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 31 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

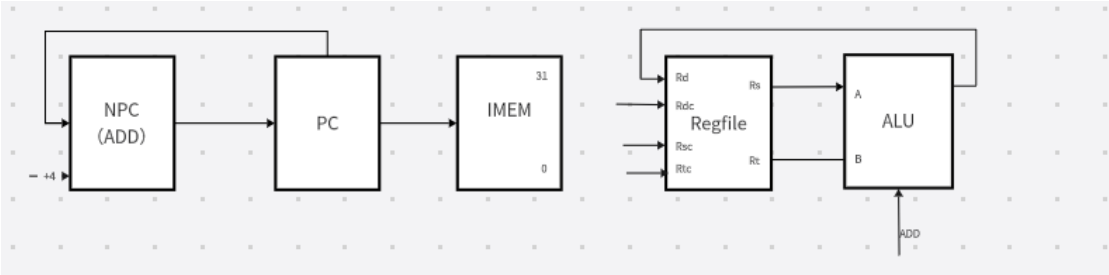
R type：

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）

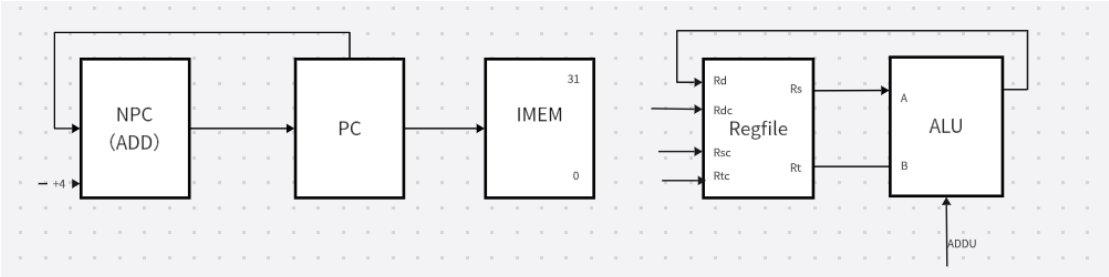


2.addu

rd ← rs + r

功能：类似 ADD，但不检查溢出（R 型）。

格式：ADDU rd, rs, rt

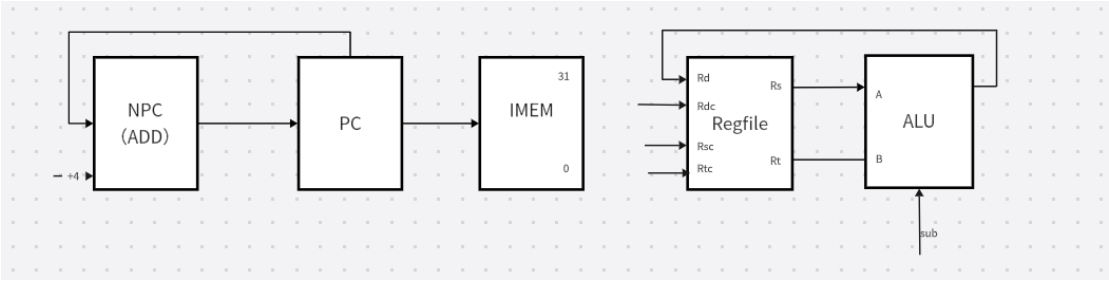


3.sub

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）

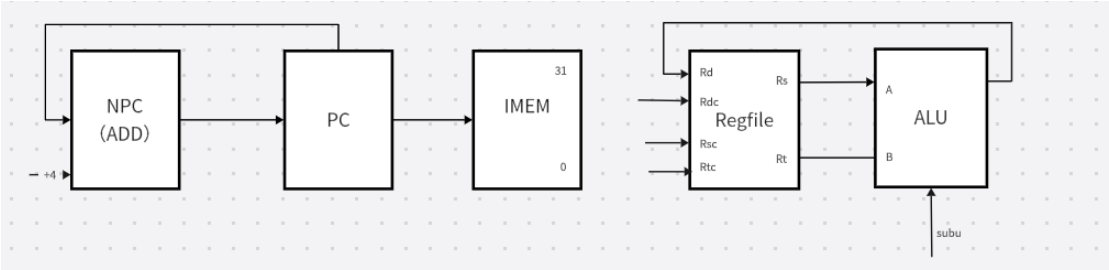


4.subu

rd ← rs - rt

功能：类似 SUB，但不检查溢出（R 型）。

格式：SUBU rd, rs, rt

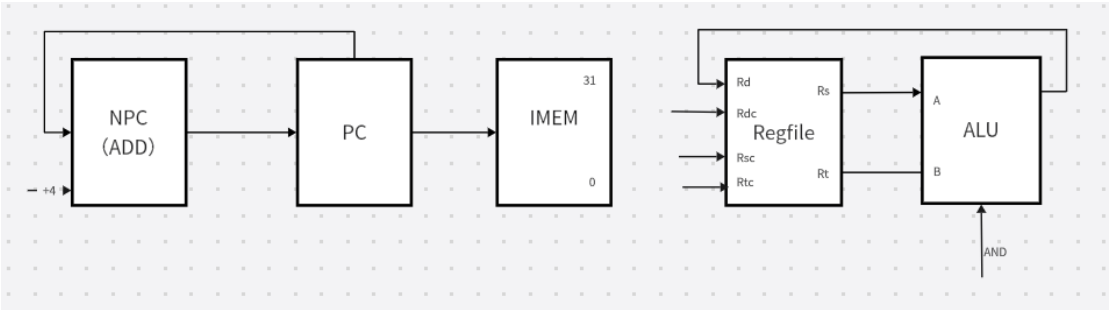


5. AND

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）

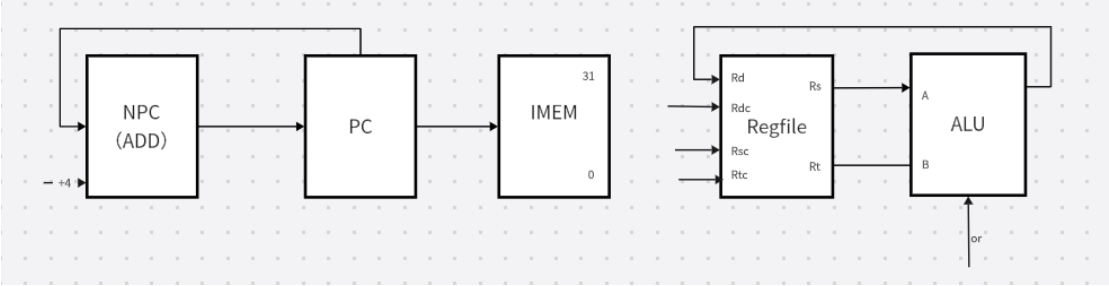


6. or

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）

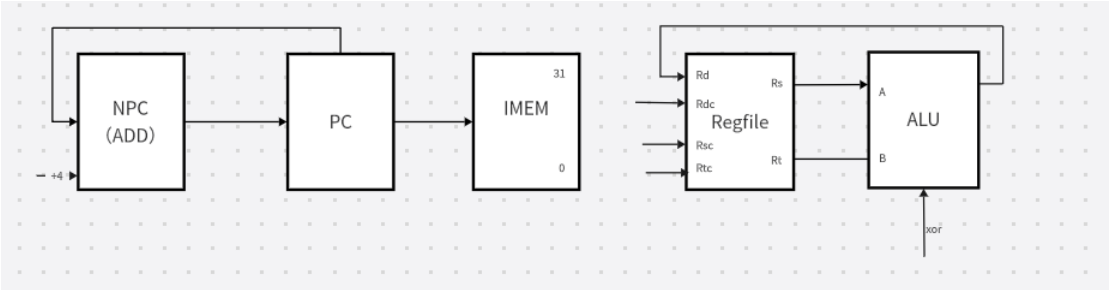


7.xor

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）

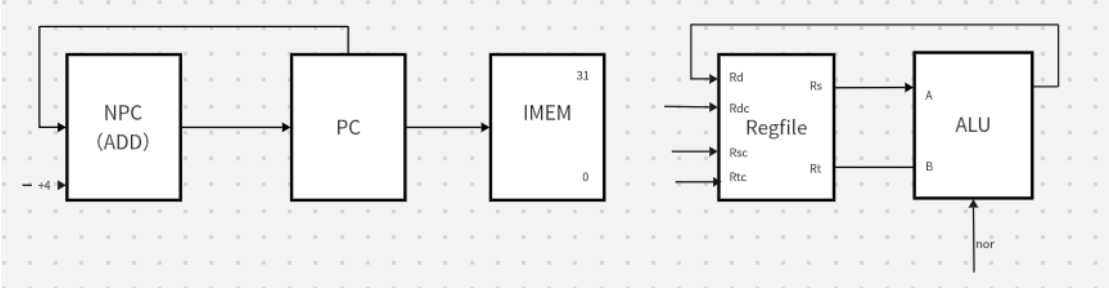


8.nor

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

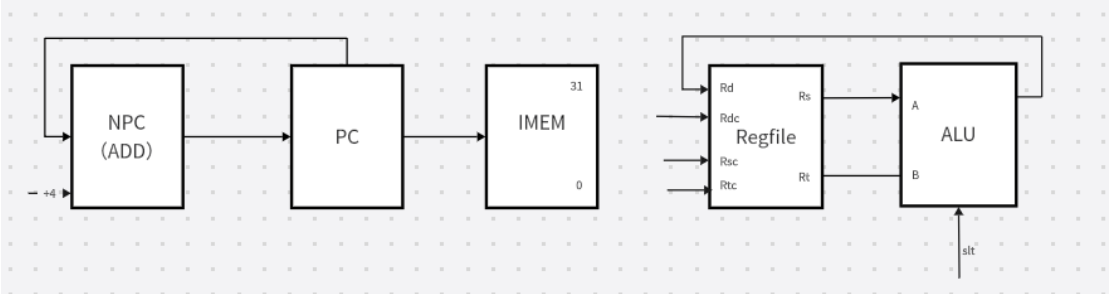
PC
↓
IMEM (读取指令)
↓
NPC (PC + 4 -> NPC)
↓
PC (NPC -> PC)
↓
寄存器文件
Rs -> A
Rt -> B
↓
ALU (执行运算)
↓
RES (结果 -> Rd)

9.slt

rd ←(rs < rt)

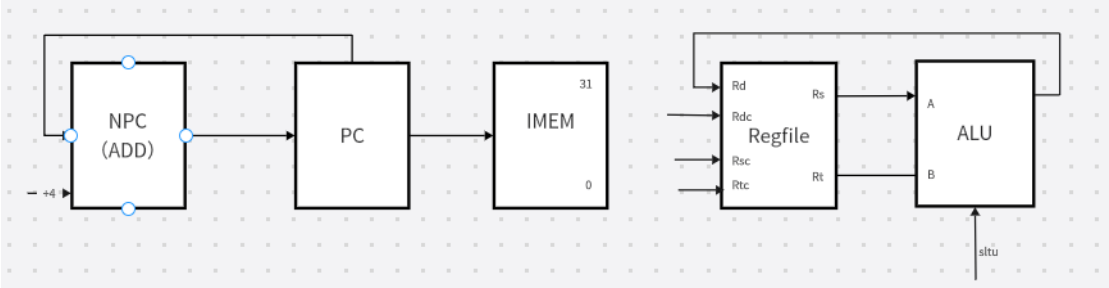
功能：比较两个寄存器值，若 rs < rt，则目标寄存器置 1，否则置 0（R 型）。

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



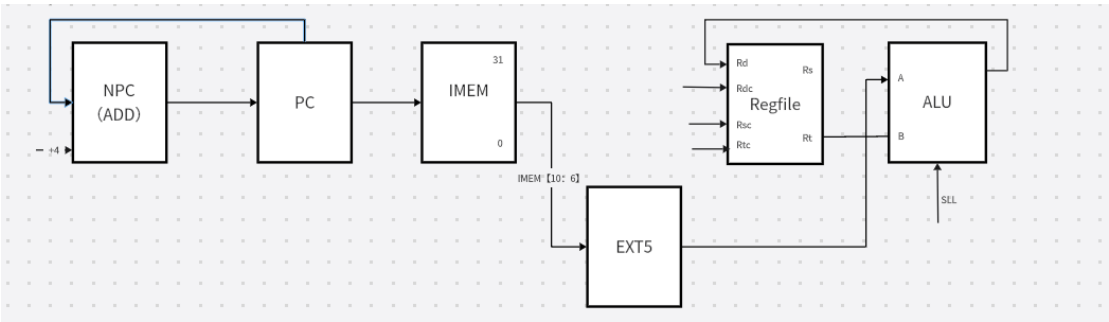
指令
PC -> IMEM
PC + 4 -> NPC
NPC -> PC
Rs -> A
Rt -> B
A - B -> RES #这里其实做的是减法操作
SF ->Rd

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

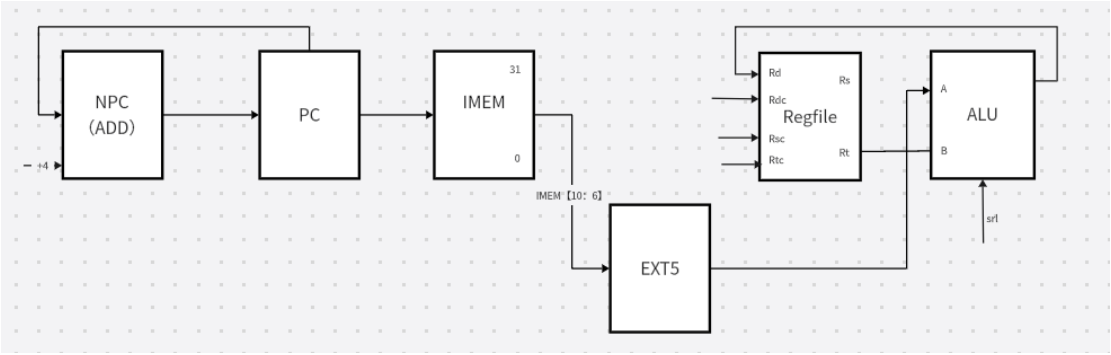
Res -> Rd

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

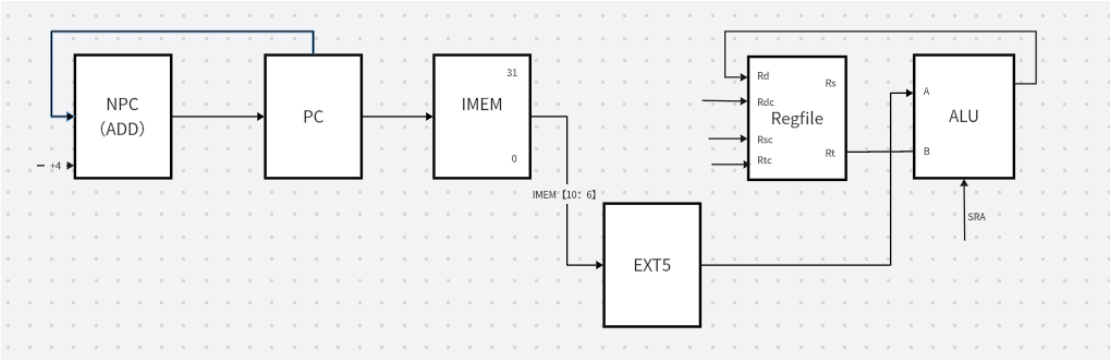
Res -> Rd

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）。

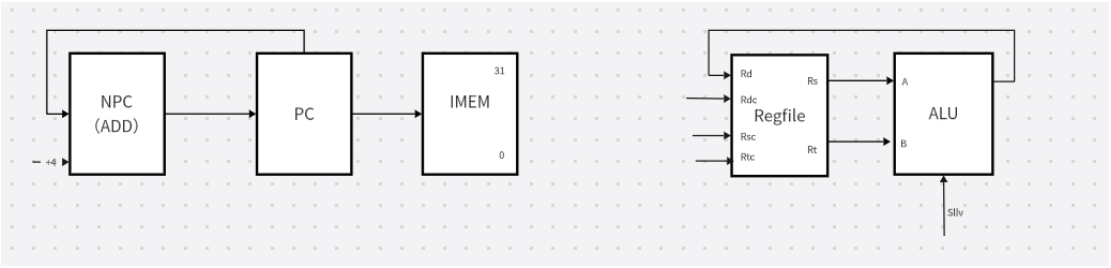
格式：SRA rd, rt, shamt （rd = rt >> shamt）



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

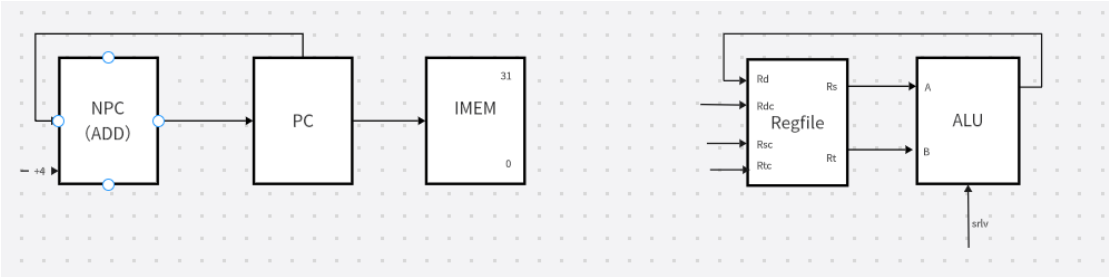
14.sllv
rd ← rt << rs
功能：根据另一寄存器值指定位数逻辑左移，结果存入目标寄存器（R 型）。
格式：SLLV rd, rt, rs （rd = rt << rs）



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

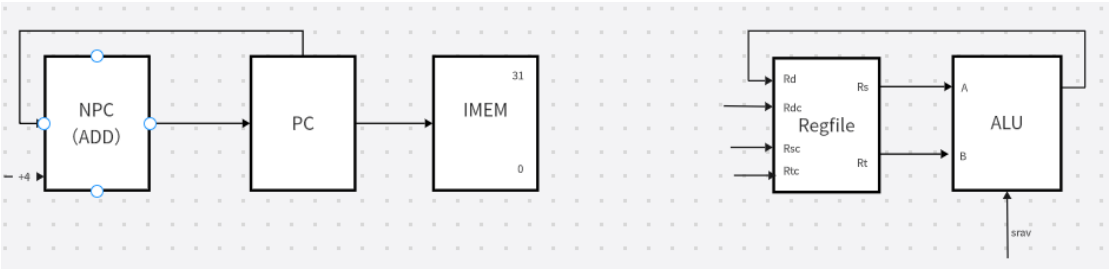
15.srlv

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）

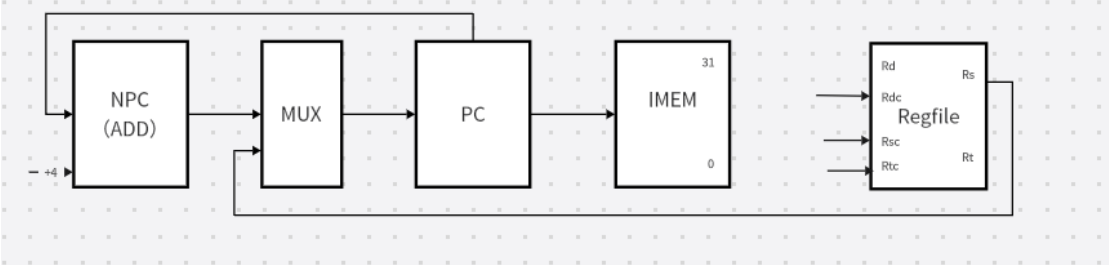


指令流程图

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。

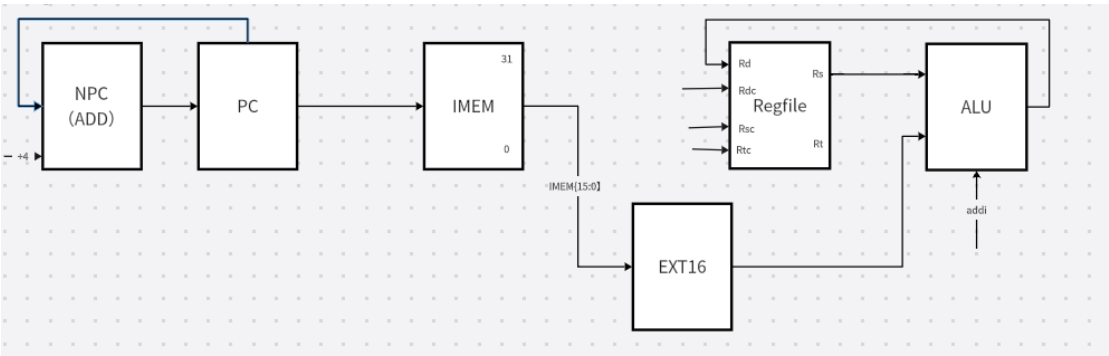


指令流程图

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

I type:

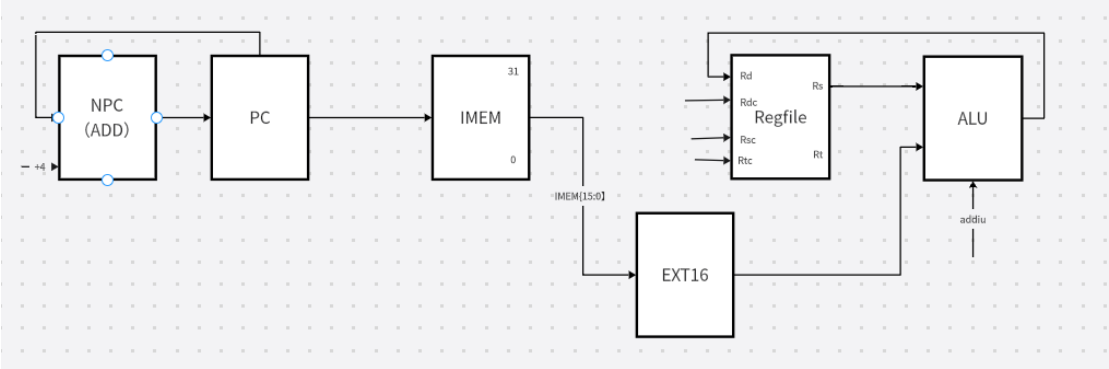
18.addi
rt ← rs + immediate
功能：将寄存器值与立即数相加，结果存入目标寄存器（I 型）。
格式：ADDI rt, rs, imm （rt = rs + imm）



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

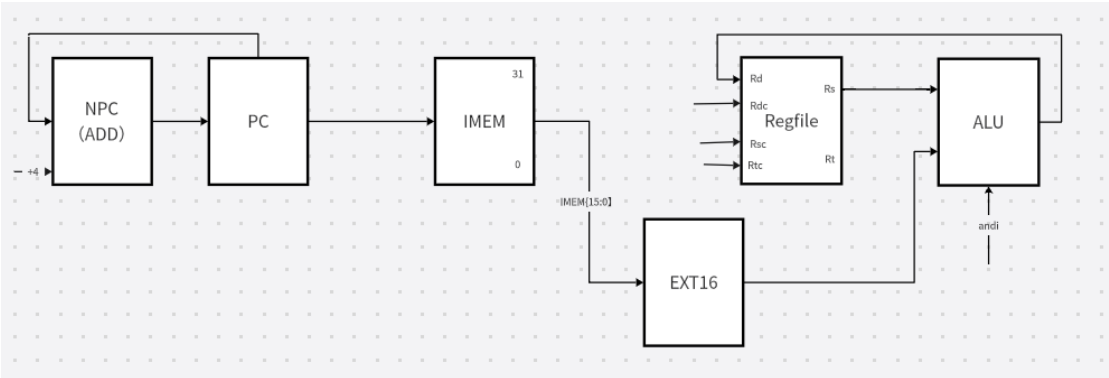
19.addiu
rt ← rs + immediate
功能：类似 ADDI，但不检查溢出（I 型）。
格式：ADDIU rt, rs, imm



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）

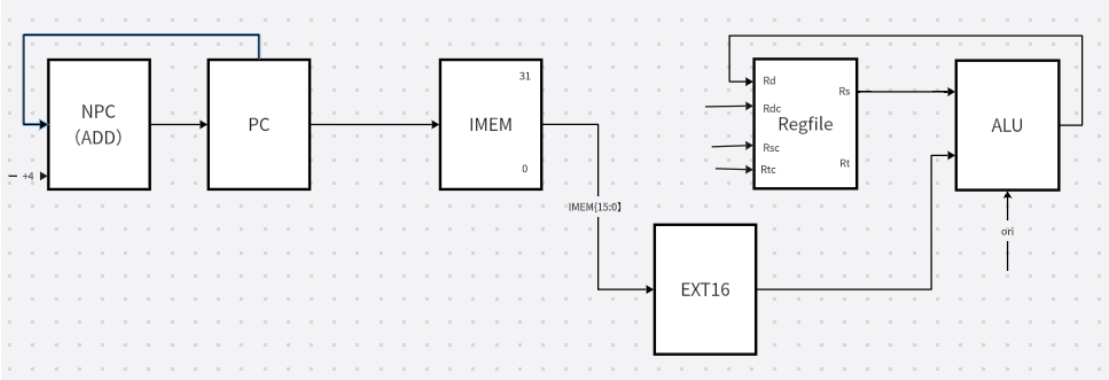


指令流程图

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

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

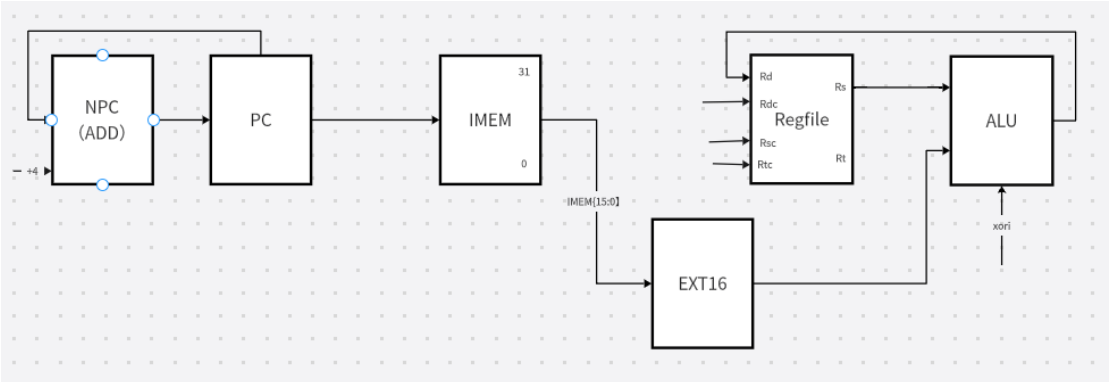
21.ori
rt ← rs or immediate
功能：寄存器值与立即数按位或，结果存入目标寄存器（I 型）。
格式：ORI rt, rs, imm （rt = rs | imm）



指令流程图

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

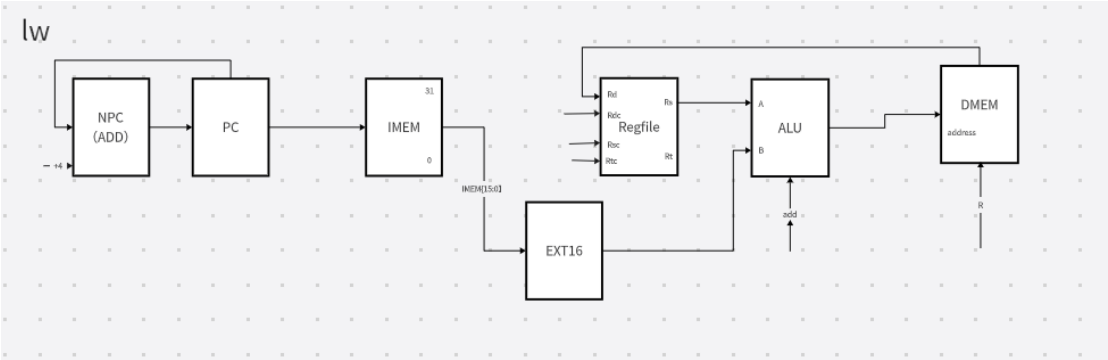
22.xori
rt ←rs XOR immediate
功能：对两个寄存器值按位异或，结果存入目标寄存器（R 型）。
格式：XOR rd, rs, rt （rd = rs ^ rt）



指令流程图

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

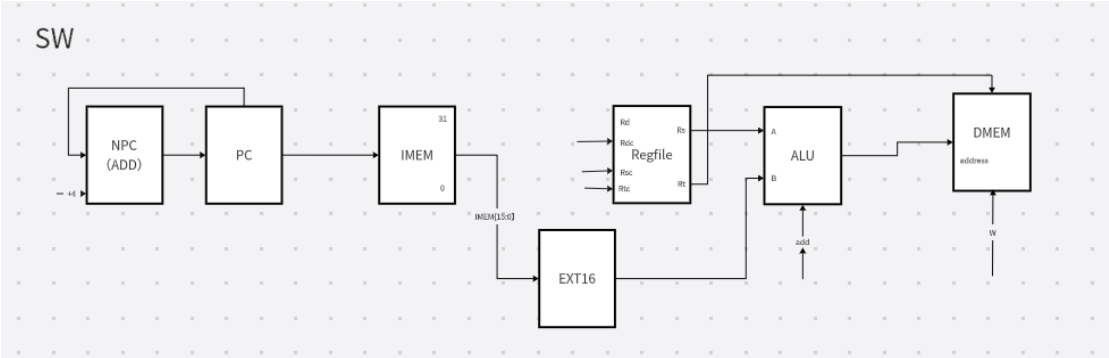
23.lw
rt ← memory[base+offset]
功能：从内存加载 32 位字到寄存器（I 型）。
格式：LW rt, offset(rs) （rt = memory[rs + offset]）



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

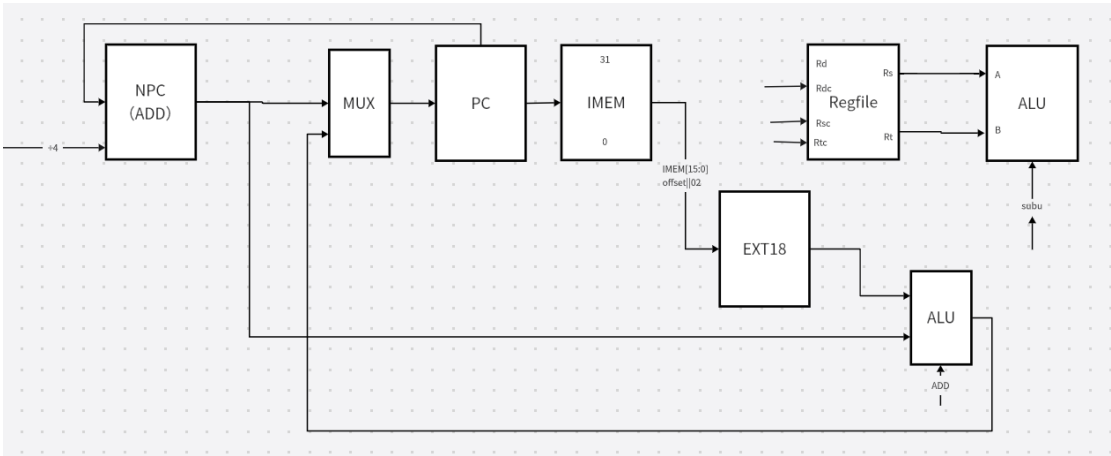
24.sw
memory[base+offset] ←r
功能：将寄存器中的 32 位字存储到内存（I 型）。
格式：SW rt, offset(rs) （memory[rs + offset] = rt



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）

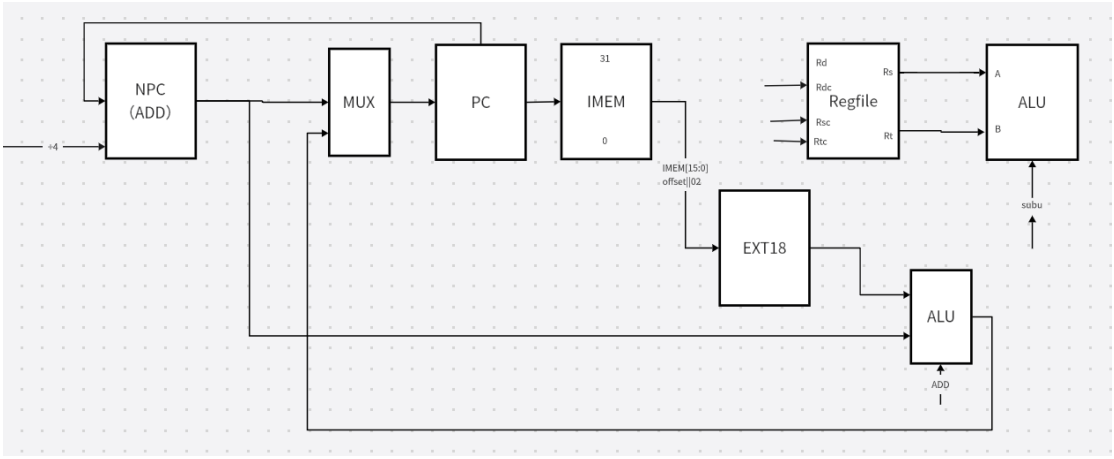


指令流程图

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

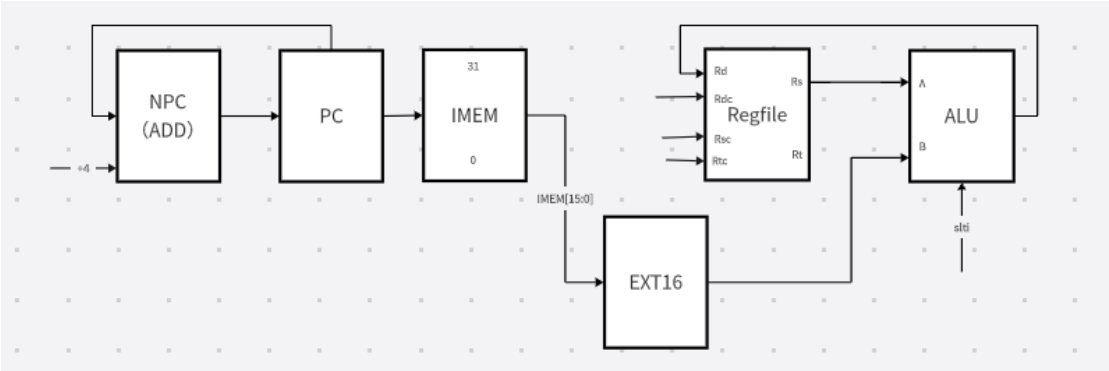
26.bne
功能：如果两个寄存器值不相等，则跳转到目标地址（I 型）。
格式：BNE rs, rt, offset



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

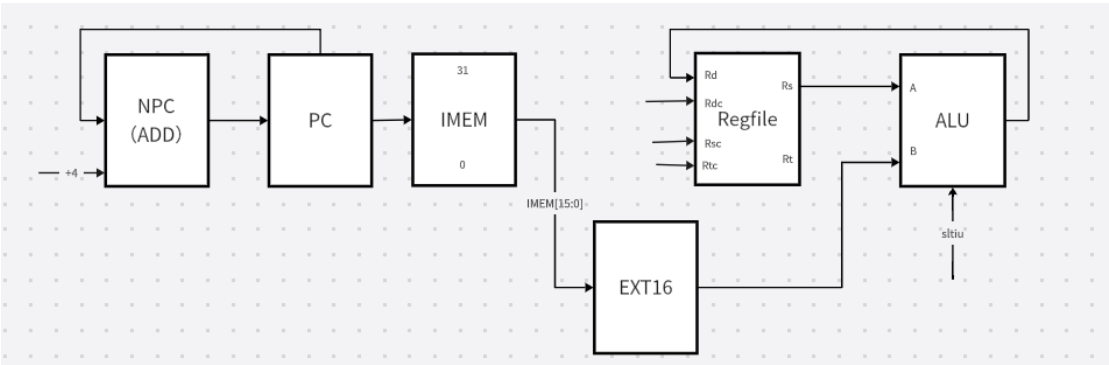
27.slti
rt ←(rs < immediate)
功能：寄存器值与立即数比较，若 rs < imm，则目标寄存器置 1，否则置 0（I 型）。
格式：SLTI rt, rs, imm （rt = (rs < imm) ? 1 : 0）



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

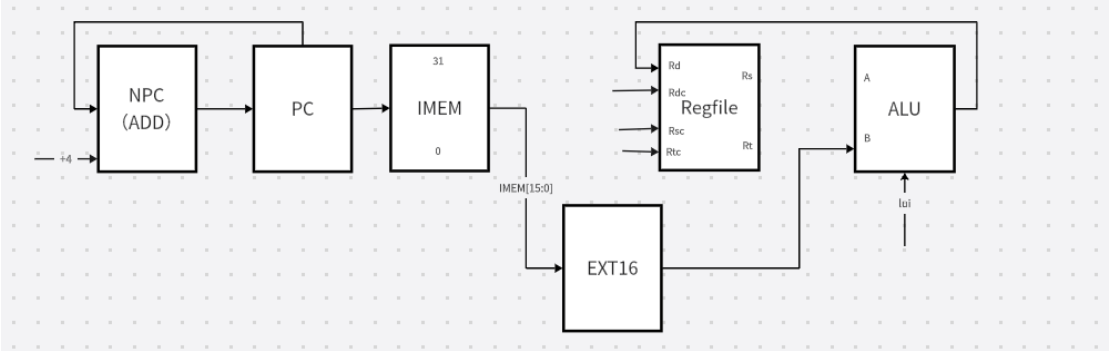
28.sltiu
rt ←(rs < immediate)
功能：类似 SLTI，但进行无符号比较（I 型）。
格式：SLTIU rt, rs, imm



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

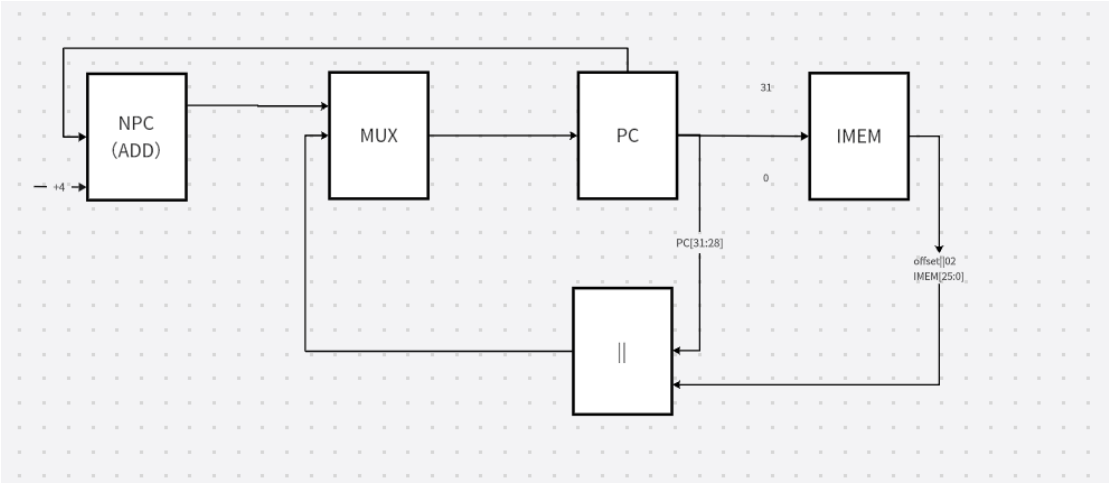
29.lui
rt ← immediate || 016
功能：将 16 位立即数加载到寄存器的高 16 位，低 16 位清零（I 型）。
格式：LUI rt, imm



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）

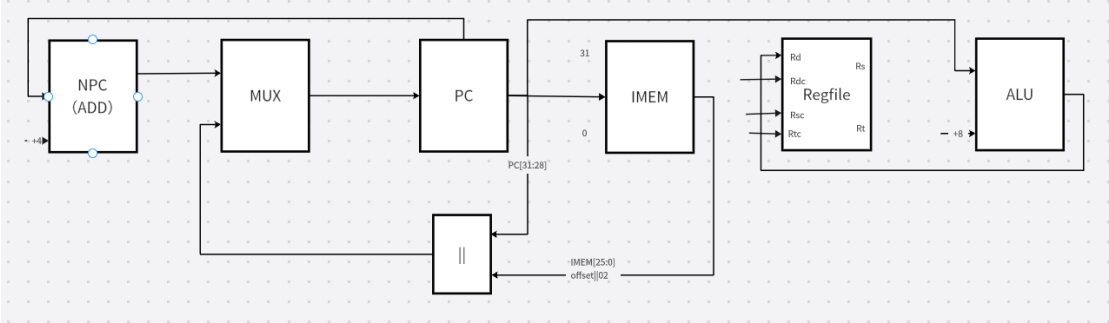


指令流程图

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

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

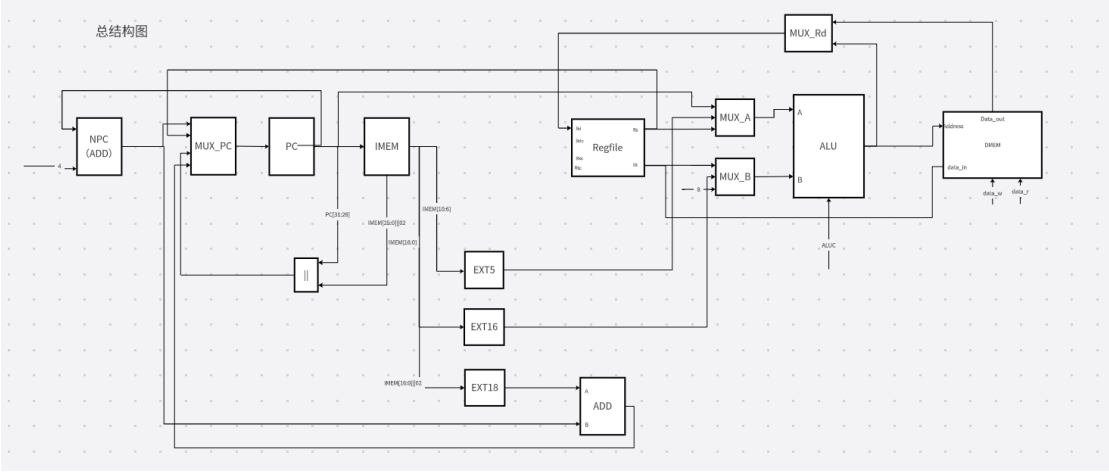
31.jal
功能：无条件跳转到目标地址，并保存返回地址（J 型）。
格式：JAL target
target：26 位目标地址（左移 2 位后与 PC 高 4 位组合）。



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

（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	
					rd	rdc	A	B				Addr	Data	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	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	slti	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							

控制信号表

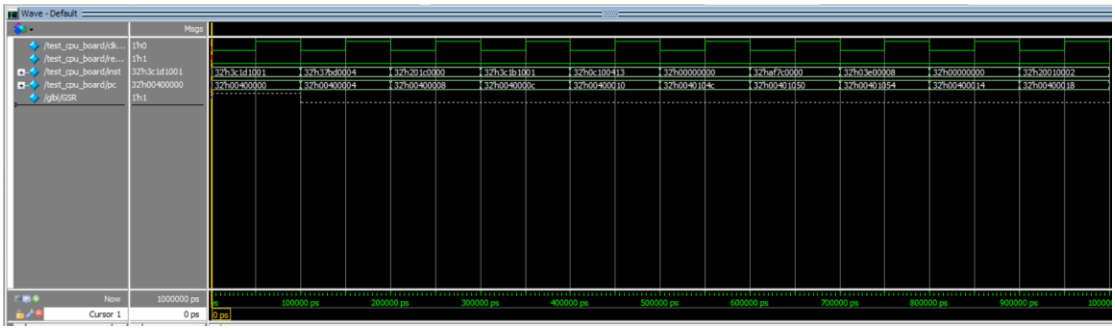
		IMEM_r	data_w	data_r	ALUC	reg_w	MUX_PC	MUX_Rd	MUX_A	MUX_B
1	add	1	0	0	00000	1	00	0	00	00
2	addu	1	0	0	00001	1	00	0	00	00
3	sub	1	0	0	00010	1	00	0	00	00
4	subu	1	0	0	00011	1	00	0	00	00
5	and	1	0	0	00100	1	00	0	00	00
6	or	1	0	0	00101	1	00	0	00	00
7	xor	1	0	0	00110	1	00	0	00	00
8	nor	1	0	0	00111	1	00	0	00	00
9	slt	1	0	0	01000	1	00	0	00	00
10	sltu	1	0	0	01001	1	00	0	00	00
11	sll	1	0	0	01010	1	00	0	10	00
12	srl	1	0	0	01011	1	00	0	10	00
13	sra	1	0	0	01100	1	00	0	10	00
14	sllv	1	0	0	01101	1	00	0	00	00
15	srlv	1	0	0	01110	1	00	0	00	00
16	srav	1	0	0	01111	1	00	0	00	00
17	jr	1	0	0			01			
18	addi	1	0	0	00000	1	00	0	00	01
19	addiu	1	0	0	00001	1	00	0	00	01
20	andi	1	0	0	00100	1	00	0	00	01
21	ori	1	0	0	00101	1	00	0	00	01
22	xori	1	0	0	00110	1	00	0	00	01
23	lw	1	0	1	00000	1	00	1	00	01
24	sw	1	1	0	00000	0	00		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	00	0	00	01
28	sltiu	1	0	0	01001	1	00	0	00	01
29	lui	1	0	0	10000	1	00	1		01
30	j	1	0	0		0	10			
31	jal	1	0	0	00000	1	10	00	01	10

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.
```

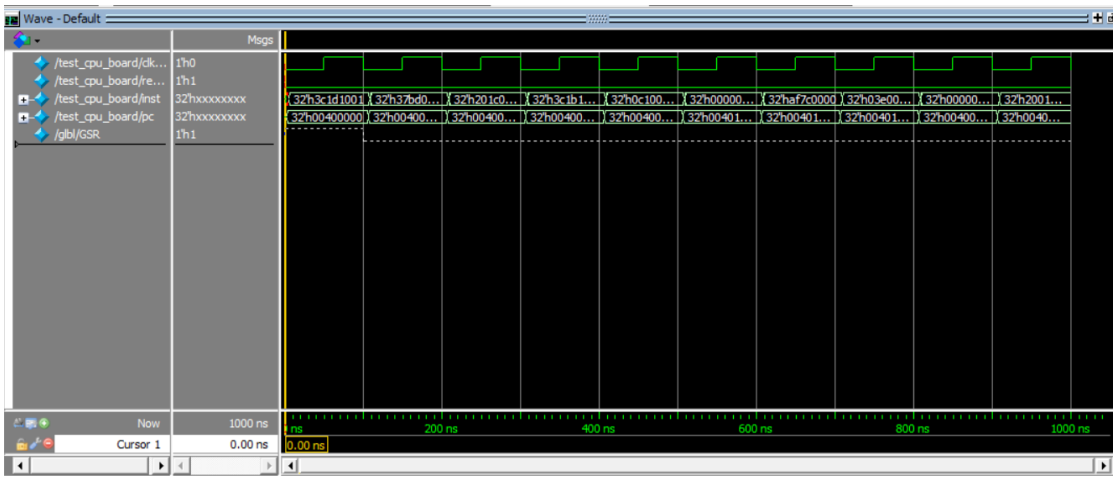
需要自行删去

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

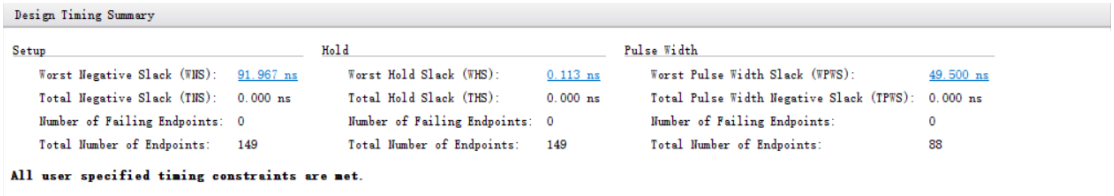
报告要求：
（1）对 CPU 后仿真的实验内容和结果进行描述，并给出 modelsim 后仿真波形图的贴图。
（2）对时序分析的结果进行分析，并截图。



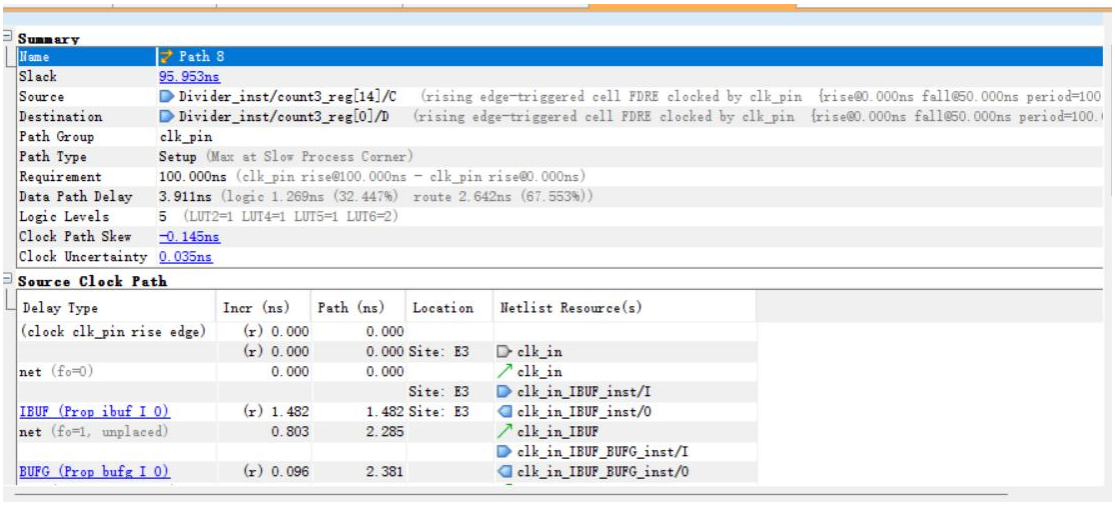
后仿真贴图
可以清除的看到，相比于前仿真，后仿真的数据在 clk 下降沿一段时间后才出现数据，证明后仿真成功
注意：

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



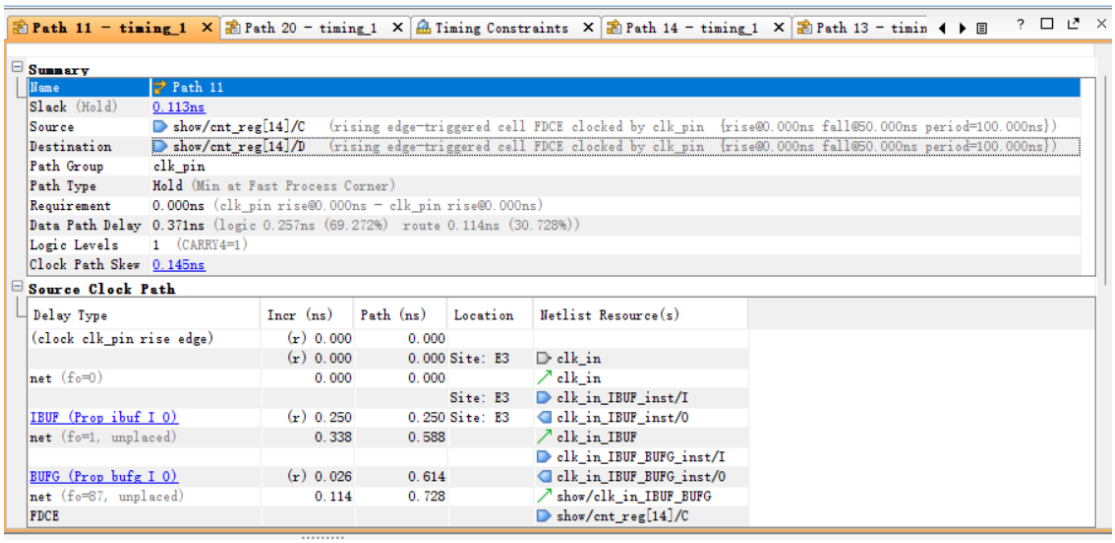
最差的建立时间余量（Slack）为 91.967ns，说明我们给出的周期十分充裕
最差的保持时间余量 为 0.113ns，说明我们给出的延时刚好，也可以在此基础上增加延时至 2ns



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：

关键路径详情

- **源**：my_sccomp_dataflow/sccpu/pc_reg/PC_reg_reg[8]/C（程序计数器寄存器）
- **目标**：show/i_data_store_reg[14]/D（显示模块的寄存器）
- **路径组**：无（未约束路径）
- **路径类型**：Setup（最大延迟，慢速工艺角）
- **数据路径延迟**：4.184ns
  - **逻辑延迟**：1.889ns（45.141%，包括 FDCE 和多个 LUT6）
  - **路由延迟**：2.295ns（54.859%）
- **逻辑级数**：7（1 个 FDCE + 6 个 LUT6）
- **时钟不确定性**：0.035ns（由系统抖动等因素引起）
- **时钟路径**：
  - 从输入引脚 clk_in 通过 IBUF 和 BUFG，延迟为 2.704ns。
- **Slack**：无限（inf），因为路径未受约束。



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。报告列出了多个最小路径，以下是代表性路径的分析：

关键路径详情（示例）

- **源**：my_sccomp_dataflow/sccpu/pc_reg/PC_reg_reg[10]/C
- **目标**：show/i_data_store_reg[6]/D
- **路径组**：无（未约束路径）
- **路径类型**：Hold（最小延迟，快速工艺角）
- **数据路径延迟**：0.562ns
  - **逻辑延迟**：0.250ns（44.490%，包括 FDCE 和 LUT5）
  - **路由延迟**：0.312ns（55.510%）
- **逻辑级数**：2（1 个 FDCE + 1 个 LUT5）
- **时钟不确定性**：0.025ns
- **时钟路径**：
  - 从 clk_in 通过 IBUF 和 BUFG，延迟为 1.082ns。
- **Slack**：无限（inf），因为路径未受约束。

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。

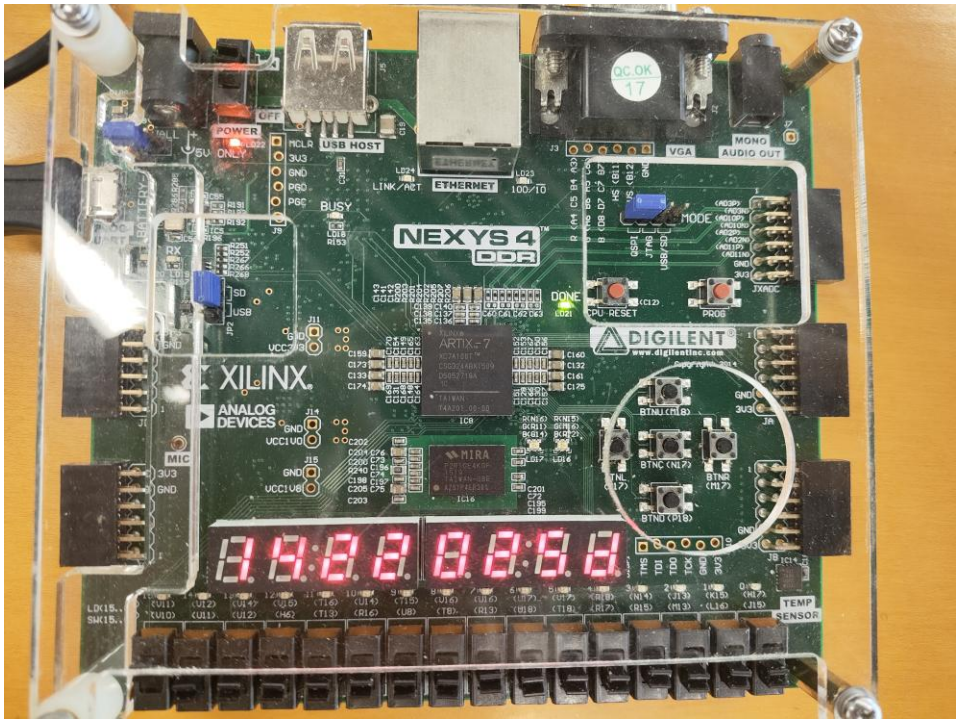
下板时需要添加两个 module

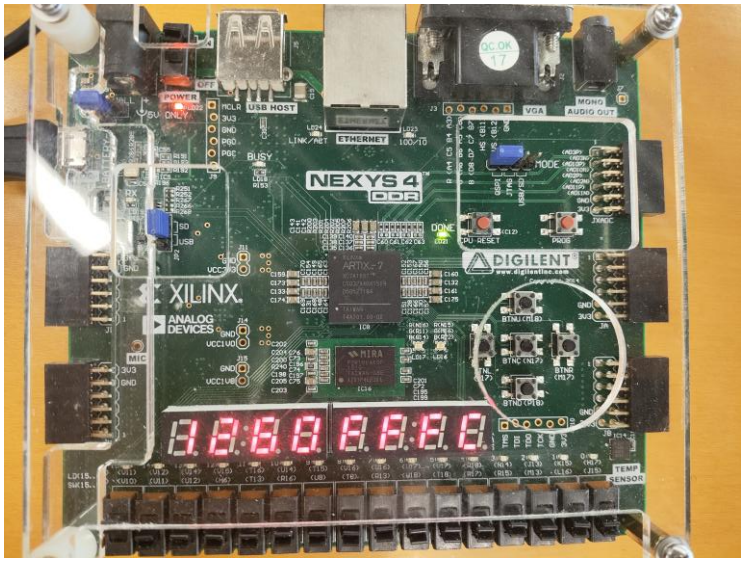
一是数码管显示模块

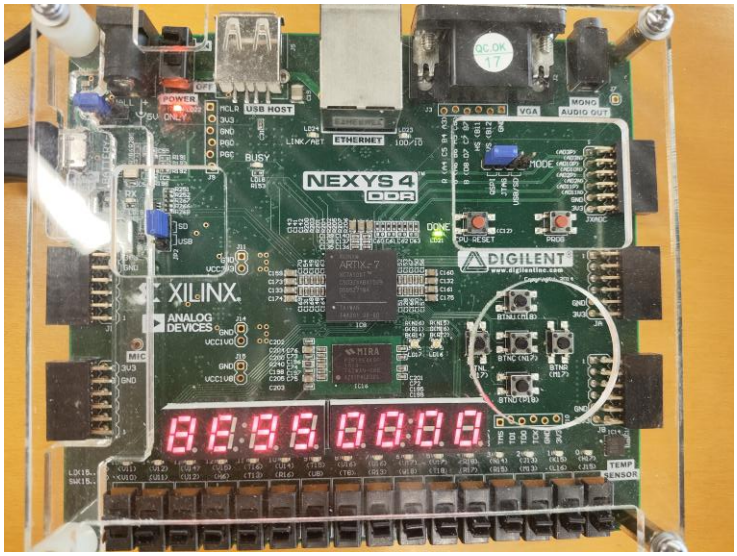
二是分频器

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







如图 n17 为 reset，按下后，数码管显示为 0，然后按照分频器设置的时钟频率进行取址，数码管上显示的是取到的指令的 16 进制编码

七、心得体会及建议

本次实验从 0 开始制作了一个单周期 31 指令 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

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。