行描述，并给出 modelsim 前仿真波形图的贴图。
（2）对单条指令、网站前仿真所用测试代码 coe 文件等测试的结果进行比对。

前仿真波形图如上
前仿真只显示 pc 地址以及对应从 IMEM 中取出的指令
可见，信号传输无延迟，inst 每次从 ip 核中取出指令，pc 每次加 4，起始地址为 32'h00400000

值得注意的是，所给的 testbench 打印出的文件，指令和对应的寄存器堆并不同步对应，而是相差一个时钟。在使用 mars 的结果进行比对时应该注意

单个指令比对示意
使用 txt_compare 进行部分指令的比对

```
E:\txt_compare>@echo off
txt_compare --file1 add.txt --file2 add_demo.txt --display detailed
比较结果输出:
================================================================================
在指定检查条件下完全一致.
================================================================================
txt_compare --file1 addi.txt --file2 addi_demo.txt --display detailed
比较结果输出:
================================================================================
在指定检查条件下完全一致.
================================================================================
txt_compare --file1 addu.txt --file2 addu_demo.txt --display detailed
比较结果输出:
================================================================================
在指定检查条件下完全一致.
================================================================================
txt_compare --file1 bne.txt --file2 bne_demo.txt --display detailed
比较结果输出:
================================================================================
在指定检查条件下完全一致.
================================================================================
txt_compare --file1 lui.txt --file2 lui_demo.txt --display detailed
比较结果输出:
================================================================================
在指定检查条件下完全一致.
================================================================================
txt_compare --file1 sll.txt --file2 sll_demo.txt --display detailed
比较结果输出:
================================================================================
在指定检查条件下完全一致.
================================================================================
txt_compare --file1 slti.txt --file2 slti_demo.txt --display detailed
比较结果输出:
================================================================================
在指定检查条件下完全一致.
================================================================================
```

注意：

自己写的 cpu 不能及时停止，会比 demo 多出一部分，这部分在比对时需要删除

例如：

```
E:\txt_compare>txt_compare --file1 add.txt --file2 add_demo.txt --display detailed
比较结果输出:
================================================================================
第[341 / 341]行 - 文件1的尾部有多余字符:
        --------------------
        0         1
        01234567890123456789
        --------------------
文件1 : pc: 00400028<CR>
文件2 : <EOF>
文件1(HEX) :
00000000 : 70 63 3a 20 30 30 34 30 - 30 30 32 38              pc:.00400028
文件2(HEX) :
<EOF>

================================================================================
在指定检查条件下共1行有差异.
```

需要自行删去

最后贴一个所给网站的 coe 的测试

```
E:\txt_compare>txt_compare --file1 demo.txt --file2 my_result3.txt --display detailed
比较结果输出:
================================================================================
在指定检查条件下完全一致.
================================================================================
```

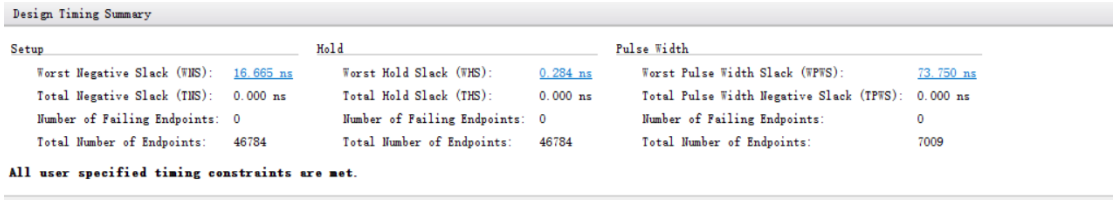
五、CPU 后仿真测试结果

报告要求：

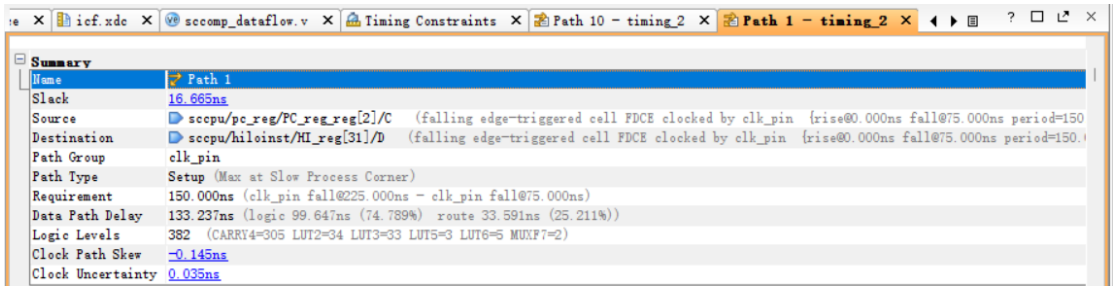
（1）对 CPU 后仿真的实验内容和结果进行描述，并给出 modelsim 后仿真波形图的贴图。
（2）对时序分析的结果进行分析，并截图。

后仿真贴图
可以清除的看到，相比于前仿真，后仿真的数据在 clk 下降沿一段时间后才出现数据，证明后仿真成功
注意：
所给的材料中已给出 icf.xdc 的约束文件，其中已经对延迟进行了约束，可以直接使用
需要先对结果进行综合以及 implement，之后再进行仿真，不能只进行综合，否则无法得到延时的波形效果

时序结果分析



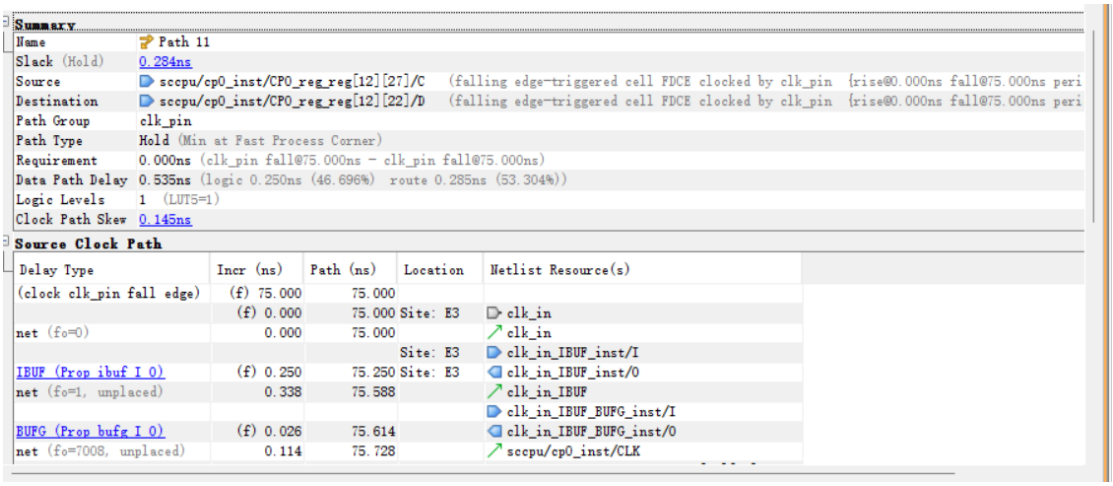
最差的建立时间余量（Slack）为 16.665ns，说明我们给出的周期十分充裕
最差的保持时间余量 为 0.284ns，说明我们给出的延时刚好，也可以在此基础上增加延时至 2ns



最大路径分析检查数据从源寄存器到目标寄存器是否能在时钟周期内完成传输（Setup 检查）。报告列出了一个关键的最大路径：
最大延迟路径分析（Max Delay Paths）
报告中最差的建立时间路径如下：
Slack：16.665 ns（满足）

源：sccpu/pc_reg/PC_reg_reg[2]/C（FDCE 触发器，时钟为 clk_pin，下降沿触发）
目标：sccpu/hiloinst/HI_reg[31]/D（FDCE 触发器，时钟为 clk_pin，下降沿触发）
路径组：clk_pin
路径类型：建立时间（Setup），慢速工艺角（Slow Process Corner）
要求时间：150.000 ns（时钟周期）
数据路径延迟：133.237 ns
逻辑延迟：99.647 ns（占 74.789%）
布线延迟：33.591 ns（占 25.211%）
逻辑级别：382（包括 CARRY4=305，LUT2=34，LUT3=33，LUT5=3，LUT6=5，MUXF7=2）
　　时钟路径偏斜：-0.145 ns
　　时钟不确定性：0.035 ns
路径分析
　　数据路径：
从程序计数器（PC_reg_reg[2]）开始，经过指令存储器（imem）、CPU 控制逻辑（cpu_ref）、除法单元（divu_inst），最终到达高位寄存器（HI_reg[31]）。

分析：由于我们使用阵列除法器作为我们的除法模块，因此电路较为复杂，延时较长



Summary				
Name	Path 11			
Slack (Hold)	0.284ns			
Source	sccpu/cp0_inst/CP0_reg_reg[12][27]/C (falling edge-triggered cell FDCE clocked by clk_pin {rise@0.000ns fall@75.000ns peri			
Destination	sccpu/cp0_inst/CP0_reg_reg[12][22]/D (falling edge-triggered cell FDCE clocked by clk_pin {rise@0.000ns fall@75.000ns peri			
Path Group	clk_pin			
Path Type	Hold (Min at Fast Process Corner)			
Requirement	0.000ns (clk_pin fall@75.000ns - clk_pin fall@75.000ns)			
Data Path Delay	0.535ns (logic 0.250ns (46.696%) route 0.285ns (53.304%))			
Logic Levels	1 (LUT5=1)			
Clock Path Skew	0.145ns			

Source Clock Path

Delay Type	Incr (ns)	Path (ns)	Location	Netlist Resource(s)
(clock clk_pin fall edge)	(f) 75.000	75.000		
	(f) 0.000	75.000	Site: E3	clk_in
net (fo=0)	0.000	75.000		clk_in
			Site: E3	clk_in_IBUF_inst/I
IBUF (Prop ibuf I O)	(f) 0.250	75.250	Site: E3	clk_in_IBUF_inst/O
net (fo=1, unplaced)	0.338	75.588		clk_in_IBUF
				clk_in_IBUF_BUFG_inst/I
BUFG (Prop bufg I O)	(f) 0.026	75.614		clk_in_IBUF_BUFG_inst/O
net (fo=7008, unplaced)	0.114	75.728		sccpu/cp0_inst/CLK

最小路径延迟（Hold Paths）

最小路径分析检查数据路径是否足够慢，以避免在时钟边沿触发时发生 Hold 违例。

最小延迟路径分析（Min Delay Paths）

Slack：0.284

4 ns（满足）

源：reset（输入端口）

目标：sccpu/cp0_inst/CP0_reg_reg[0][0-18]/CLR（清除信号）

路径组：async_default

路径类型：移除时间（Removal），慢速工艺角

要求时间：-75.000 ns

数据路径延迟：2.169 ns

逻辑延迟：1.406 ns（占 64.828%）

布线延迟：0.763 ns（占 35.172%）

逻辑级别：1（仅 IBUF）

路径分析；

这些路径是从 reset 输入到 CP0 寄存器的清除信号，逻辑简单，仅通过一个 IBUF。

时序裕量充足（74.358 ns），表明异步复位路径没有问题。

六、CPU 下板结果

报告要求：
（1）对 CPU 下板的实验内容和结果进行描述，并给出下板结果的贴图。
（2）实验的分析与结论。

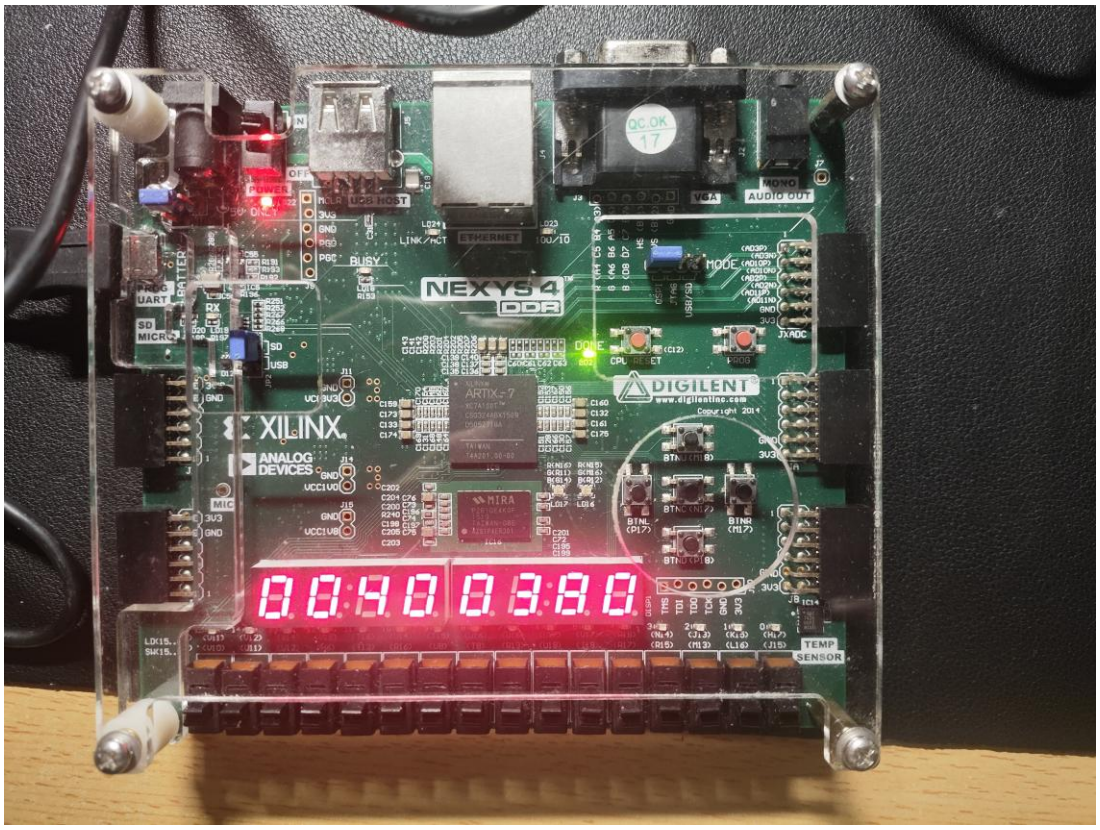
下板时需要添加两个 module

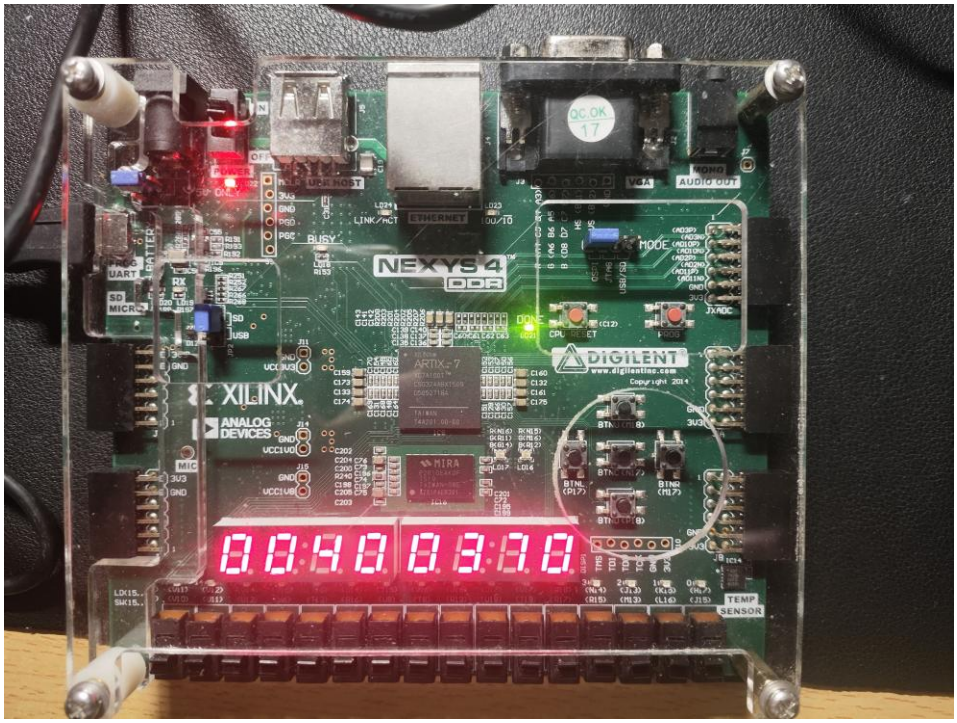
一是数码管显示模块

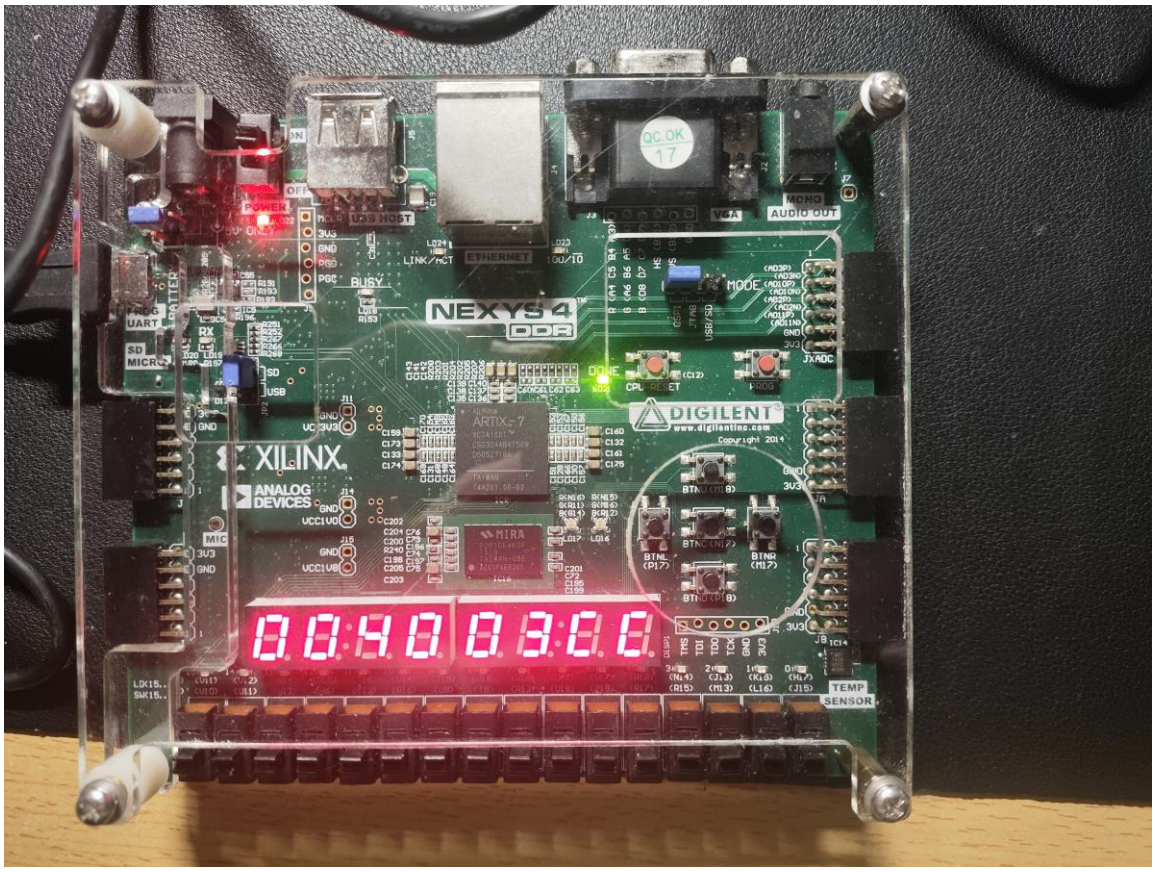
二是分频器

还要已添加一个顶层模块，包含数码管显示，分频器，以及所设计的 cpu，向外暴露接口，对应所给的 xdc 文件

下板结果如图所示


NEXYS 4
DDR
XILINX
ANALOG DEVICES
DIGILENT
ARTIX-7
MODE
VGA
AUDIO OUT
ETHERNET
USB HOST
TEMP SENSOR


NEXYS 4
DDR
XILINX
ANALOG DEVICES
DIGILENT
ARTIX-7
MODE
VGA
AUDIO OUT
ETHERNET
USB HOST
TEMP SENSOR



七、心得体会及建议

本次实验在单周期 31 指令 cpu 制作了 54 条指令单周期 CPU

在制作 cpu 的过程中，代码是最不重要的，也是应该放在最后写的

本人的认为，应该先写每一条指令的数据通路以及对应的输入输出表格，然后根据表格画出整个的数据通路图，根据数据通路图找出所有的控制信号并进行编码。最后，根据所写编码写代码。

这次实验让我更加深刻的认识到了 verilog 语言作为描述型语言与以往 c++语言的不同

54 条指令单周期 CPU 的制作与 31 条指令单周期 CPU 的制作相似，重点难点在于 CP0 中断，DIV，MULT 的运算实现。

建议提高 mips 网站的提交数量，其次是针对 mars 文件结果与程序输出相差一个周期的问题应当强调一下，以防止同学做很多无用功。