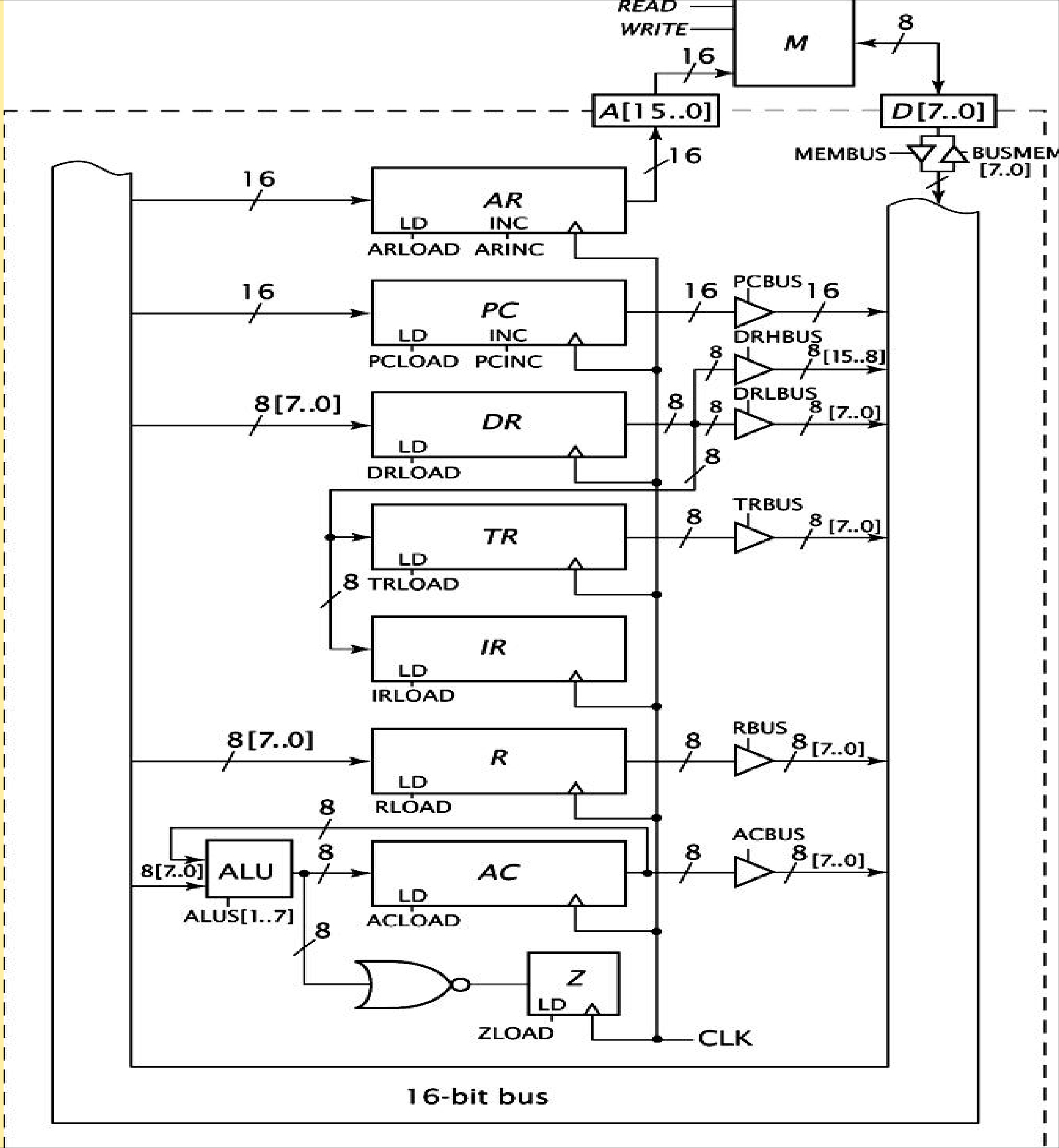
相对简单的CPU设计

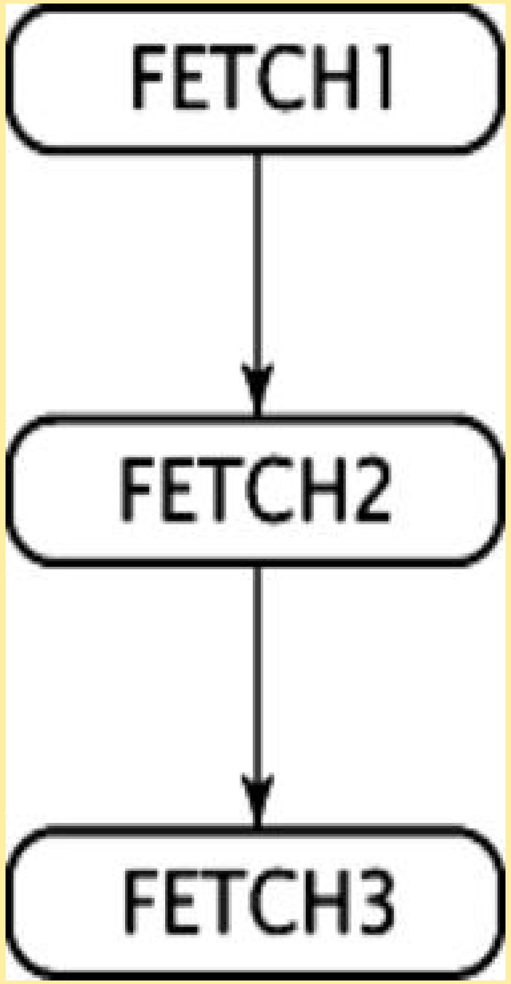
智能1602 201608010703   孙元伟

实验内容

实验的数据通路如右图所示
实现各种指令，例如LDAC，STAC，MVAC等



设计思路



取址阶段:

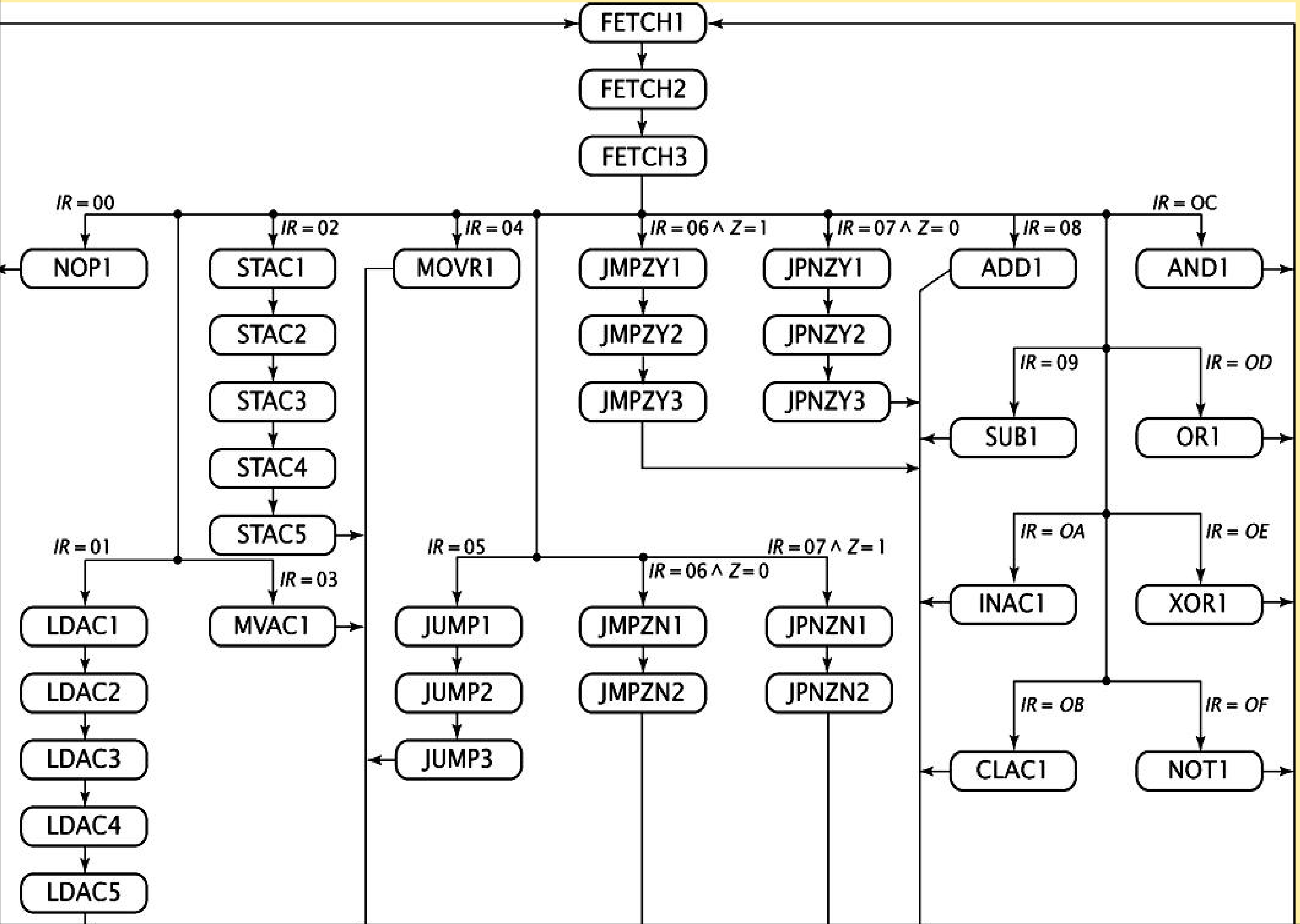
FETCH1: AR←PC

FETCH2: DR←M，PC←PC＋1

FETCH3: IR←DR[7..6]，AR←DR[5..0]

IR指令寄存器，DR数据寄存器

设计思路



译码阶段：

CPU把一条指令从存储器中取出来之后，判断所取的是哪种指令，从而可以调用正确的执行周期。

设计思路

执行阶段:

以LDAC为例:

第一个状态:

LDAC1: DR←M, PC←PC + 1, AR←AR + 1

第二个状态:

LDAC2: TR←DR, DR←M, PC←PC + 1

LDAC3: AR←DR, TR

LDAC4: DR←M

LDAC5: AC←DR

设计思路

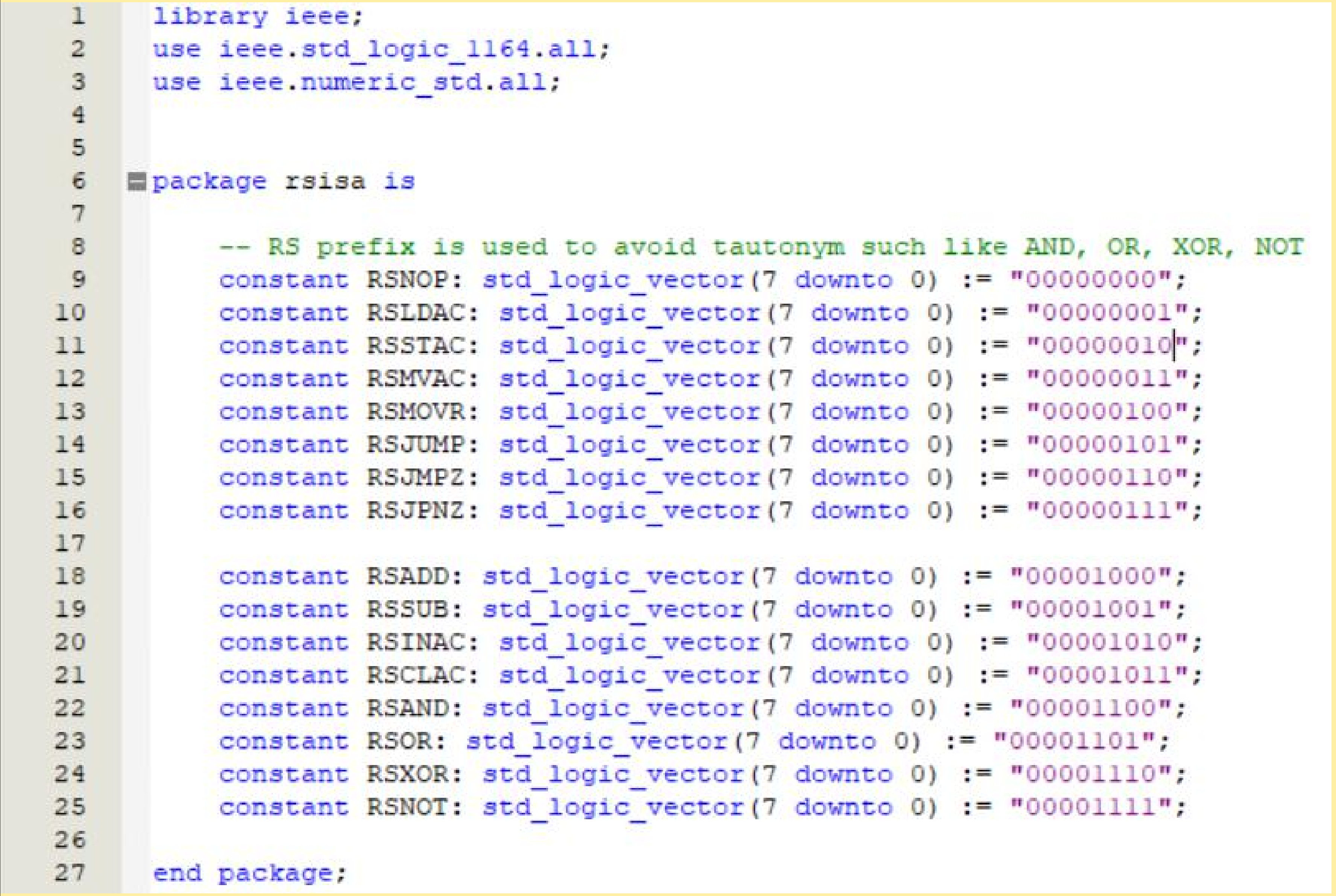
我们分为了3个vhd文件实现:

指令集：rsisa.vhd （声明每条指令对应的变量名）

内存：mem.vhd （初始化内存单元，设置读写信号）

cpu：cpu.vhd （设置CPU组成，各个状态下的动作等）

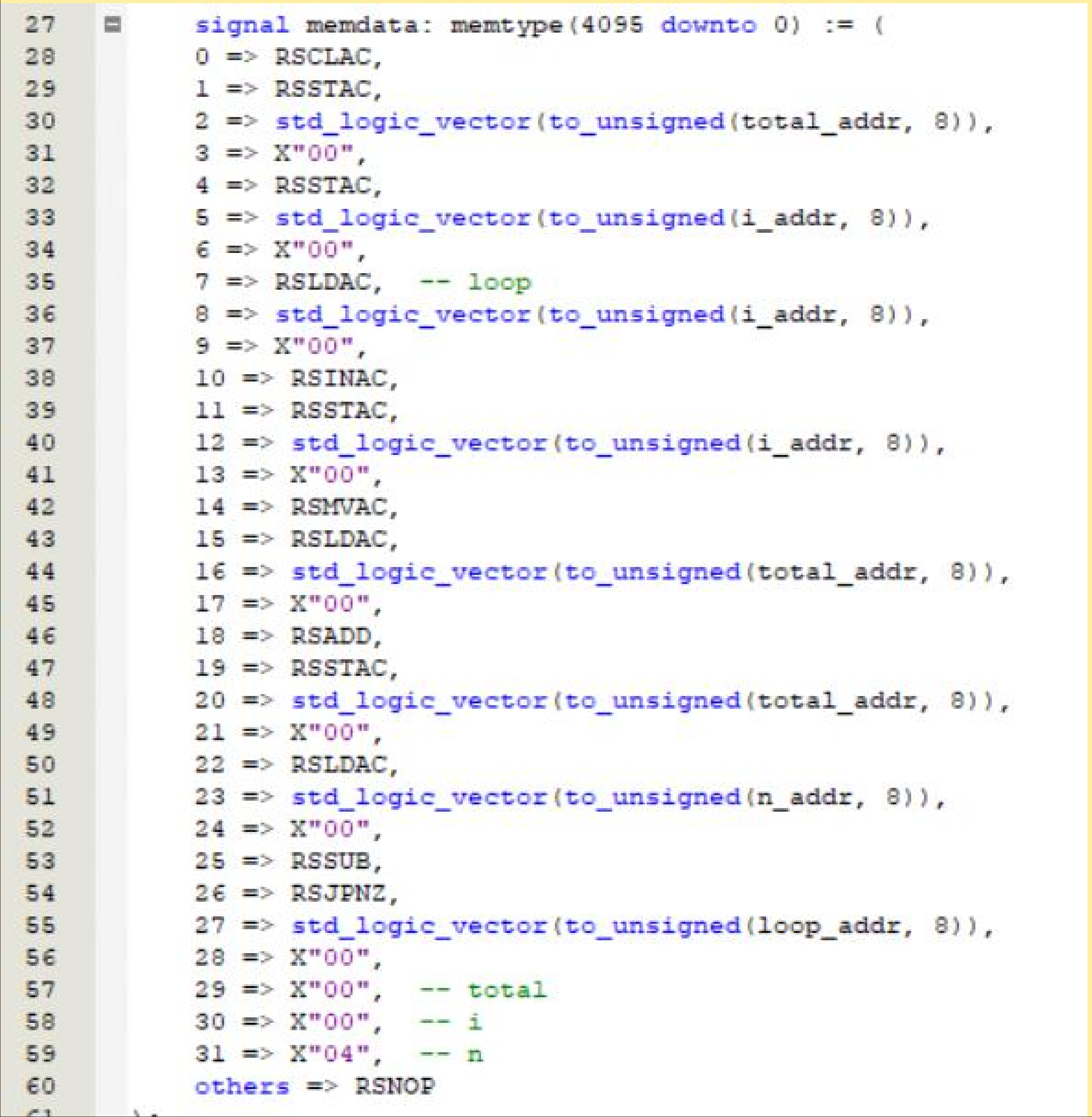
代码解析



```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


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";

end package;
```

rsisa.vhd是指定每个指令的对应的变量名，这样就可以用通俗易懂的变量名代替“XXXXXXXX”了。

代码解析



```vhdl
    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,
    19 => RSSTAC,
    20 => std_logic_vector(to_unsigned(total_addr, 8)),
    21 => X"00",
    22 => RSLDAC,
    23 => std_logic_vector(to_unsigned(n_addr, 8)),
    24 => X"00",
    25 => RSSUB,
    26 => RSJPNZ,
    27 => std_logic_vector(to_unsigned(loop_addr, 8)),
    28 => X"00",
    29 => X"00",   -- total
    30 => X"00",   -- i
    31 => X"04",   -- n
    others => RSNOP
```

memory的关键就是指定内存位置存储的指令。

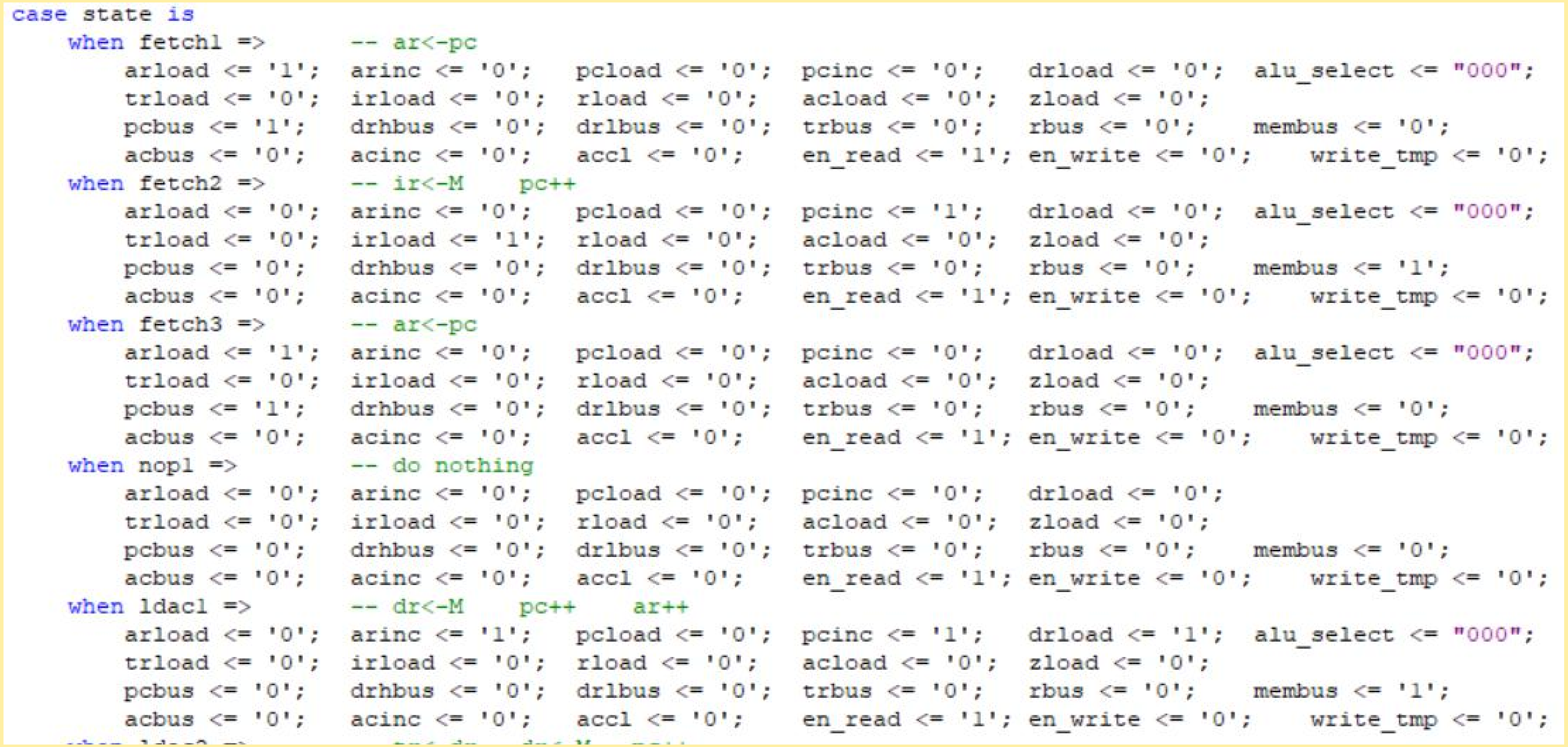
代码解析

在cpu.vhd中主要就是构造结构。例如clk信号，pc，ac寄存器等。address and data bus等bus的赋值，例如main_bus的赋值操作：

```
main_bus <= pc when pcbus='1' else
            (dr & tr) when (drhbus='1' and trbus='1') else
            (X"00" & dr) when drlbus='1' else
            (X"00" & r) when rbus='1' else
            (X"00" & ac) when acbus='1' else
            (X"00" & databus) when membus='1' else
            "ZZZZZZZZZZZZZZZZ";
```

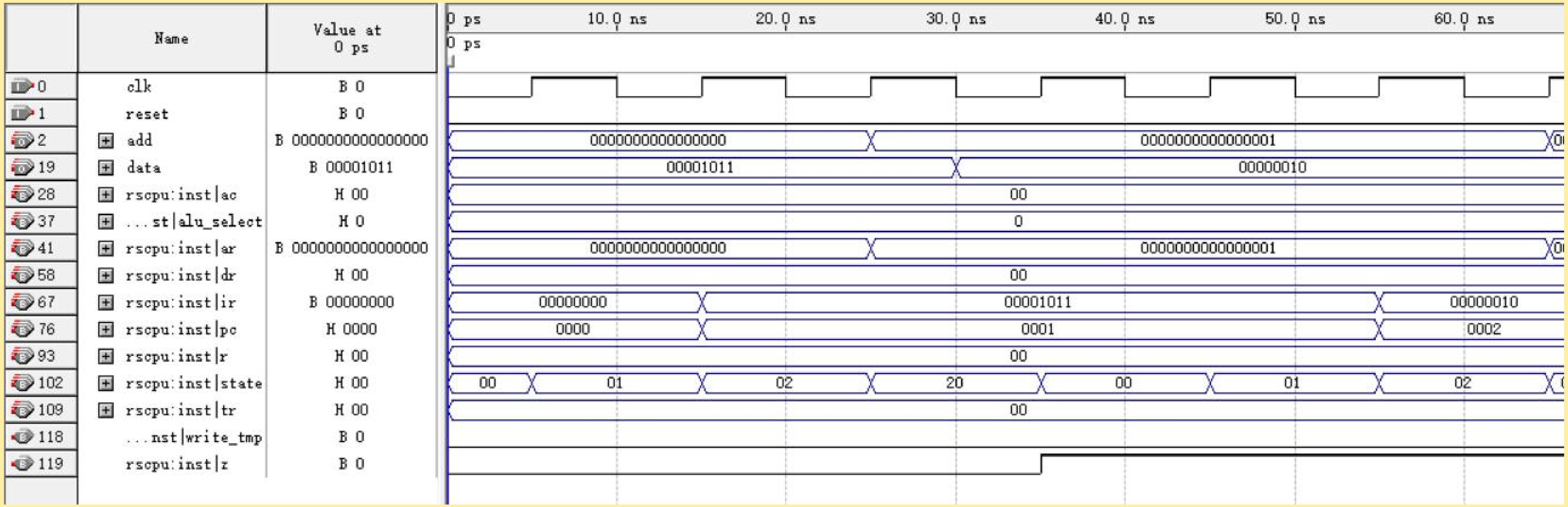
代码解析

各个状态下应该执行的操作:



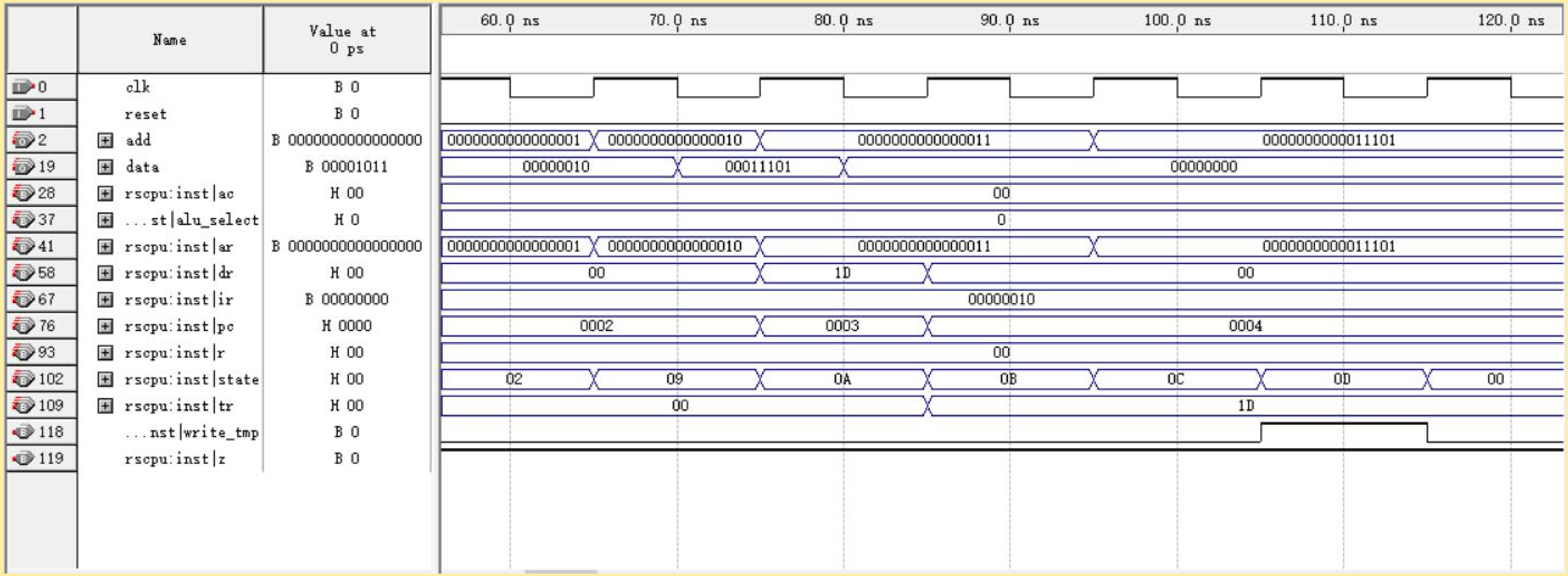
```vhdl
case state is
    when fetch1 =>       -- ar<-pc
        arload <= '1';  arinc <= '0';   pcload <= '0';  pcinc <= '0';   drload <= '0';  alu_select <= "000";
        trload <= '0';  irload <= '0';  rload <= '0';   acload <= '0';  zload <= '0';
        pcbus <= '1';   drhbus <= '0';  drlbus <= '0';  trbus <= '0';   rbus <= '0';    membus <= '0';
        acbus <= '0';   acinc <= '0';   accl <= '0';    en_read <= '1'; en_write <= '0';    write_tmp <= '0';
    when fetch2 =>       -- ir<-M    pc++
        arload <= '0';  arinc <= '0';   pcload <= '0';  pcinc <= '1';   drload <= '0';  alu_select <= "000";
        trload <= '0';  irload <= '1';  rload <= '0';   acload <= '0';  zload <= '0';
        pcbus <= '0';   drhbus <= '0';  drlbus <= '0';  trbus <= '0';   rbus <= '0';    membus <= '1';
        acbus <= '0';   acinc <= '0';   accl <= '0';    en_read <= '1'; en_write <= '0';    write_tmp <= '0';
    when fetch3 =>       -- ar<-pc
        arload <= '1';  arinc <= '0';   pcload <= '0';  pcinc <= '0';   drload <= '0';  alu_select <= "000";
        trload <= '0';  irload <= '0';  rload <= '0';   acload <= '0';  zload <= '0';
        pcbus <= '1';   drhbus <= '0';  drlbus <= '0';  trbus <= '0';   rbus <= '0';    membus <= '0';
        acbus <= '0';   acinc <= '0';   accl <= '0';    en_read <= '1'; en_write <= '0';    write_tmp <= '0';
    when nop1 =>         -- do nothing
        arload <= '0';  arinc <= '0';   pcload <= '0';  pcinc <= '0';   drload <= '0';
        trload <= '0';  irload <= '0';  rload <= '0';   acload <= '0';  zload <= '0';
        pcbus <= '0';   drhbus <= '0';  drlbus <= '0';  trbus <= '0';   rbus <= '0';    membus <= '0';
        acbus <= '0';   acinc <= '0';   accl <= '0';    en_read <= '1'; en_write <= '0';    write_tmp <= '0';
    when ldac1 =>        -- dr<-M    pc++    ar++
        arload <= '0';  arinc <= '1';   pcload <= '0';  pcinc <= '1';   drload <= '1';  alu_select <= "000";
        trload <= '0';  irload <= '0';  rload <= '0';   acload <= '0';  zload <= '0';
        pcbus <= '0';   drhbus <= '0';  drlbus <= '0';  trbus <= '0';   rbus <= '0';    membus <= '1';
        acbus <= '0';   acinc <= '0';   accl <= '0';    en_read <= '1'; en_write <= '0';    write_tmp <= '0';
```

仿真结果



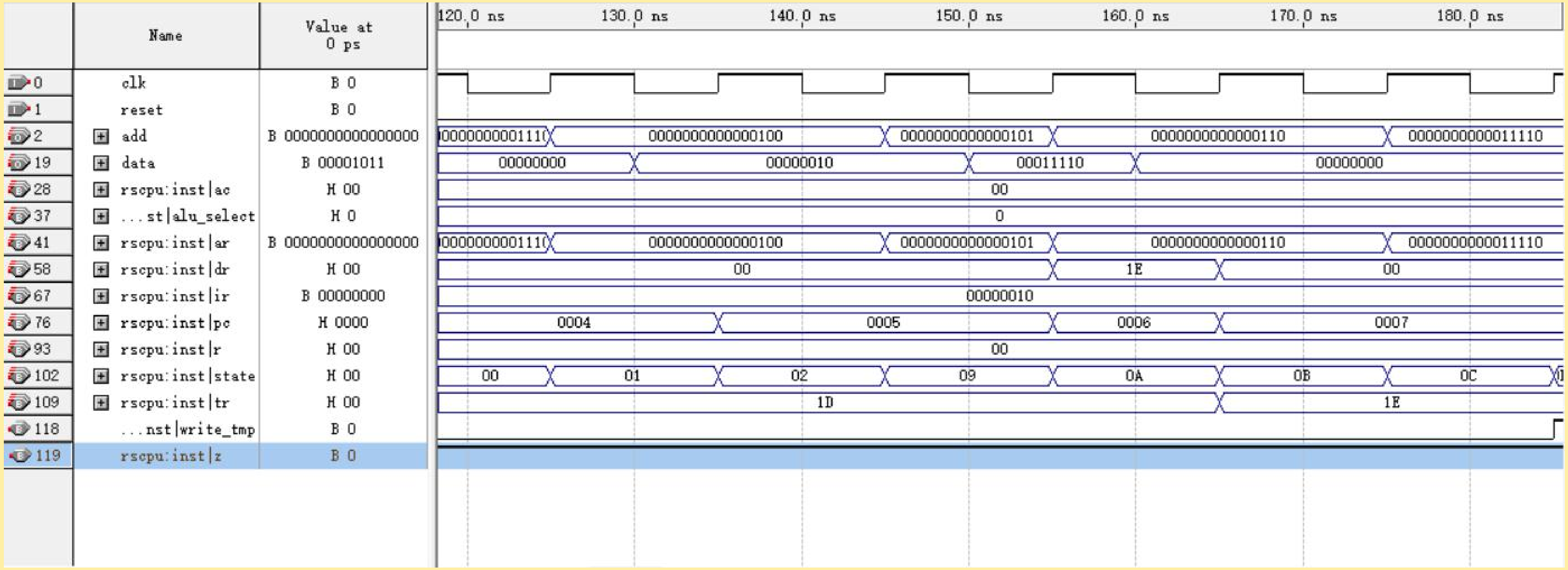
	Name	Value at 0 ps
0	clk	B 0
1	reset	B 0
2	add	B 0000000000000000
19	data	B 00001011
28	rscpu:inst|ac	H 00
37	...st|alu_select	H 0
41	rscpu:inst|ar	B 0000000000000000
58	rscpu:inst|dr	H 00
67	rscpu:inst|ir	B 00000000
76	rscpu:inst|pc	H 0000
93	rscpu:inst|r	H 00
102	rscpu:inst|state	H 00
109	rscpu:inst|tr	H 00
118	...nst|write_tmp	B 0
119	rscpu:inst|z	B 0

仿真结果



```
      Name                  Value at    60.0 ns   70.0 ns   80.0 ns   90.0 ns   100.0 ns   110.0 ns   120.0 ns
                            0 ps
0     clk                   B 0
1     reset                 B 0
2     add                   B 0000000000000000   0000000000000001 | 0000000000000010 | 0000000000000011 | 0000000000011101
19    data                  B 00001011           00000010 | 00011101 | 00000000
28    rscpu:inst|ac         H 00                 00
37    ...st|alu_select      H 0                  0
41    rscpu:inst|ar         B 0000000000000000   0000000000000001 | 0000000000000010 | 0000000000000011 | 0000000000011101
58    rscpu:inst|dr         H 00                 00 | 1D | 00
67    rscpu:inst|ir         B 00000000           00000010
76    rscpu:inst|pc         H 0000               0002 | 0003 | 0004
93    rscpu:inst|r          H 00                 00
102   rscpu:inst|state      H 00                 02 | 09 | 0A | 0B | 0C | 0D | 00
109   rscpu:inst|tr         H 00                 00 | 1D
118   ...nst|write_tmp      B 0
119   rscpu:inst|z          B 0
```

仿真结果



	Name	Value at 0 ps
0	clk	B 0
1	reset	B 0
2	add	B 0000000000000000
19	data	B 00001011
28	rscpu:inst|ac	H 00
37	...st|alu_select	H 0
41	rscpu:inst|ar	B 0000000000000000
58	rscpu:inst|dr	H 00
67	rscpu:inst|ir	B 00000000
76	rscpu:inst|pc	H 0000
93	rscpu:inst|r	H 00
102	rscpu:inst|state	H 00
109	rscpu:inst|tr	H 00
118	...nst|write_tmp	B 0
119	rscpu:inst|z	B 0

谢谢