

2018

CPU设计实验报告

201608010219 罗泉鸿 智能1601

目

录

CONTENTS

001

实验内容

002

设计思路

003

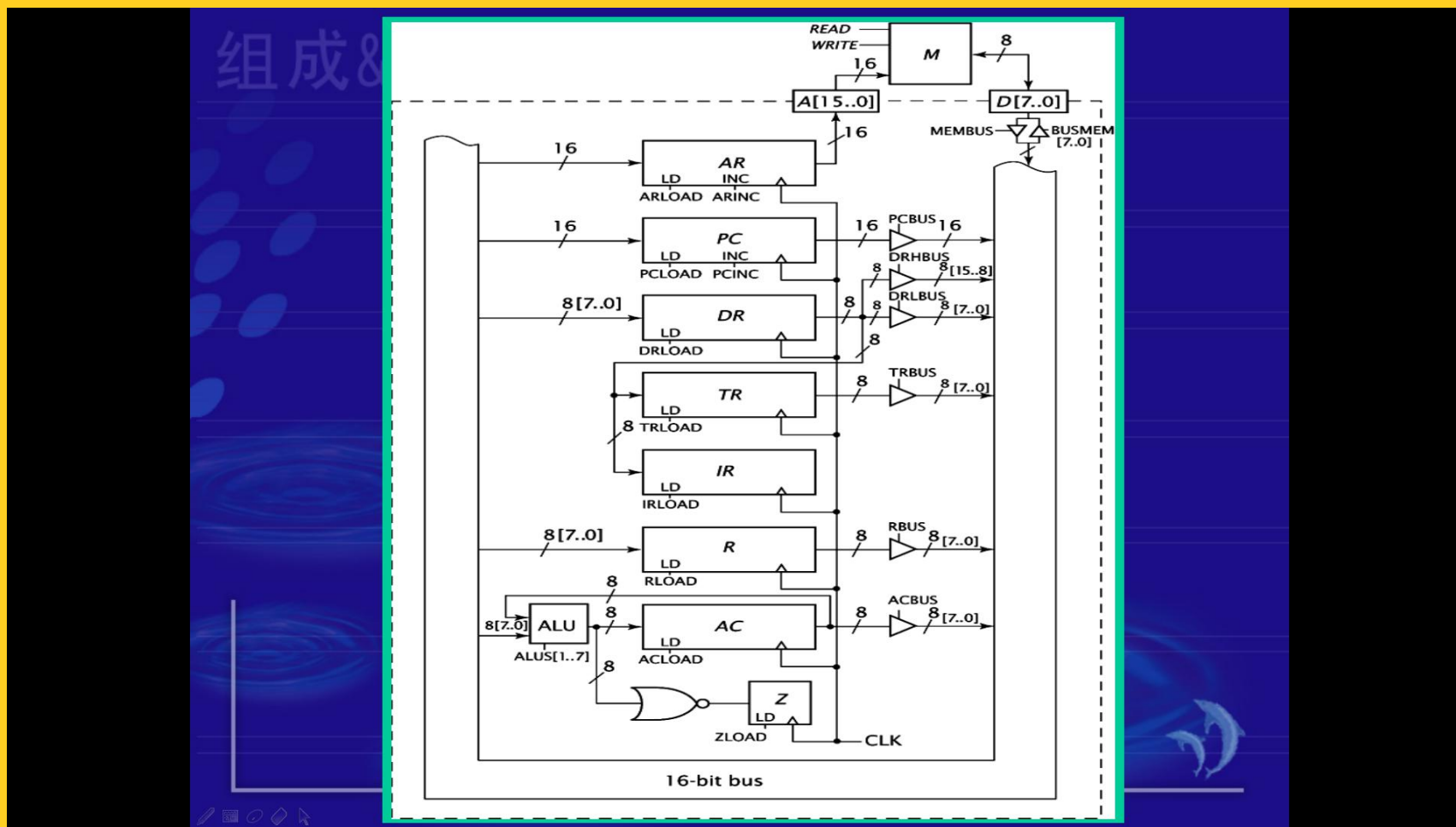
代码解析

004

仿真结果

实验内容

设计相对简单的CPU，其数据通路如下所示。实现LDAC、STAC等16条指令，指令的具体内容在课件中。



设计思路

按照**取址**、**译码**、**执行**三个阶段来设计cpu的动作。

取址及译码阶段可以用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）

而指令的执行阶段可以用状态的转换来实现：

$state \leq next_state$

设计思路

代码分成4部分进行编写：

rsisa.vhd：声明每条指令对应的变量名。

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。

cpu.vhd：cpu的vhdl代码。在这里声明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。

代码解析 (rsisa.vhd)

```
rsisa.vhd
~/school/computerSYS

1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4
5
6 package rsisa is
7
8     -- RS prefix is used to avoid tautonym such like AND, OR, XOR, NOT
9     constant RSNOP: std_logic_vector(7 downto 0) := "00000000";
10    constant RSLDAC: std_logic_vector(7 downto 0) := "00000001";
11    constant RSSTAC: std_logic_vector(7 downto 0) := "00000010";
12    constant RSMVAC: std_logic_vector(7 downto 0) := "00000011";
13    constant RSMOVR: std_logic_vector(7 downto 0) := "00000100";
14    constant RSJUMP: std_logic_vector(7 downto 0) := "00000101";
15    constant RSJMPZ: std_logic_vector(7 downto 0) := "00000110";
16    constant RSJPNZ: std_logic_vector(7 downto 0) := "00000111";
17
18    constant RSADD: std_logic_vector(7 downto 0) := "00001000";
19    constant RSSUB: std_logic_vector(7 downto 0) := "00001001";
20    constant RSINAC: std_logic_vector(7 downto 0) := "00001010";
21    constant RSCLAC: std_logic_vector(7 downto 0) := "00001011";
22    constant RSAND: std_logic_vector(7 downto 0) := "00001100";
23    constant RSOR: std_logic_vector(7 downto 0) := "00001101";
24    constant RSXOR: std_logic_vector(7 downto 0) := "00001110";
25    constant RSNOT: std_logic_vector(7 downto 0) := "00001111";
26
27 end package;
```

VHDL 制表符宽度: 4 第 13 行, 第 64 列 插入

如左图所示，rsia主要是指定每条指令对应的变量名，如：

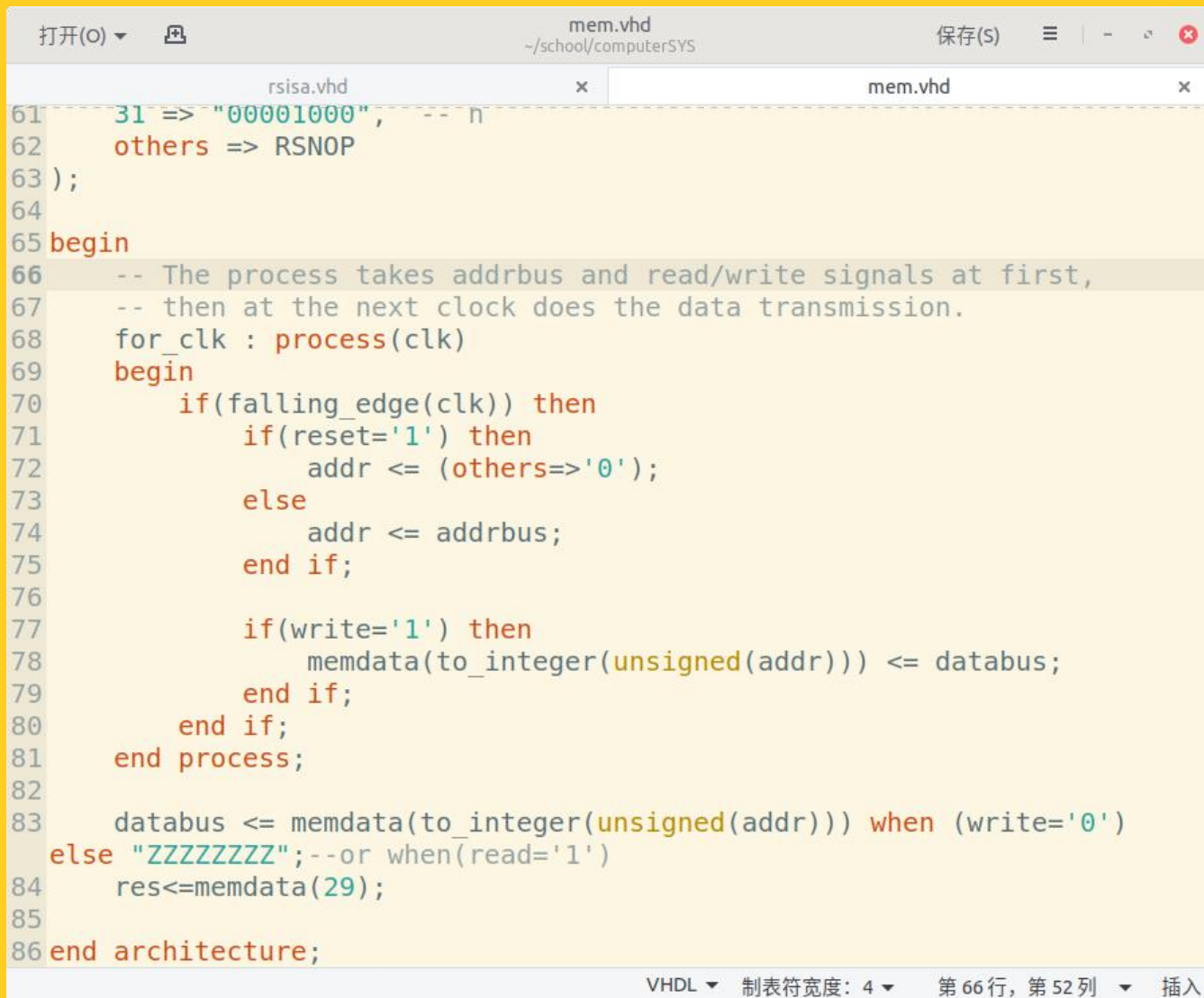
“0000 0000”对应RSNOP

“0000 0001”对应RSLADC

.....

在另外的代码中包含rsisa这个包，就可以使用指令的变量名而不需要每次进行判断时都输入“XXXX XXXX”了。

代码解析 (mem.vhd)



```
61 31 => "00001000", -- n
62 others => RSNOP
63 );
64
65 begin
66 -- The process takes addrbus and read/write signals at first,
67 -- then at the next clock does the data transmission.
68 for_clk : process(clk)
69 begin
70     if(falling_edge(clk)) then
71         if(reset='1') then
72             addr <= (others=>'0');
73         else
74             addr <= addrbus;
75         end if;
76
77         if(write='1') then
78             memdata(to_integer(unsigned(addr))) <= databus;
79         end if;
80     end if;
81 end process;
82
83 databus <= memdata(to_integer(unsigned(addr))) when (write='0')
84 else "ZZZZZZZZ"; --or when(read='1')
85 res<=memdata(29);
86 end architecture;
```

左图是mem.vhd的部分代码，大体与给出的参考代码rsmem.vhd相似。改动的地方主要是：

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（cpu.vhd）

下面主要对cpu.vhd的architecture部分进行说明：

architecture部分主要有一下内容组成：

architecture

...

内部信号声明

——>声明CPU的各个寄存器、声明内部总线、
声明各个控制信号、声明每一个状态的编
码.....

...

process外部的逻辑内容

——>在这里写一些逻辑功能：总线上数据的
传输、ALU的运算

...

process(clk)

——>这里写时钟上升沿要做的事情：

- ① 状态的更替
- ② 根据控制信号的值来确定做什么

...

for_nextstate: process(state, ir, z)

——>这个进程用于生成下一个状态

——>在这里写好每个状态的下一个状态是
什么

...

gen_controls: process(state)

——>这个进程用于根据当前状态给每一个
控制信号赋值

——>在这里写好每个状态下每个控制信号
的值

...

end

代码解析（cpu.vhd）

编写cpu.vhd的工作主要是在参考代码rscpu.vhd的基础上进行添加添加：

添加内部信号、内部寄存、状态；

添加状态与下一状态的转移；

添加时钟上升沿的动作、当前状态的动作。

但是，也有与rscpu.vhd不同的地方：

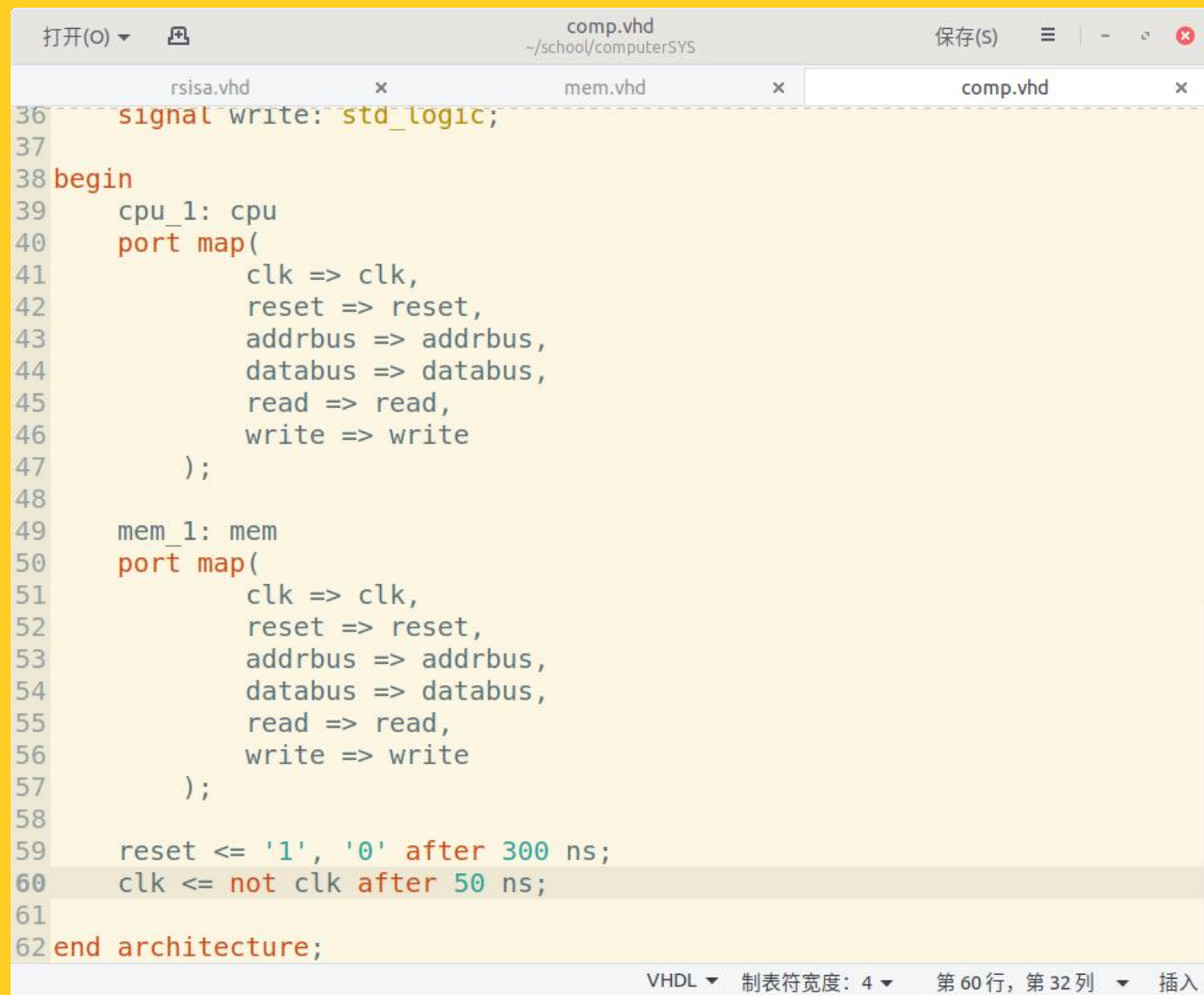
1. 没有产生下一个pc（next_pc）的进程

理由：因为每个状态都有确定的下一状态，每一状态又有确定的动作，所以没有必要使用改变pc的进程。

2. 使用FETCH4，其内容是：AR<=PC，而FETCH3的内容改为：IR<=DR

理由：IR没从DR得到正确的值的时候，state已经改变了，导致无法正确的译码。

代码解析 (comp.vhd)



```
36 signal write: std_logic;
37
38 begin
39   cpu_1: cpu
40   port map(
41     clk => clk,
42     reset => reset,
43     addrbus => addrbus,
44     databus => databus,
45     read => read,
46     write => write
47   );
48
49   mem_1: mem
50   port map(
51     clk => clk,
52     reset => reset,
53     addrbus => addrbus,
54     databus => databus,
55     read => read,
56     write => write
57   );
58
59   reset <= '1', '0' after 300 ns;
60   clk <= not clk after 50 ns;
61
62 end architecture;
```

左图是comp.vhd的部分代码，与给出的参考代码rscomp.vhd相同。

需要在意的地方是：

1. 时间周期为100ns
2. 前300ns重置信号（reset）有效

仿真结果

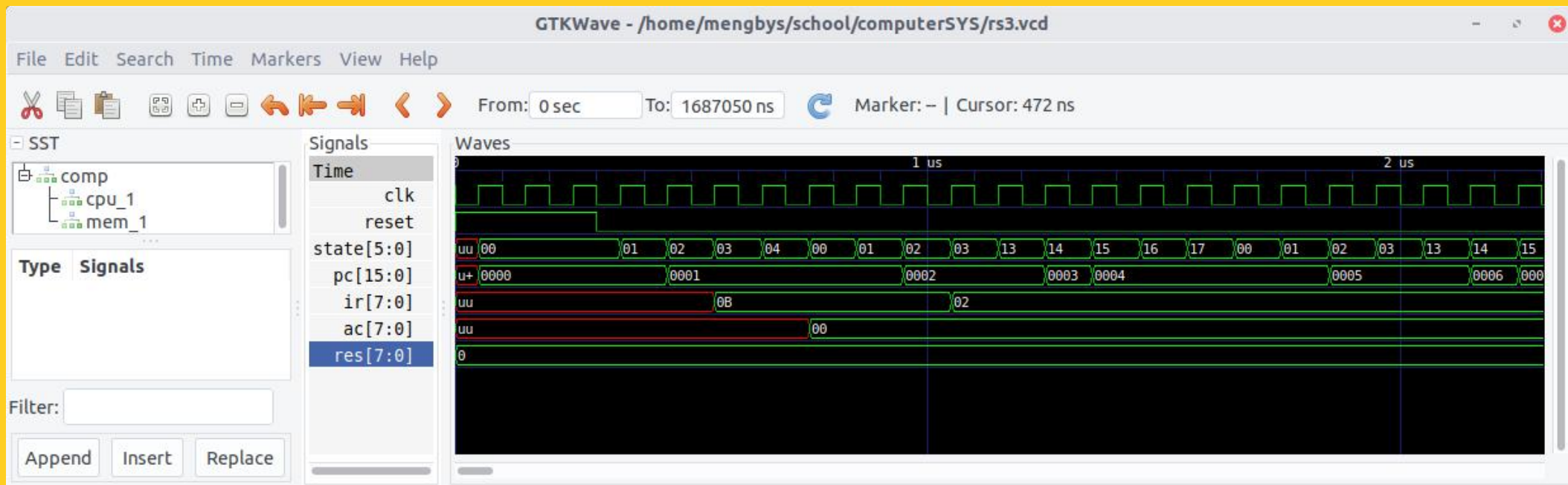
clk : 时钟信号 (周期为100ns)

reset : 重置信号

state : 状态

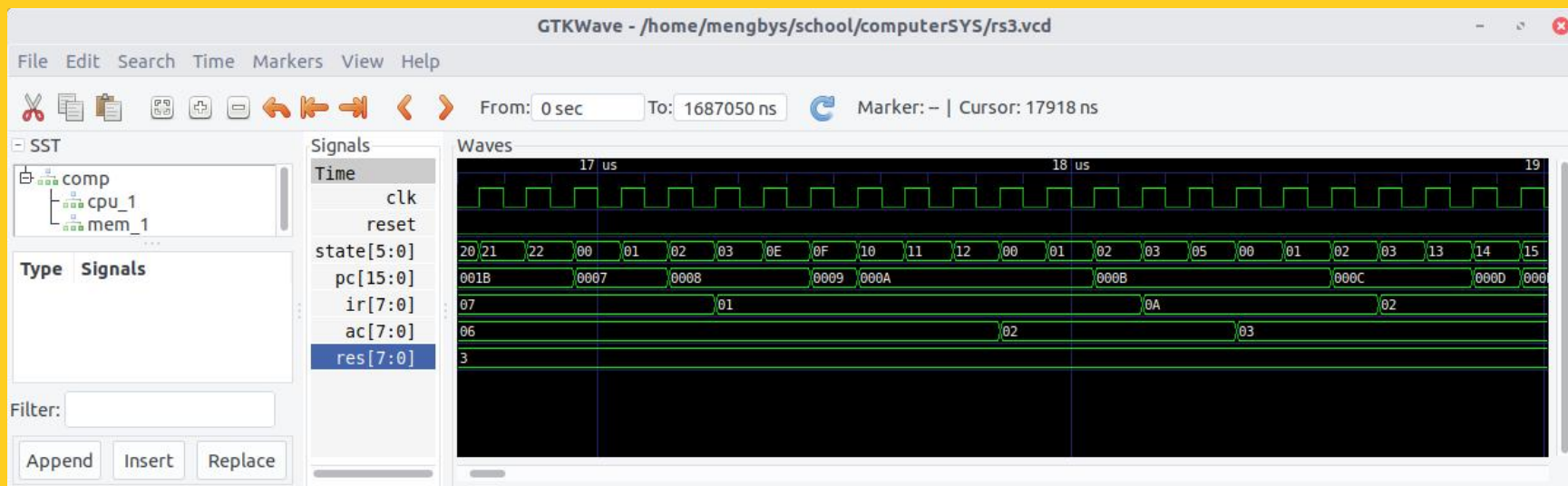
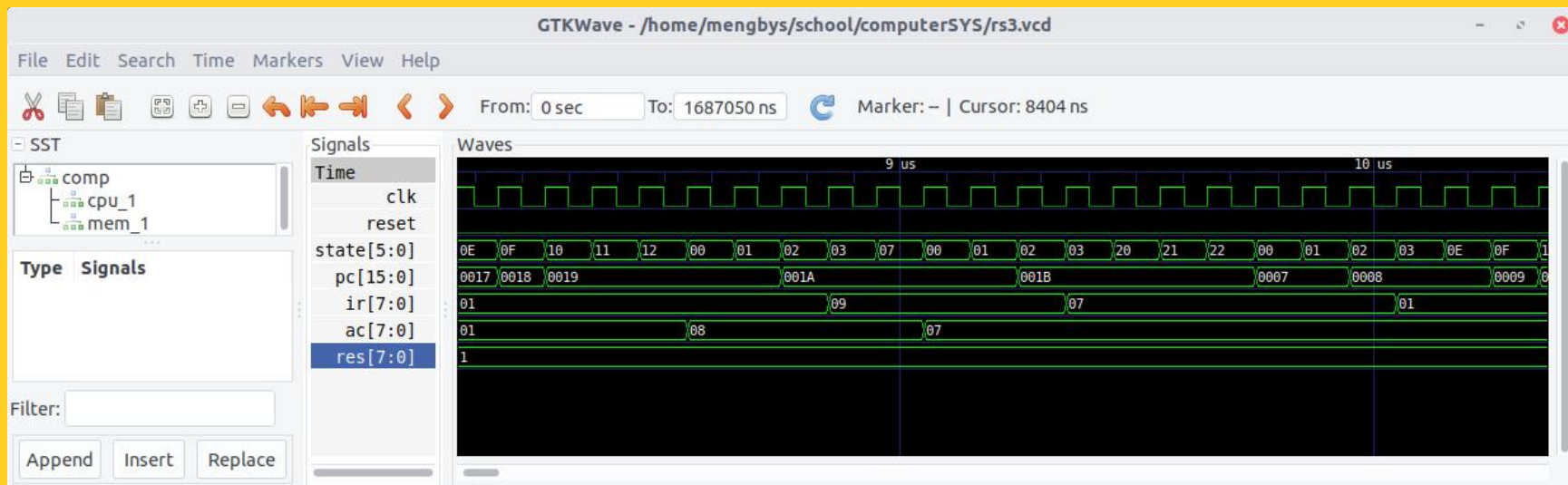
res : total的复制值 , 用来显示total的结果

可以看到 , 初始状态res=0

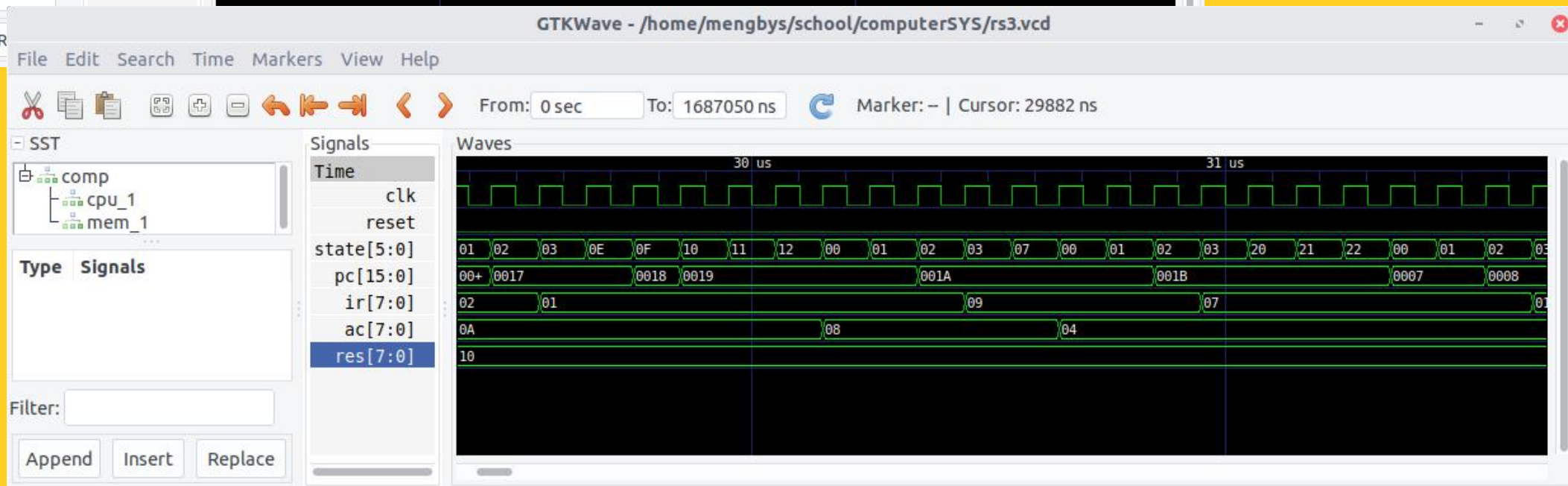
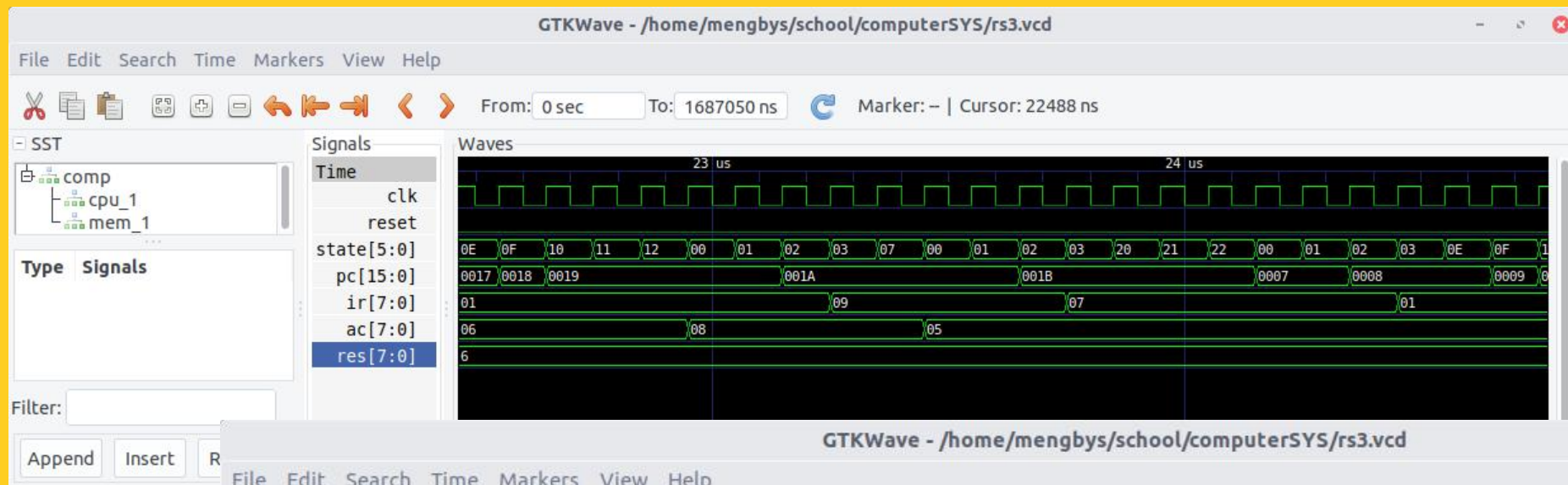


仿真结果

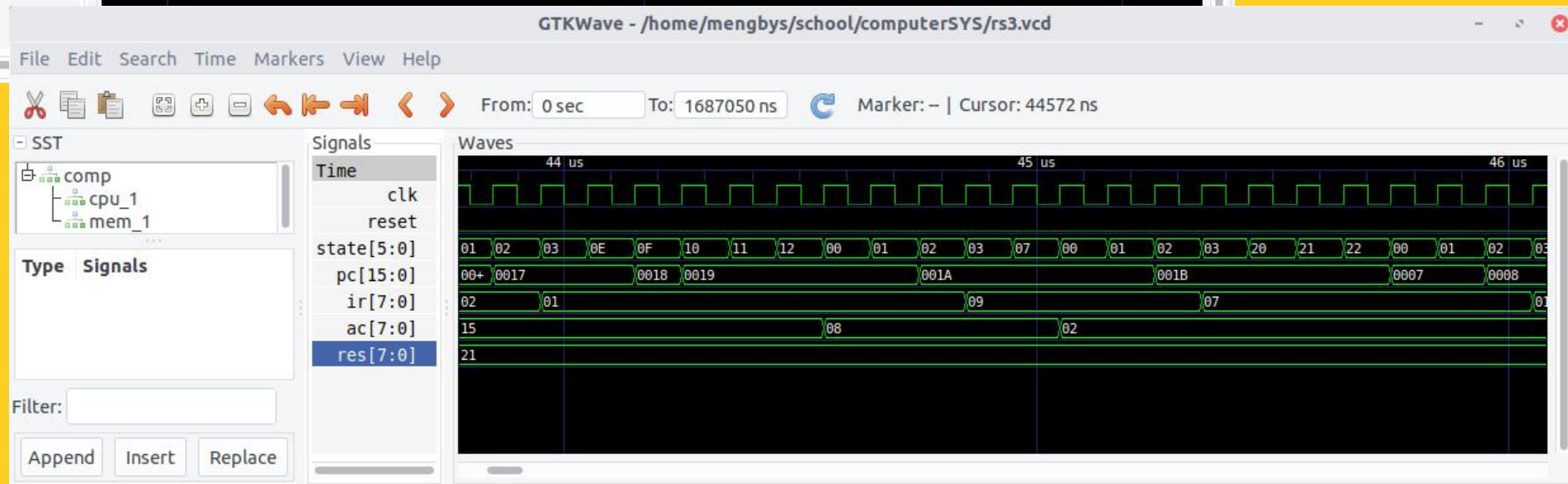
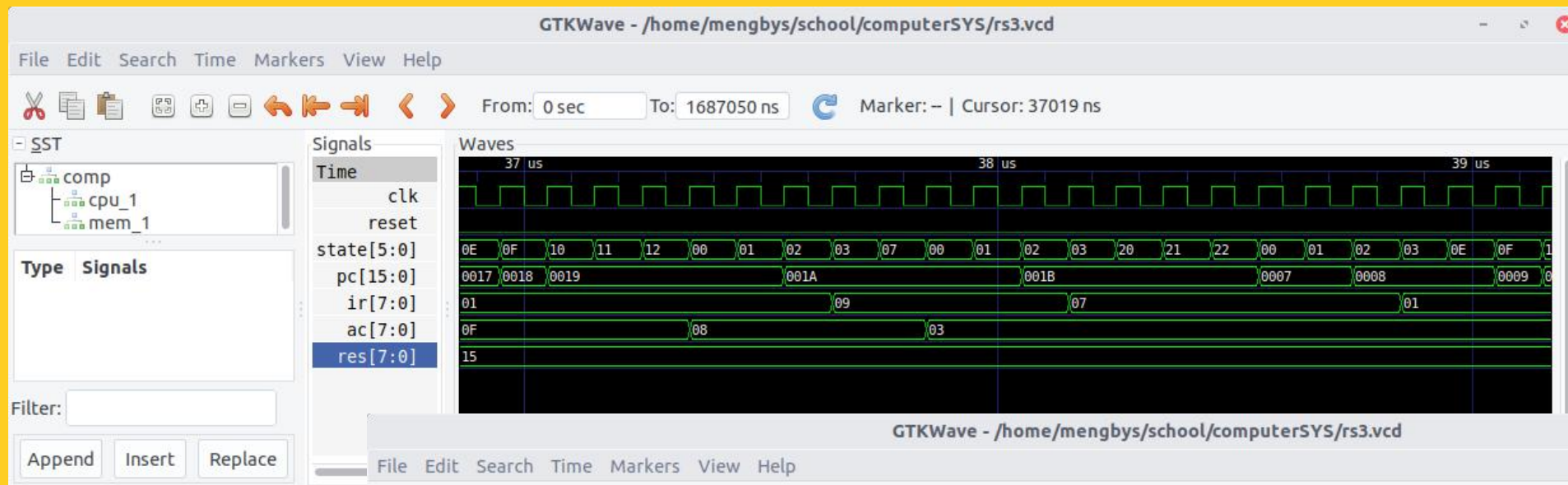
以下截图展示了res的变化过程：



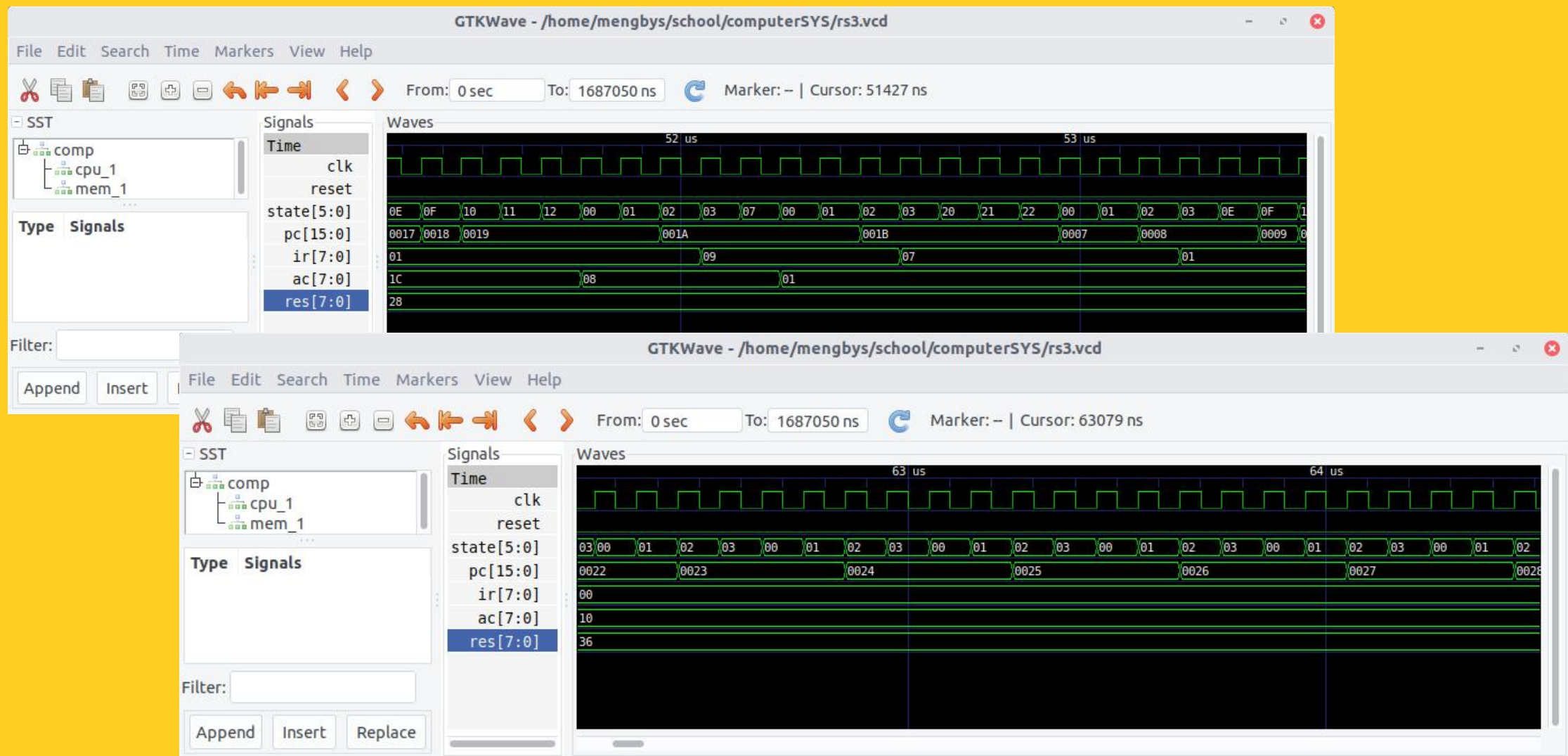
仿真结果



仿真结果



仿真结果



仿真结果

当res变成36之后，它不在发生改变，而且res是按如下顺序变化：

0->1->3->6->10->15->21->28->36

可以知道，这正是我们设计时想要的结果。cpu运行正确。

谢谢！