

CPU设计实验

汇报人：潘小天
智能1601
201608010309

CONTENTS



实验内容



设计思路



代码解析



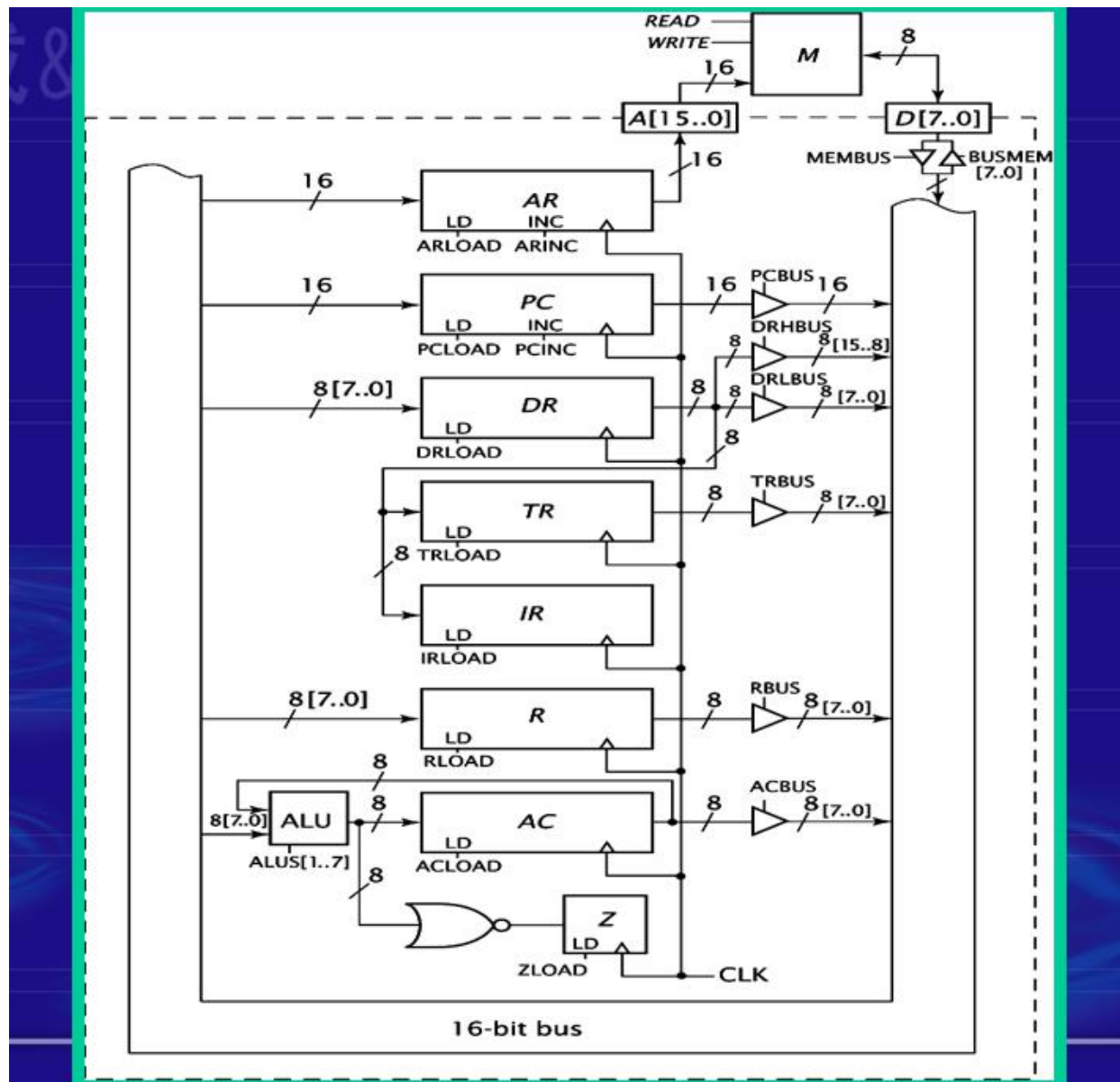
仿真结果

The background of the slide features a person in profile, focused on a task, possibly a craft or repair project. A large, semi-transparent orange circle with a white border and a white outline is positioned on the left side of the image. The text "Part 01" is written in white, bold, sans-serif font inside this circle.

Part 01

实验内容

设计相对简单的CPU，按照取址、译码、执行三个阶段来设计cpu的动作。其数据通路如下：



The background of the slide features a person in profile, focused on a task, possibly a craft or design project. A large, semi-transparent orange circle with a white border and a white outline is positioned on the left side of the image. The text 'Part 02' is written in white, bold, sans-serif font inside this circle.

Part 02

设计思路

02

设计思路

设计CPU的步骤:

- ◆ 确定它的用途

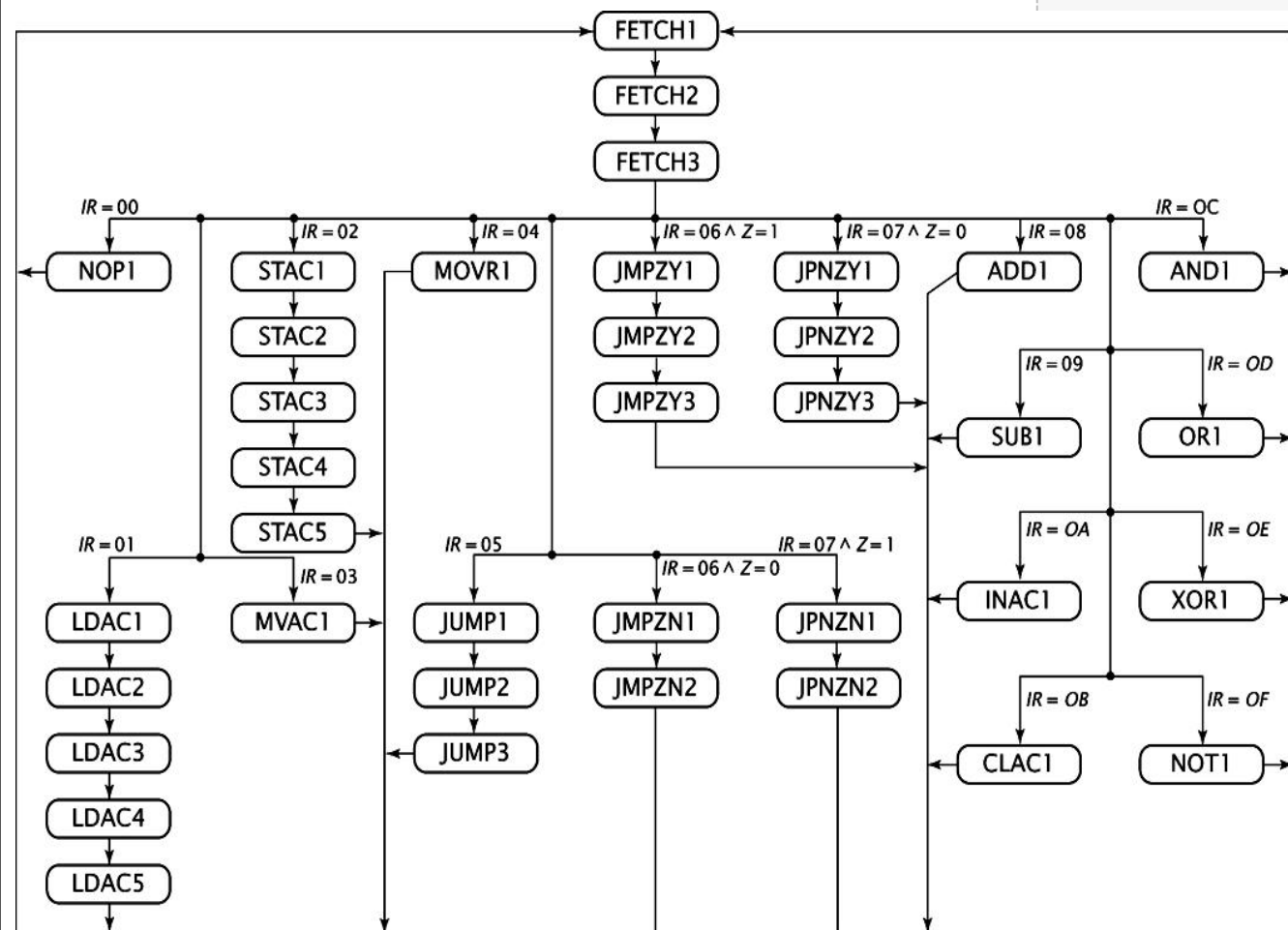
关键: 使CPU的处理能力和它所执行的任务匹配。

- ◆ 设计指令集结构

- ◆ 设计状态图

列出在每个状态中执行的微操作

从一个状态转移到另外一个状态的条件



1

rsisa.vhd: 声明每条指令对应的变量名。

2

mem.vhd: 内存的vhdl代码。在这里声明内存的大小、初始化内存, 并规定读写信号 (read、write) 有效时内存的动作。

3

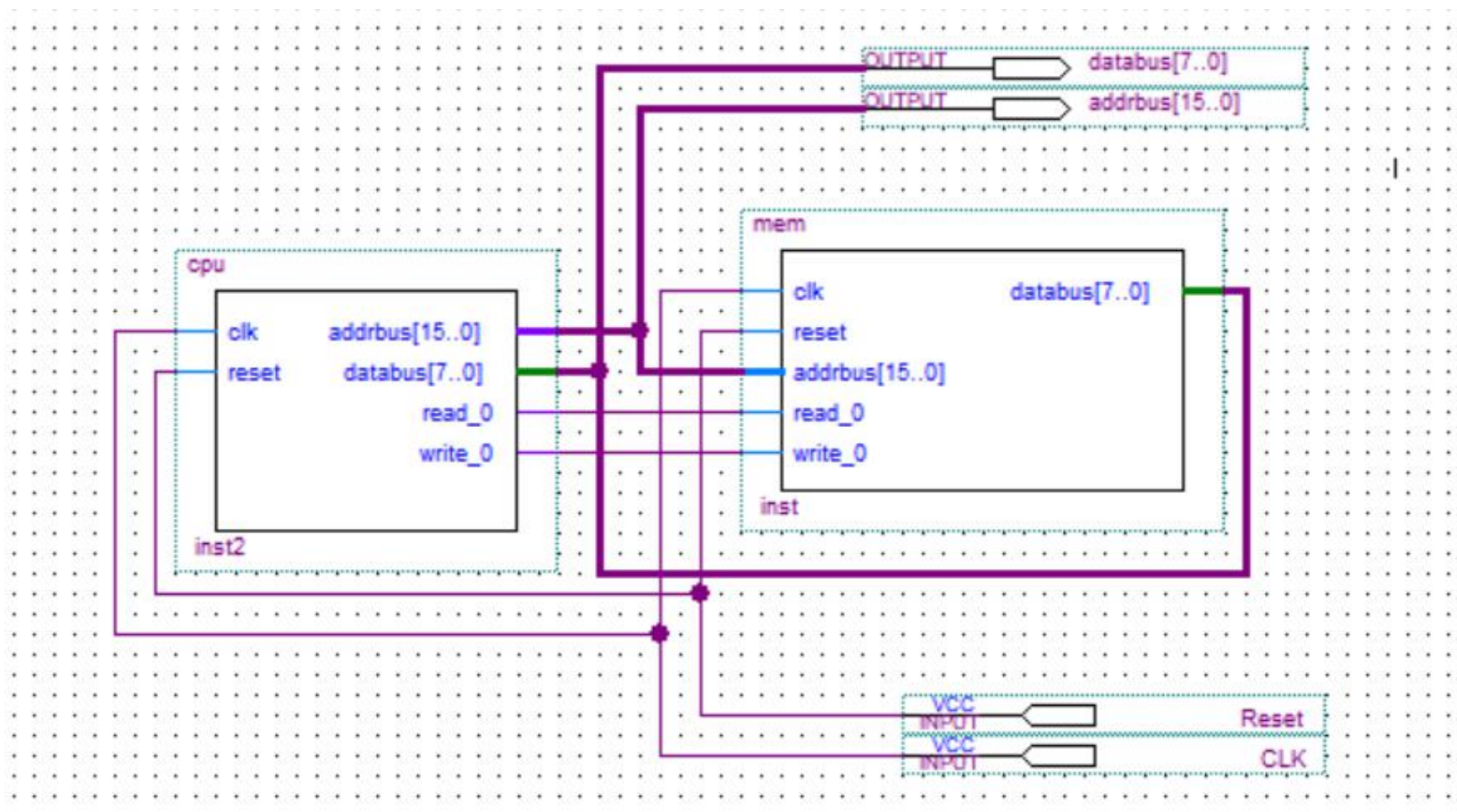
cpu.vhd: cpu的vhdl代码。这里声明cpu的内部组成、cpu可能达到的各个状态, 和cpu处于各个状态下采取的动作。

The background of the slide features a person with a nose ring, seen from the side, working on a project. The image is dimmed and serves as a backdrop for the title elements.

Part 03

代码解析

顶层设计如下图所示：



声明每条指令对应的
变量名，如：

“0000 0000” 对应
RSNOP

“0000 0001” 对应
RSLADC

```
package rsisa is
```

```
-- RS prefix is used to avoid tautonym such like AND, OR, XOR, NOT
```

```
constant RSNOP: std_logic_vector(7 downto 0) := "00000000";
```

```
constant RSLDAC: std_logic_vector(7 downto 0) := "00000001";
```

```
constant RSSTAC: std_logic_vector(7 downto 0) := "00000010";
```

```
constant RSMVAC: std_logic_vector(7 downto 0) := "00000011";
```

```
constant RSMOVR: std_logic_vector(7 downto 0) := "00000100";
```

```
constant RSJUMP: std_logic_vector(7 downto 0) := "00000101";
```

```
constant RSJMPZ: std_logic_vector(7 downto 0) := "00000110";
```

```
constant RSJPNZ: std_logic_vector(7 downto 0) := "00000111";
```

```
constant RSADD: std_logic_vector(7 downto 0) := "00001000";
```

```
constant RSSUB: std_logic_vector(7 downto 0) := "00001001";
```

```
constant RSINAC: std_logic_vector(7 downto 0) := "00001010";
```

```
constant RSCLAC: std_logic_vector(7 downto 0) := "00001011";
```

```
constant RSAND: std_logic_vector(7 downto 0) := "00001100";
```

```
constant RSOR: std_logic_vector(7 downto 0) := "00001101";
```

```
constant RSXOR: std_logic_vector(7 downto 0) := "00001110";
```

```
constant RSNOT: std_logic_vector(7 downto 0) := "00001111";
```



代码解析

```
for_clk : process(clk)
begin
    if(falling_edge(clk)) then
        if(reset='1') then
            addr <= (others=>'0');
        else
            addr <= addrbus;
        end if;

        if(write_0='1') then
            memdata(to_integer(unsigned(addr))) <= databus;
        end if;
    end if;
end process;

databus <= memdata(to_integer(unsigned(addr))) when (write_0='0') else "ZZZZZZZZ";
```

1. n设置成8
2. 时钟下降沿执行对mem的读写操作
3. 没有使用rw变量, 而是直接对write的值进行判断



代码解析

取址及译码阶段用FETCH1、
FETCH2、FETCH3、FETCH4这4
个状态来完成：

FETCH1: $AR \leq PC$

FETCH2: $DR \leq M$ $PC \leq PC + 1$

FETCH3: $IR \leq DR$

FETCH4: $AR \leq PC$ (这样设计是
为了方便每条指令的执行)

```
constant fetch1: std_logic_vector(5 downto 0) := "000000";-- ar<=pc
constant fetch2: std_logic_vector(5 downto 0) := "000001";-- dr<=m pc<=pc+1
constant fetch3: std_logic_vector(5 downto 0) := "000010";-- ir<=dr
constant fetch4: std_logic_vector(5 downto 0) := "000011";-- ar<=pc
constant clacl: std_logic_vector(5 downto 0) := "000100";--ac<=0 z<=1
constant incacl: std_logic_vector(5 downto 0) := "000101";--ac++ z change
constant addl: std_logic_vector(5 downto 0) := "000110";--ac=ac+r z change
constant subl: std_logic_vector(5 downto 0) := "000111";--ac=ac-r z change
constant andl: std_logic_vector(5 downto 0) := "001000";--ac=ac&r z change
constant orl: std_logic_vector(5 downto 0) := "001001";--ac=ac|r z change
constant xorl: std_logic_vector(5 downto 0) := "001010";--ac=ac xor r z change
constant notl: std_logic_vector(5 downto 0) := "001011";--ac=not ac z change
constant mvac1: std_logic_vector(5 downto 0) := "001100";--r<=ac
constant movr1: std_logic_vector(5 downto 0) := "001101";--ac<=r
constant ldac1: std_logic_vector(5 downto 0) := "001110";--dr<=m pc=pc+1 ar=ar+1
constant ldac2: std_logic_vector(5 downto 0) := "001111";--tr<=dr dr<=m pc=pc+1
constant ldac3: std_logic_vector(5 downto 0) := "010000";--ar<=dr, tr
constant ldac4: std_logic_vector(5 downto 0) := "010001";--dr<=m
constant ldac5: std_logic_vector(5 downto 0) := "010010";--ac<=dr
constant stac1: std_logic_vector(5 downto 0) := "010011";--dr<=m pc++ ar++
constant stac2: std_logic_vector(5 downto 0) := "010100";--tr<=dr dr<=m pc++
constant stac3: std_logic_vector(5 downto 0) := "010101";--ar<=dr, tr
```



代码解析

执行每个状态下的操作，指令的执行阶段可以用状态的转换来实现：state<=next_state

```
gen_controls: process(state)
begin
    if(state=fetch1) then
        arload<='1';pcbus<='1';pcinc<='0';drload<='0';membus<='0';irload<='0';acreset<='0';acinc<='0';--ar<=pc
        rbus<='0';s<="0000";acload<='0';rload<='0';acbus<='0';arinc<='0';trbus<='0';drbus<='0';trload<='0';
        read_0<='0';write_1<='0';write_0<='0';pcload<='0';
    elsif(state=fetch2) then
        arload<='0';pcbus<='0';pcinc<='1';drload<='1';membus<='1';irload<='0';acreset<='0';acinc<='0';-- dr<=m  pc<=pc+1
        rbus<='0';s<="0000";acload<='0';rload<='0';acbus<='0';arinc<='0';trbus<='0';drbus<='0';trload<='0';
        read_0<='1';write_1<='0';write_0<='0';pcload<='0';
    elsif(state=fetch3) then
        arload<='0';pcbus<='0';pcinc<='0';drload<='0';membus<='0';irload<='1';acreset<='0';acinc<='0';--ir<=dr
        rbus<='0';s<="0000";acload<='0';rload<='0';acbus<='0';arinc<='0';trbus<='0';drbus<='0';trload<='0';
        read_0<='0';write_1<='0';write_0<='0';pcload<='0';
    elsif(state=fetch4) then
        arload<='1';pcbus<='1';pcinc<='0';drload<='0';membus<='0';irload<='0';acreset<='0';acinc<='0';--ar<=pc
        rbus<='0';s<="0000";acload<='0';rload<='0';acbus<='0';arinc<='0';trbus<='0';drbus<='0';trload<='0';
        read_0<='0';write_1<='0';write_0<='0';pcload<='0';
    elsif(state=clacl) then
        arload<='0';pcbus<='0';pcinc<='0';drload<='0';membus<='0';irload<='0';acreset<='1';acinc<='0';--ac<=0  z<=1
        rbus<='0';s<="0000";acload<='0';rload<='0';acbus<='0';arinc<='0';trbus<='0';drbus<='0';trload<='0';
        read_0<='0';write_1<='0';write_0<='0';pcload<='0';
end
```




状态之间的转换:

```
if (state=fetch1) then nextstate<=fetch2;
elseif (state=fetch2) then nextstate<=fetch3;
elseif (state=fetch3) then nextstate<=fetch4;
elseif (state=fetch4) then
    if (ir=RSCLAC) then nextstate<=clacl;
    elseif (ir=RSINAC) then nextstate<=incacl;
    elseif (ir=RSADD) then nextstate<=addl;
    elseif (ir=RSSUB) then nextstate<=subl;
    elseif (ir=RSAND) then nextstate<=andl;
    elseif (ir=RSOR) then nextstate<=orl;
    elseif (ir=RSXOR) then nextstate<=xorl;
    elseif (ir=RSNOT) then nextstate<=notl;
    elseif (ir=RSMVAC) then nextstate<=mvac1;
    elseif (ir=RSMOVR) then nextstate<=movrl;
    elseif (ir=RSLDAC) then nextstate<=ldacl;
    elseif (ir=RSSTAC) then nextstate<=stacl;
    elseif (ir=RSJUMP) then nextstate<=jumpl;
```


The background of the slide features a person, likely a woman, working on a project. She is wearing a dark t-shirt with a colorful pattern on the sleeve and has a red string tied around her wrist. She is looking down at her work, which appears to be a small object or component. The background is slightly blurred, focusing attention on the person and the text overlay.

Part 04

仿真结果

仿真结果

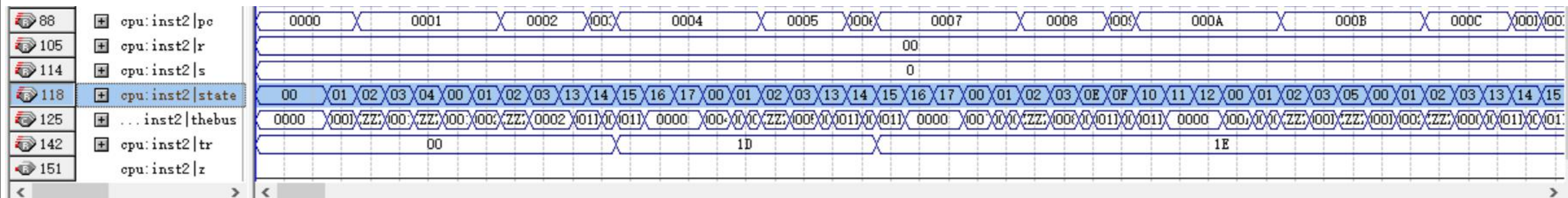
整体的测试结果：

clk: 时钟信号 (周期为50ns)

reset: 重置信号

state: 状态

res: total的复制值，用来显示total的结果



The background is a blurred photograph of a person's hands holding a magazine titled 'UPPERCASE' on a wooden table. A white cup of coffee sits on a saucer to the right. Overlaid on this is a large white circle with a thick orange border made of several concentric, slightly offset rings.

感谢您的观看

汇报人：潘小天