2020

和 P3 差不多，仔细看 RTL。总的来说感觉最好课下好好练练 verilog，课上写得时候感觉有点手生，或许是因为不如电路那么直观。但是在写复杂逻辑的时候不用连电路了确实很好。

- 第一题：`bsoal`

  感觉这个指令是全场最难（溜）。需要熟悉组合电路的 `always @(*)` 写法，或者直接用 `function`。

  当寄存器值有奇数个 11 的时候跳转。

$$
\begin{aligned}
&\text{if}\ \ \text{has\_odd\_one\_bits}(GRF[rs])\ \ \text{then}\\
&\qquad PC = PC + 4 + \text{sign\_ext}(offset||0^2)\\
&\qquad GRF[31] = PC + 4\\
&\text{else}\\
&\qquad PC = PC + 4
\end{aligned}
$$

- 第二题：`xor`

  就是需要加一个高老板 PDF 的 `xor`，居然出了原题，这应该是全场最简单的了。

- 第三题：`swrr`

  向右循环位移，有点类似 `sw`。注意好好看题！题目要求 `$display()` 要字对齐。

$$
\begin{aligned}
&Addr \leftarrow GPR[base] + \texttt{sign\_ext}(offset)\\
&temp \leftarrow Addr_{1..0}\\
&\text{if}\ \ temp == 0\ \ \text{then}\\
&\qquad mem_{addr} \leftarrow GRF[rt]\\
&\text{else}\\
&\qquad mem_{addr} \leftarrow GRF[rt]_{8*temp-1\cdots 0}||GRF[rt]_{31\cdots 8*temp}
\end{aligned}
$$

课下一定要模块化，课上不要慌！（我课上因为失误，第三题拖了好长时间）

最重要的是拿到指令先去和写过的找共同点，分析要用哪些部件。然后按照套路添加即可。

2021

P4_L1_rlb

编码：

31 26	25 21	20 16	15 0
rlb 101010	rs	rt	imm
6	5	5	16

格式：

```
rlb rs, rt, imm
```

描述：

```
指定位数取反
```

操作：

```
temp <- GPR[rs]imm-1...0
GPR[rt] <- ( GPR[rs]31...imm || ~temp )
```

实例：

```
rlb $t1, $t2, 5
```

其他：

```
保证 imm 的范围在 0 ~ 31 之间
```

注意：这个下标是变量不能直接写，需要用for循环

P4_L2_bnezalc

编码：

31 26	25 21	20 16	15 0
regimm 000001	rs	bnezalc 11001	offset
6	5	5	16

格式：

```
bnezalc rs, offset
```

描述：

```
若 GPR[rs] != 0 跳转并 Link
```

操作：

```
I:
    target_offset <- sign_extend(offset||00)
    condition <- GPR[rs] != 0
    If Condition:
        GPR[31] <- PC + 8
    endif
I+1:
    If Condition:
        PC ← PC + target_offset
    endif
```

实例：

```
bnezalc $t1, -1
```

其他：

```
不跳转则不Link，视为nop
```

P4_L3_lwrr

编码：

31 26	25 21	20 16	15 0
lwrr 000011	base	rt	offset
6	5	5	16

格式：

```
lwrr rt, offset(base)
```

描述：

```
根据内存后两位，循环右移后存入对应寄存器中
```

操作：

```
addr <- GPR[base] + sign_extend(offset)
Paddr <- Memory[addr]
temp <- Paddr1...0
if temp != 0:
    Vaddr <- ( Paddr(temp-1)...0 || Paddr31...temp )
else:
    Vaddr <- Paddr
GPR[rt] <- Vaddr
```

实例：

```
lwrr $t5, 1($0)
```

其他：

```
无
```

2022

第一次

组合逻辑别用非阻塞赋值，会死的。。

```
always @(*) begin
    if(reset)
        cnt = 32'b0;
    else begin
        cnt = 32'b0; // 这个很重要
        for(i=0;i<32;i=i+1)begin
            cnt = cnt + RD2[i];
        end
    end
end
```

在组合逻辑中使用寄存器记得每次使用前要清零 不然会导致错误。

算术右移

```
$signed($signed(A)>>>B);
```

没来得及看的第一道题

ras

if（imm == 0）GPR[rt] <= GPR[rs]
else if (imm < 32)
temp = ~GPR[rs][31:32-imm] || GPR[rs][31-imm:0]
num = how many 1 in imm
GPR[rt] = temp >>> num
else
temp = ~GPR[rs]
GPR[rt] = temp >>> num

```
if(Imm<= 16'd32) begin
    for(i=31;i>=0&&i>=(32-Imm);i=i-1)begin
        A[i] = ~SrcB[i];
    end
    for(i=(32-Imm-1);i>=0;i=i-1)begin
        A[i] = SrcB[i];
    end
end
else begin
    A = ~SrcB;
end
```

要给Imm判断大小，貌似不能判断 i >= 一个负数的情况

写while一定要用 i++ 千万不能忘了

L1

GPR[rs]中如果1的个数不为零且被0的个数整除则bal

```
always @(*) begin
    integer cnt1 = 0;
    integer cnt0;
    integer i = 0;
    for (i = 0; i < 32; i = i + 1) begin
        if (SrcA[i] == 1'b1) cnt1 = cnt1 + 1;
    end
    cnt0 = 32 - cnt1;
    if (cnt1 % cnt0 == 0) xxxx = 1;
end
```

L2

GPR[rt]和GPR[rs]如果后缀0个数相同则GPR[rd]置1 否则GPR[rd]置0

```
always @(*) begin
    cnt1 = 0;
    cnt2 = 0;
    for (i = 0; i < 32 && SrcA[i] == 1'b0; i = i + 1) begin
        cnt1 = cnt1 + 1;
    end
    for (i = 0; i < 32 && SrcB[i] == 1'b0; i = i + 1) begin
        cnt2 = cnt2 + 1;
    end
    if (cnt1 == cnt2) Result = 32'b1;
    else Result = 32'b0;
end
// RegDst = 1;
```