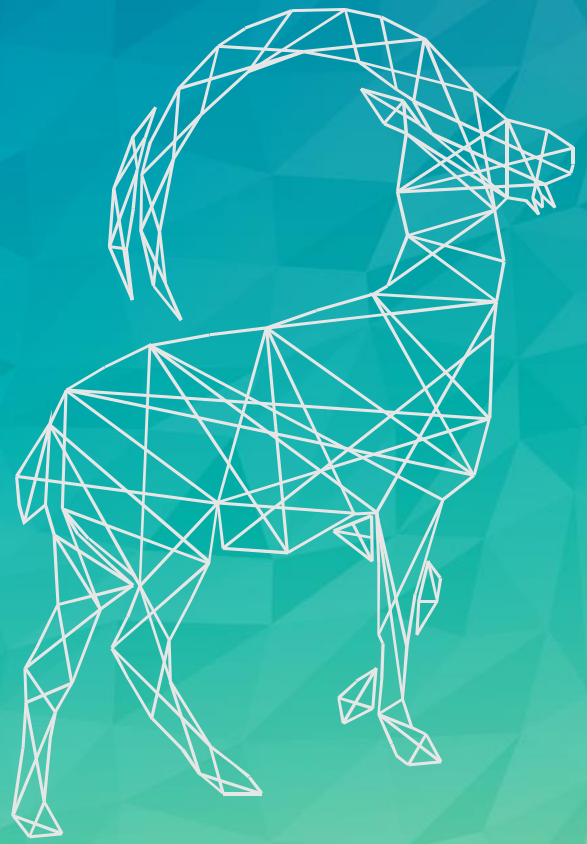


相对简单CPU

物联1601 201608010424 潘婷婷

PS：因为课堂时间不够，没有能够在课堂上进行汇报，我把自己想表达的大致内容用**蓝色**字体写在PPT上了。

定义端口

地址引脚A[15..0]
数据引脚D[7..0]

```
entity rscpu is
    port(
            clk: in std_logic;
            reset: in std_logic;
            addrbus: out std_logic_vector(15 downto 0);
            databus: inout std_logic_vector(7 downto 0);
            readl: out std_logic;
            writel: out std_logic
        );
end entity;
```

寄存器

- 8位累加器AC
- 寄存器R
- 零标志位Z
- 16位的地址寄存器AR
- 16位的程序计数器PC
- 8位的数据寄存器DR
- 8位的指令寄存器IR
- 8位的临时寄存器TR

```
signal pc: std_logic_vector(15 downto 0);
signal ac: std_logic_vector(7 downto 0);
signal r: std_logic_vector(7 downto 0);
signal ar: std_logic_vector(15 downto 0);
signal ir: std_logic_vector(7 downto 0);
signal dr: std_logic_vector(7 downto 0);
signal tr: std_logic_vector(7 downto 0);
signal z: std_logic;
```

状态操作码

```
constant fetch1: std_logic_vector(5 downto 0) := "000000";
constant fetch2: std_logic_vector(5 downto 0) := "000001";
constant fetch3: std_logic_vector(5 downto 0) := "000010";
constant fetch4: std_logic_vector(5 downto 0) := "000011";

constant nop1: std_logic_vector(5 downto 0)  := "000100";

constant ldac1: std_logic_vector(5 downto 0) := "000101";
constant ldac2: std_logic_vector(5 downto 0) := "000110";
constant ldac3: std_logic_vector(5 downto 0) := "000111";
constant ldac4: std_logic_vector(5 downto 0) := "001000";
constant ldac5: std_logic_vector(5 downto 0) := "001001";

constant stac1: std_logic_vector(5 downto 0) := "001010";
constant stac2: std_logic_vector(5 downto 0) := "001011";
constant stac3: std_logic_vector(5 downto 0) := "001100";
constant stac4: std_logic_vector(5 downto 0) := "001101";
constant stac5: std_logic_vector(5 downto 0) := "001110";

constant mvac1: std_logic_vector(5 downto 0) := "001111";

constant movr1: std_logic_vector(5 downto 0) := "010000";
```

```
constant jump1: std_logic_vector(5 downto 0)  := "010001";
constant jump2: std_logic_vector(5 downto 0)  := "010010";
constant jump3: std_logic_vector(5 downto 0)  := "010011";

constant jmpzn1: std_logic_vector(5 downto 0) := "010100";
constant jmpzn2: std_logic_vector(5 downto 0) := "010101";

constant jpnzn1: std_logic_vector(5 downto 0) := "010110";
constant jpnzn2: std_logic_vector(5 downto 0) := "010111";

constant jmpzy1: std_logic_vector(5 downto 0) := "011000";
constant jmpzy2: std_logic_vector(5 downto 0) := "011001";
constant jmpzy3: std_logic_vector(5 downto 0) := "011010";

constant jpnzy1: std_logic_vector(5 downto 0) := "011011";
constant jpnzy2: std_logic_vector(5 downto 0) := "011100";
constant jpnzy3: std_logic_vector(5 downto 0) := "011101";

constant add1: std_logic_vector(5 downto 0)   := "011110";
constant sub1: std_logic_vector(5 downto 0)   := "011111";
constant inac1: std_logic_vector(5 downto 0)  := "100000";
constant clac1: std_logic_vector(5 downto 0)  := "100001";
constant and1: std_logic_vector(5 downto 0)   := "100010";
constant or1: std_logic_vector(5 downto 0)    := "100011";
constant xor1: std_logic_vector(5 downto 0)   := "100100";
constant not1: std_logic_vector(5 downto 0)   := "100101";
```

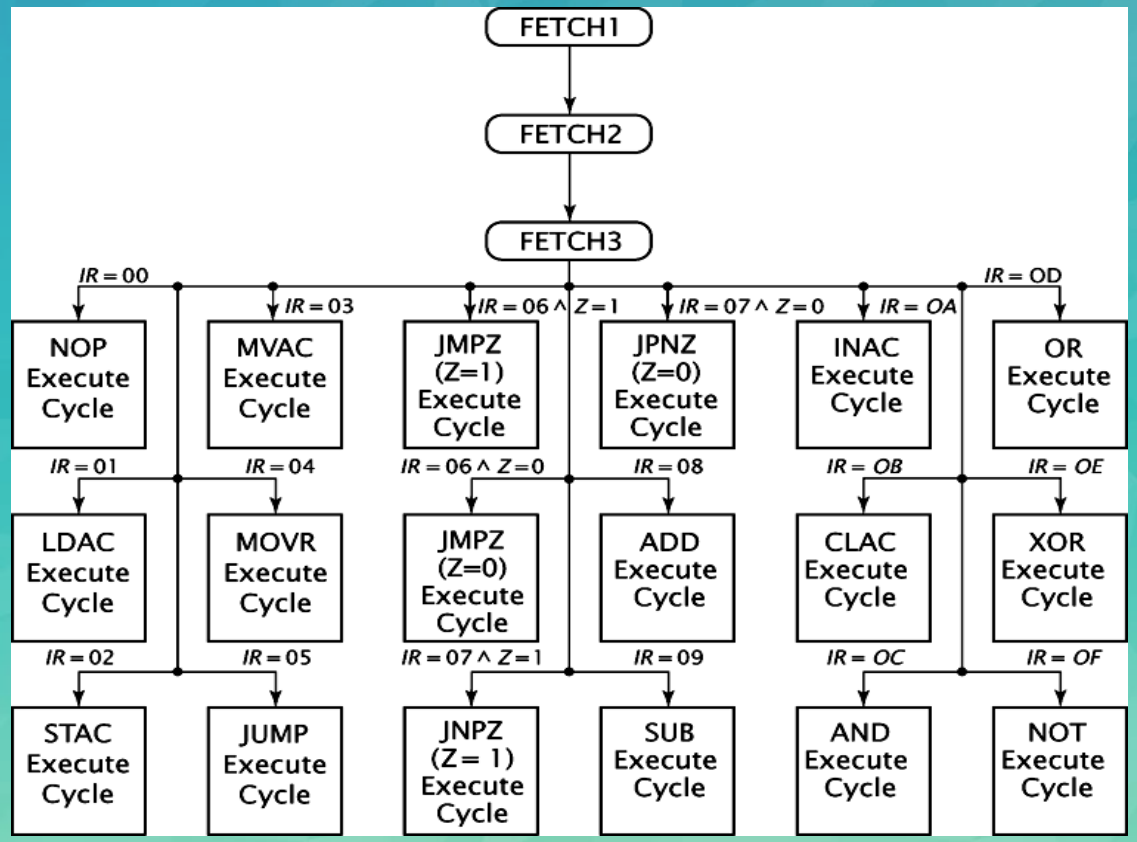
取指令和译码

FETCH1：AR←PC
FETCH2：DR←M，PC←PC + 1
FETCH3：IR←DR
FETCH4：AR←PC

根据每个状态需要完成的操作
产生相应的控制信号。

```
case state is
    when fetch1 =>      --AR←PC
        arload <= '1';
        arinc  <= '0';
        pcload <= '0';
        pcinc  <= '0';
        drload <= '0';
        trload <= '0';
        irload <= '0';
        rload  <= '0';
        acload <= '0';
        acinc  <= '0';
        writel  <= '0';
        readl   <= '0';
        membus <= '0';
        pcbus  <= '1';
        drbus  <= '0';
        trbus  <= '0';
        rbus   <= '0';
        acbus  <= '0';
        cle    <= '0';
        busmem <= '0';
```

数据通路



译码。从FETCH4状态到各指令执行状态序列的第一个状态。

```
case state is
    when fetch1 =>
        nextstate <= fetch2;
    when fetch2 =>
        nextstate <= fetch3;
    when fetch3 =>
        nextstate <= fetch4;
    when fetch4 =>
        case ir is
            when RSNOP =>
                nextstate <= nop1;
            when RSCLAC =>
                nextstate <= clac1;
            when RSSTAC=>
                nextstate <= stac1;
            when RSLDAC=>
                nextstate <= ldac1;
            when RSINAC=>
                nextstate <= inac1;
            when RSMVAC=>
                nextstate <= mvac1;
            when RSADD=>
                nextstate <= add1;
            when RSSUB=>
                nextstate <= sub1;
```

数据通路

```
-- address and data bus
addrbus <= ar;
databus <= dr when busmem='1' else "ZZZZZZZZ";
```

```
--the bus
thebus<=pc                          when pcbus='1'       else
        "00000000"&databus          when membus='1'      else
        "00000000"&r                when rbus='1'        else
        "00000000"&ac               when acbus='1'       else
        dr&tr                       when (trbus='1' and drbus='1')      else
        "00000000"&dr               when (drbus='1' and trbus/='1')     else
        "ZZZZZZZZZZZZZZZZ";
```

```
alu_result <= std_logic_vector(unsigned(ac)-unsigned(thebus(7 downto 0))) when state=subl else
              std_logic_vector(unsigned(ac)+unsigned(thebus(7 downto 0)))  when state=addl else
              std_logic_vector(ac and thebus(7 downto 0)) when state=andl else
              std_logic_vector(ac or thebus(7 downto 0)) when state=orl else
              std_logic_vector(ac xor thebus(7 downto 0)) when state=xorl else
              std_logic_vector( not ac) when state=notl else
              thebus(7 downto 0);
```

根据控制信号执行相应的操作

数据通路

DR和TR必须被同时送到总线上，DR在位15..8，TR在位7..0。

```
dr&tr                    when (trbus='1' and drbus='1')      else
"00000000"&dr            when (drbus='1' and trbus/='1')     else
"0000000000000000"
```

数据通路

零标志位Z：每次执行算术运算或者逻辑运算的时候，它都将被置位。

```
if(state=ADD1 or state=SUB1 or state=NOT1 or state=AND1 or state=OR1 or state=XOR1) then
if(alu_result="00000000") then
    z<='1';
else
    z<='0';
end if;
end if;
```

```
if(acinc='1') then
    ac <= std_logic_vector(unsigned(ac)+1);
    if(ac="11111111")   then    z<='1';
    else    z<='0';
    end if;
end if;
```

数据通路

```
if(rising_edge(clk)) then
--pc
    if(reset='1') then
        pc <= X"0000";
        state <= fetch1;
    else
        --pc <= nextpc;
        state <= nextstate;
    end if;

    if(arload='1') then
        ar <= thebus;
    end if;
    if(pcload='1') then
        pc <= thebus;
    end if;
    if(drload='1') then
        dr <= thebus(7 downto 0);
    end if;
    if(trload='1') then
        tr <= dr;
    end if;
    if(irload='1') then
        ir <= dr;
    end if;
    if(rload='1') then
        r <= thebus(7 downto 0);
    end if;
```

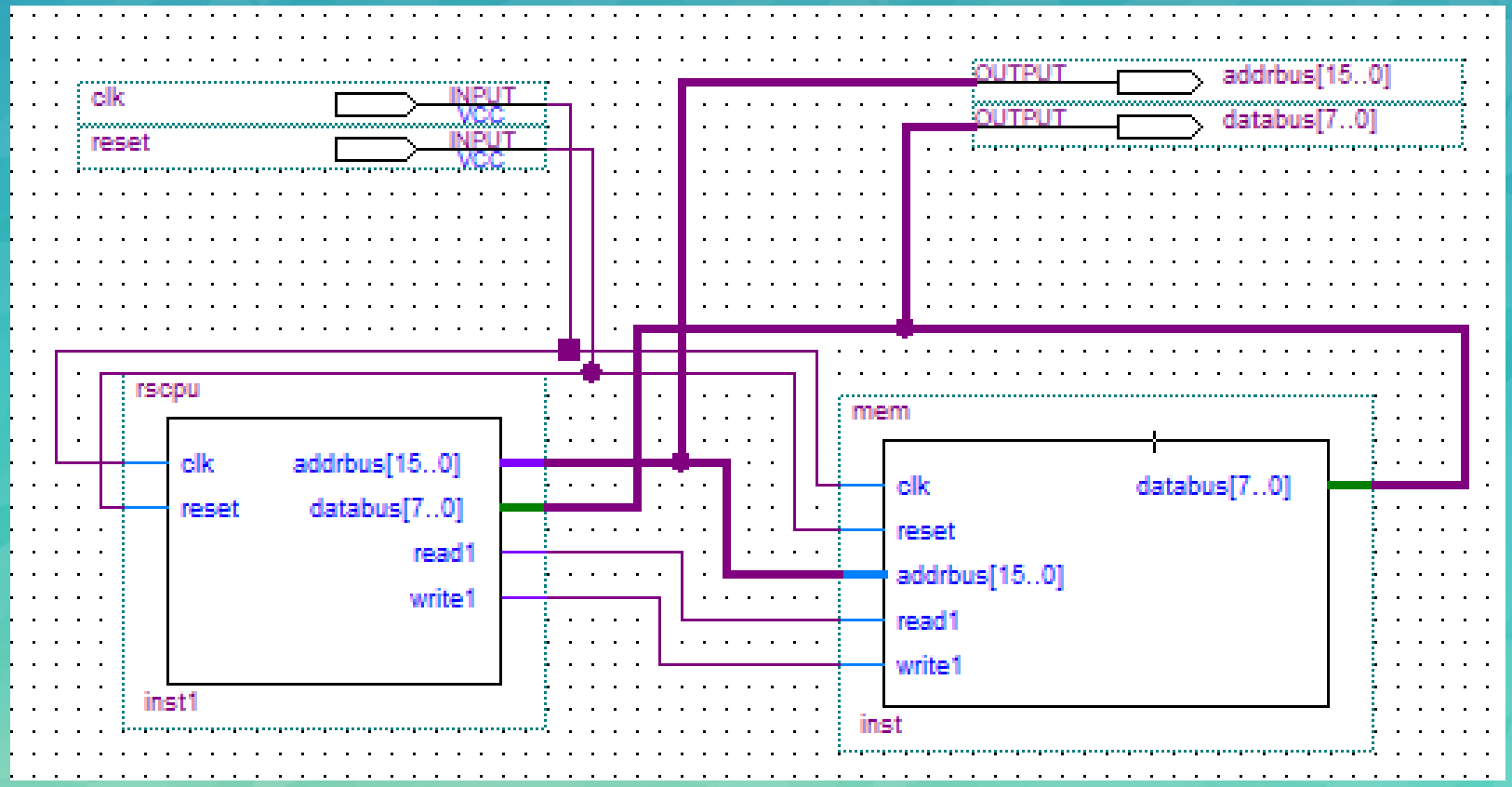
建立一条从DR输出端到IR输入端的直接通路，使得IR←DR不使用内部总线。

```
if(trload='1') then
    tr <= dr;
end if;
```

TR只从DR接受数据，所以CPU可以包含一条从DR的输出到TR的输入的直接通路。

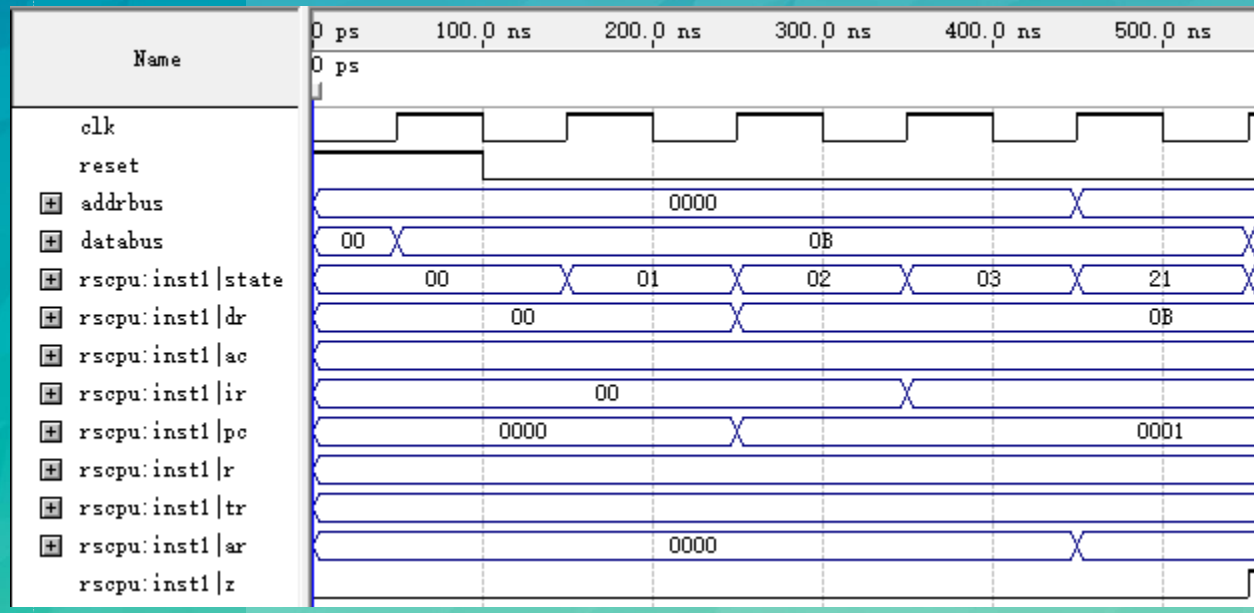
```
if(irload='1') then
    ir <= dr;
```

数据通路



根据同学的建议，不使用VHDL程序，而采用模块图连接的方式将cpu和mem连接起来。

仿真结果




```
constant total_addr : integer := 29;
constant i_addr : integer := 30;
constant n_addr : integer := 31;
constant loop_addr : integer := 7;

signal memdata: memtype(4095 downto 0) := (
0 => RSCLAC,
1 => RSSTAC,
2 => std_logic_vector(to_unsigned(total_addr, 8)),
3 => X"00",
4 => RSSTAC,
5 => std_logic_vector(to_unsigned(i_addr, 8)),
6 => X"00",
7 => RSLDAC,  -- loop
8 => std_logic_vector(to_unsigned(i_addr, 8)),
9 => X"00",
10 => RSINAC,
11 => RSSTAC,
12 => std_logic_vector(to_unsigned(i_addr, 8)),
13 => X"00",
14 => RSMVAC,
15 => RSLDAC,
16 => std_logic_vector(to_unsigned(total_addr, 8)),
17 => X"00",
18 => RSADD,
```

①

```
gen_controls: process(state)
begin
    pcbus <= fetch1 or fetch3;
    trbus <= LDAC3 or STAC3 or JUMP3 or JMPZY3 or JPNZY3;
    rbus  <= MOVR1 or ADD1 or SUB1 or AND1 or OR1 or XOR1;
    acbus <= STAC4 or MVAC1;
    drbus <= fetch3 or ldac2 or ldac5 or stac3 or stac5 or jump3 or jmpzy3 or jpnzy3;
    membus<= FETCH2 or LDAC1 or LDAC2 or LDAC4 or STAC1 or STAC2 or JUMP1 or JUMP2 or JMPZY1 or JMPZY2 or JPNZY1 or JPNZ
    busmem<= STAC5;

    cle   <= clac1;
    pcinc <= fetch2 or ldac1 or ldac2 or stac1 or stac2 or jmpzn1 or jmpzn2 or jpnzn1 or jpnzn2;
    arinc <= LDAC1 or STAC1 or JUMP1 or JMPZY1 or JPNZY1;

    arload<= FETCH1 or FETCH3 or LDAC3 or STAC3;
    pcload<= jump3 or jmpzy3 or jpnzy3;
    drload<= fetch2 or ldac1 or ldac2 or ldac4 or stac1 or stac2 or stac4;
```

老师写的代码是按不同的状态产生相应的控制信号，但是写代码的时候感觉有点繁琐，因此打算用这样的方式来写，后来又想到这样写每个状态都得赋值为1位的数，或者创建新的变量，所以就放弃了。

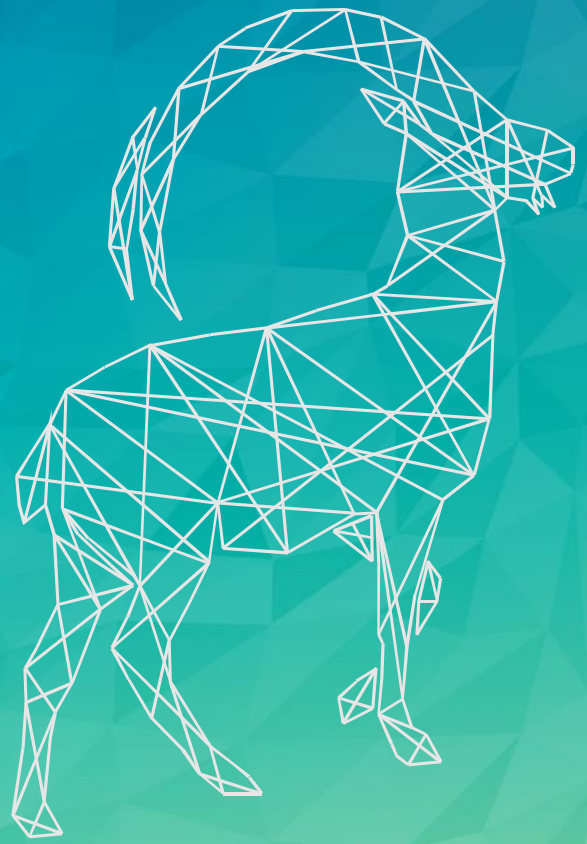
②

状态	功能	状态	功能
FETCH1	T0	JMPZY1	IJMPZ∧Z∧T3
FETCH2	T1	JMPZY2	IJMPZ∧Z∧T4
FETCH3	T2	JMPZY3	IJMPZ∧Z∧T5
NOP1	INOP∧T3	JMPZN1	IJMPZ∧Z'∧T3
LDAC1	ILDAC∧T3	JMPZN2	IJMPZ∧Z'∧T4
LDAC2	ILDAC∧T4	JPNZY1	IJPNZ∧Z'∧T3
LDAC3	ILDAC∧T5	JPNZY2	IJPNZ∧Z'∧T4
LDAC4	ILDAC∧T6	JPNZY3	IJPNZ∧Z'∧T5
LDAC5	ILDAC∧T7	JPNZN1	IJPNZ∧Z∧T3

FETCH3：IR←DR，AR←PC

```
case state is
    when fetch1 =>
        nextstate <= fetch2;
    when fetch2 =>
        nextstate <= fetch3;
    when fetch3 =>
        nextstate <= fetch4;
    when fetch4 =>
        case ir is
            when RSNOP =>
                nextstate <= nop1;
            when RSCLAC =>
                nextstate <= clac1;
```

上次报告CPU的时候，讨论了fetch3状态，因为fetch3时将数据加载到IR寄存器中，同时还要根据IR寄存器中的值判断下一状态，可能会有冲突，发生错误。后来我仔细看了PPT，我认为PPT是正确的，因为它有明确每个指令的执行时间，fetch是T2，而后面进行判断下一状态是T3。不过我们没有去实现让这两步操作隔一个时钟，因此我还是采用的fetch4的方法。



感谢您的观看与欣赏

Happy New Year!