现代计算机组成原理

潘 明　潘 松 编著

科学出版社

第 6 章

16位CISC CPU设计

6.1 顶层系统设计

6.1.1 16位CPU的组成结构

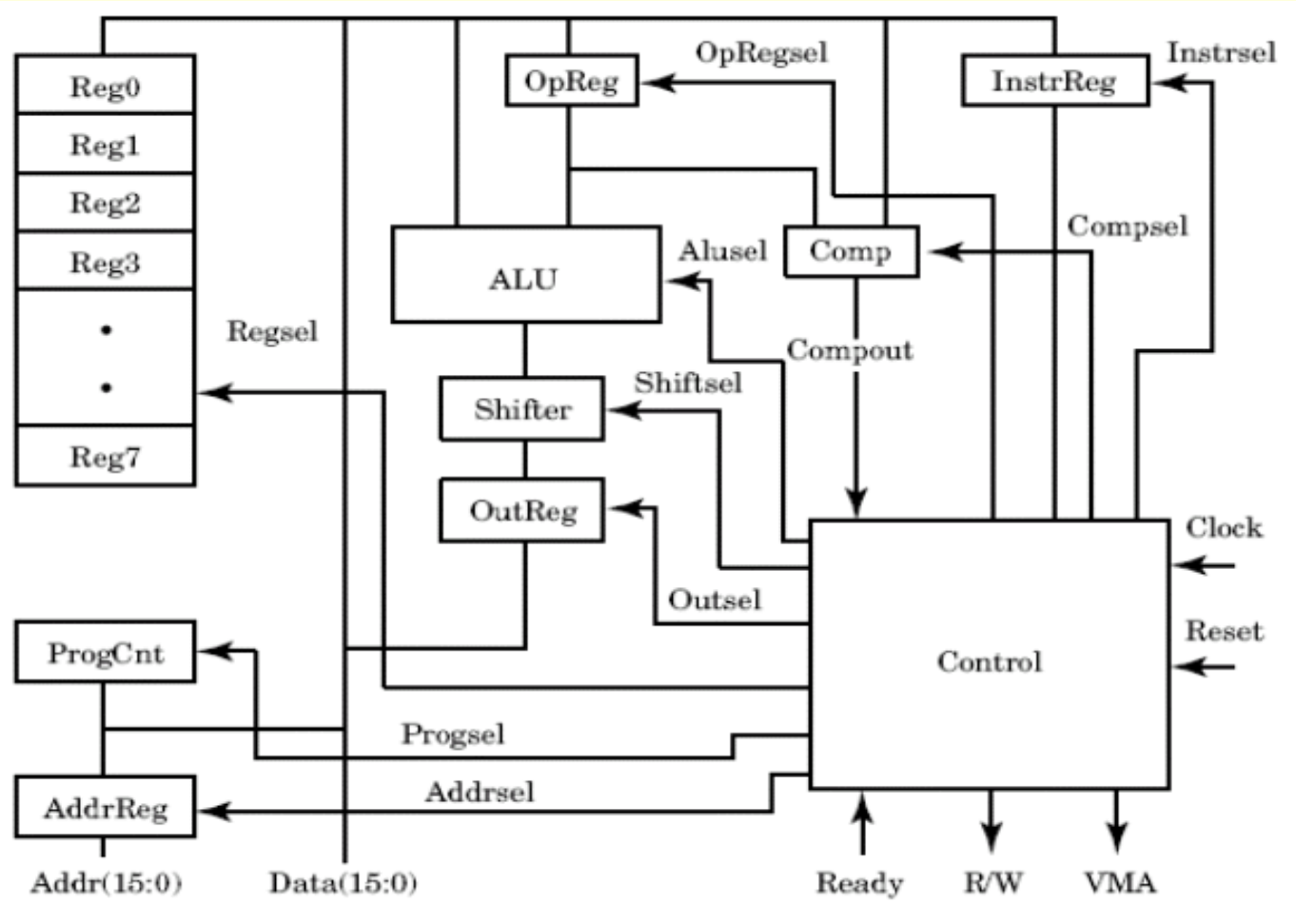

图6-1 16位CPU结构框图

6.1 顶层系统设计

6.1.2指令系统设计

1. 指令格式

（1）单字指令

表6-1　　单字节指令格式

操作码						源操作数			目的操作数		
Opcode						SRC			DST		
15	14	13	12	11		5	4	3	2	1	0

6.1 顶层系统设计

（2）双字指令

表6-2　　双字指令格式

操作码													目的操作数		
Opcode													DST		
15	14	13	12	11									2	1	0

16 位 操作数															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

表6-3　　双字节指令

操作码													目的操作数		
0	0	1	0	0									0	0	1

0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1
0				0				1				5			

2. 指令操作码　　表6-4　　操作码功能表

操作码	指令	功　　能
00000	NOP	空操作
00001	LOAD	装载数据到寄存器
00010	STORE	将寄存器的数据存入存储器
00011	MOVE	在寄存器之间传送操作数
00100	LOADI	将立即数装入寄存器
00101	BRANCHI	转移到由立即数指定的地址
00110	BRANCHGTI	大于时转移到由立即数指定的地址
00111	INC	加 1 指令
01000	DEC	减 1 指令
01001	AND	两个寄存器与操作
01010	OR	两个寄存器或操作
01011	XOR	两个寄存器异或操作
01100	NOT	寄存器求反
01101	ADD	两个寄存器加运算
01110	SUB	两个寄存器减运算
01111	ZERO	寄存器清零
10000	BRANCHLTI	小于时转移到由立即数指定的地址
10001	BRANCHLT	小于时转移
10010	BRANCHNEQ	等于时转移
10011	BRANCHEQI	转移到由立即数指定的地址
10100	BRANCHGT	大于时转移
10101	BRANCH	无条件转移
10110	BRANCHEQ	等于时转移
10111	BRANCHEQI	等于时转移到应即地址
11000	BRANCHLTEI	小于等于时转移到立即地址
11001	BRANCHLTE	小于等于时转移
11010	SHL	向左逻辑移位
11011	SHR	向右逻辑移位
11100	ROTR	循环右移
11101	ROTL	循环左移

6.1 顶层系统设计

6.1.2指令系统设计

2. 指令操作码

表6-5 常用指令举例

指令	机器码	字长	操作码 Opcode										源操作数 SRC			目的操作数 DST			功能说明
LOADI R1,0021H	2001H	2	0	0	1	0	0	x	x	x	x	x	x	x	x	0	0	1	立即数 0021H
	0021H		0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	送 R1
LOAD R3, [R1]	080BH	1	0	0	0	0	1	x	x	x	x	x	0	0	1	0	1	1	从R1指定的存储单元取数送 R3
STORE [R2], R3	101AH	1	0	0	0	1	0	x	x	x	x	x	0	1	1	0	1	0	将 R3 的内容存入 R2 指定单元
BRANCHGTI [0000]	300EH	2	0	0	1	1	0	x	x	x	x	x	0	0	1	1	1	0	若 R1>R6，则转向地址[0000]
	0000H		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
INC R2	3802H	1	0	0	1	1	1	x	x	x	x	x	x	x	x	0	1	0	修改目的指针 R2<=R2+1
BRANCHI [0006]	2800H	2	0	0	1	0	1	x	x	x	x	x	x	x	x	x	x	x	绝对地址转移指令，goto [0006],
	0006H		0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	

6.1 顶层系统设计

6.1.3 顶层结构的VHDL设计

1. CPU元件的VHDL描述

【例6-1】CPU_LIB.VHD

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
package cpu_lib is
type t_shift is (shftpass, shl, shr, rotl, rotr);
subtype t_alu is unsigned(3 downto 0);
```

（接下页）

```
    constant alupass : unsigned(3 downto 0) := "0000";
    constant andOp : unsigned(3 downto 0) := "0001";
    constant orOp : unsigned(3 downto 0) := "0010";
    constant notOp : unsigned(3 downto 0) := "0011";
    constant xorOp : unsigned(3 downto 0) := "0100";
    constant plus : unsigned(3 downto 0) := "0101";
    constant alusub : unsigned(3 downto 0) := "0110";
    constant inc : unsigned(3 downto 0) := "0111";
    constant dec : unsigned(3 downto 0) := "1000";
    constant zero : unsigned(3 downto 0) := "1001";
    type t_comp is (eq, neq, gt, gte, lt, lte);
    subtype t_reg is std_logic_vector(2 downto 0);
    type state is (reset1, reset2, reset3, reset4, reset5,reset6, execute, nop, load, store,
move,
    load2, load3, load4, store2, store3,store4, move2, move3, move4,incPc, incPc2,
    incPc3, incPc4, incPc5, incPc6, loadPc,loadPc2,loadPc3, loadPc4, bgtI2, bgtI3,
    bgtI4, bgtI5, bgtI6, bgtI7,bgtI8, bgtI9,bgtI10, braI2, braI3, braI4, braI5, braI6,
    loadI2,loadI3, loadI4, loadI5, loadI6,inc2, inc3, inc4);
    subtype bit16 is std_logic_vector(15 downto 0);
    end cpu_lib;
```

【例6-2】top.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use work.cpu_lib.all;
entity top is
end top;
architecture behave of top is
component mem
port (addr : in bit16；  sel,rw : in std_logic；  ready : out std_logic；
data : inout bit16)；
end component;
component cpu
port(clock, reset, ready : in std_logic；  addr : out bit16;
rw, vma : out std_logic；  data : inout bit16)；
end component;
signal addr, data : bit16 ；  signal vma, rw, ready : std_logic；
signal clock, reset : std_logic := '0'；
begin
clock <= not clock after 50 ns ；  reset <= '1', '0' after 100 ns；
m1 : mem port map (addr, vma, rw, ready, data)；
u1 : cpu port map(clock, reset, ready, addr, rw, vma,data);
end behave;
```

2. 顶层文件的原理图设计

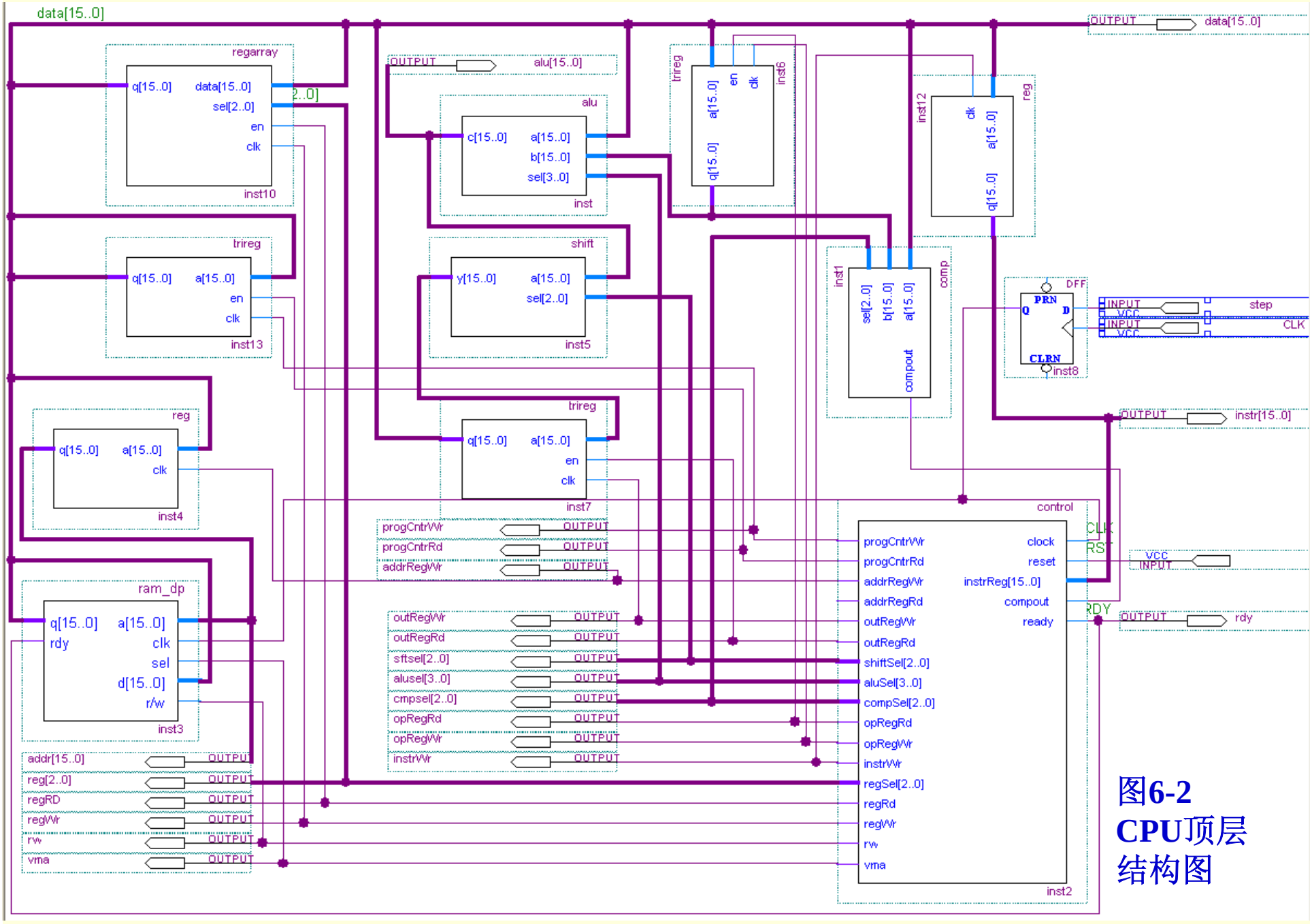

图6-2
CPU顶层
结构图

6.1 顶层系统设计

6.1.3 顶层结构的VHDL设计

3. CPU与LCD显示模块的接口

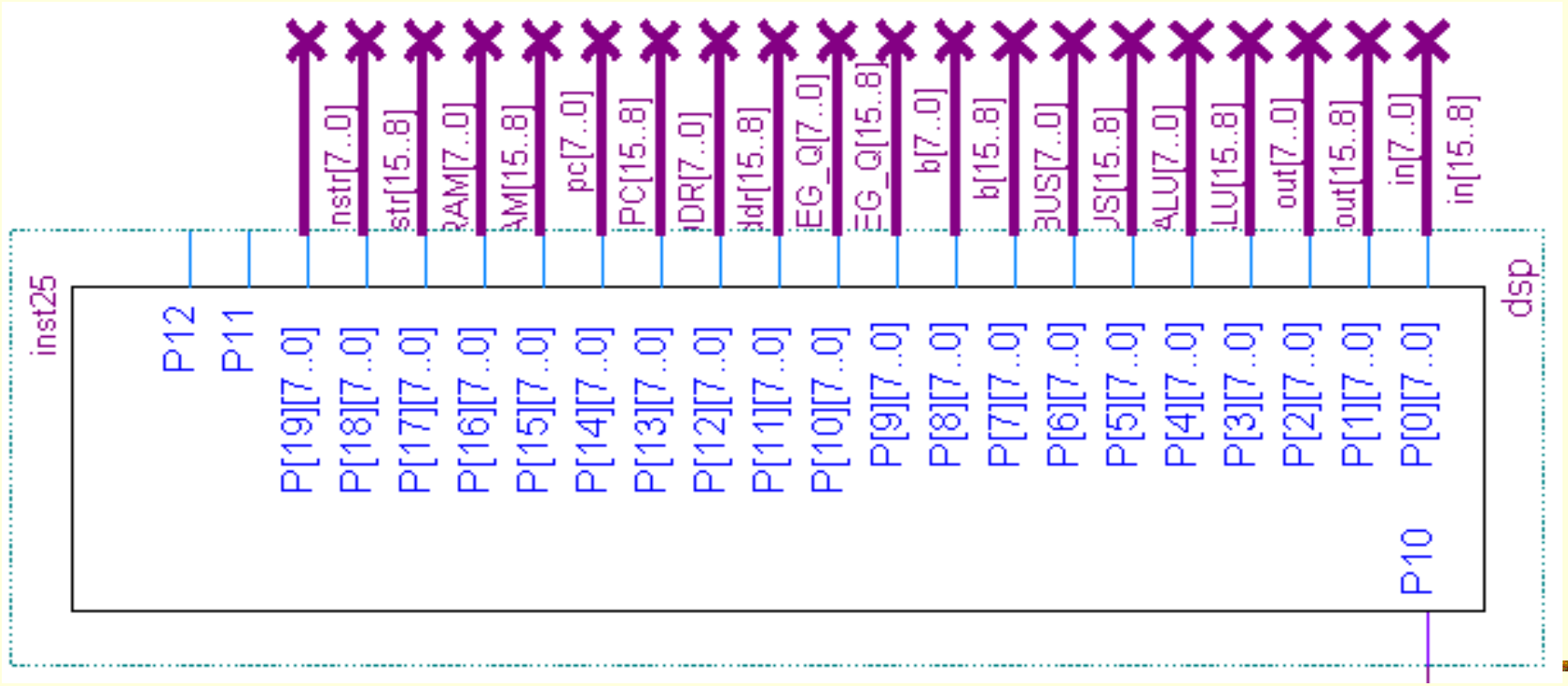

图6-3 显示模块dsp的实体结构图

6.1 顶层系统设计

6.1.3 顶层结构的VHDL设计

3. CPU与LCD显示模块的接口

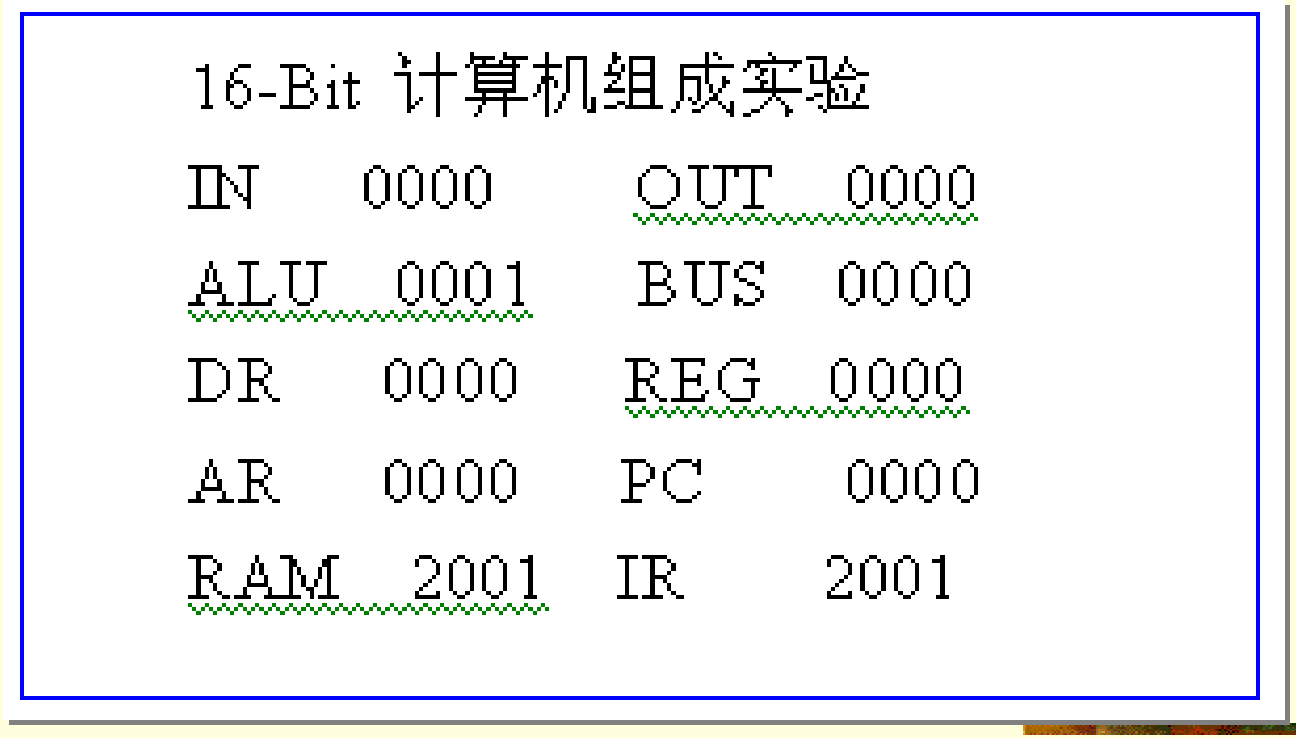

图6-4 LCD显示屏的数据显示

6.1 顶层系统设计

6.1.4 软件设计实例

表6-6　　示例程序

地址	机器码	指令	功能说明
0000H 0001H	2001H 0010H	LOADI··R1,0010H	源操作数首地址（0010）送 REG1
0002H 0003H	2002H 0030H	LOADI··R2,0030H	目的操作数首地址（0030）送 REG2
0004H 0005H	2006H 002FH	LOADI··R6,002FH	结束地址（002F）送 REG6
0006H	080BH	LOAD··[R1]	取数
0007H	101AH	STORE··[R2]	存数
0008H 0009H	300EH 0000H	BRANCHGTI··[0000]	比较·R1>R6·？，若 yes，则转向地址[0000]
000AH	3801H	INC··R1	-修改源指针·R1<=R1+1
000BH	3802H	INC··R2	修改目的指针 R2<=R2+1
000CH	2800H 0006H	BRANCHI··[0006] ····	这是一个绝对地址转移指令，goto·到地址[0006]，执行此处指令，实现循环。

表6-7　　存储器初始化文件RAM_16.mif的内容

```
WIDTH = 16;
DEPTH = 256;
ADDRESS_RADIX = HEX;
DATA_RADIX = HEX;
CONTENT BEGIN
0       :       2001;--源操作数（10）送REG1
1       :       0010;--
2       :       2002;--目的操作数（30）送REG2
3       :       0030;--
4       :       2006;--结束地址（2f）送REG6
5       :       002f;--
6       :       080b; --取数
7       :       101a;-- 存数
8       :       300e;--比较 R1>R6 ?
9       :       0000;-- 若yes,则停止
a       :       3801;--修改源指针 R1<=R1+1
b       :       3802;--修改目的指针R2<=R2+1
c       :       280d;-- goto [06]单元地址，循环
d       :       0006;--
e       :       0000;
;  -----地址10H—2fH单元为数据块
f       :       0001;
10      :       0002;
11      :       0003;
12      :       0004;
13      :       0005;
14      :       0006;
15      :       0007;
16      :       0008;
17      :       0009;
18      :       000a;
19      :       000b;
1a      :       000c;
1b      :       000d;
1c      :       000e;
1d      :       000f;
1e      :       0010;
1f      :       0011;
20..7F::        0000;
END;
```

6.2 CPU基本部件设计

6.2.1 运算器ALU

表6-8　　运算器ALU的功能

Sel 输入	操作	说明
0000	C=A	通过PASS
0001	C=A AND B	与
0010	C=A OR B	或
0011	C=NOT A	非
0100	C=A XOR B	异或
0101	C=A + B	加法
0110	C=A – B	减法
0111	C=A + 1	加1
1000	C=A – 1	减1
1001	C=0	清0

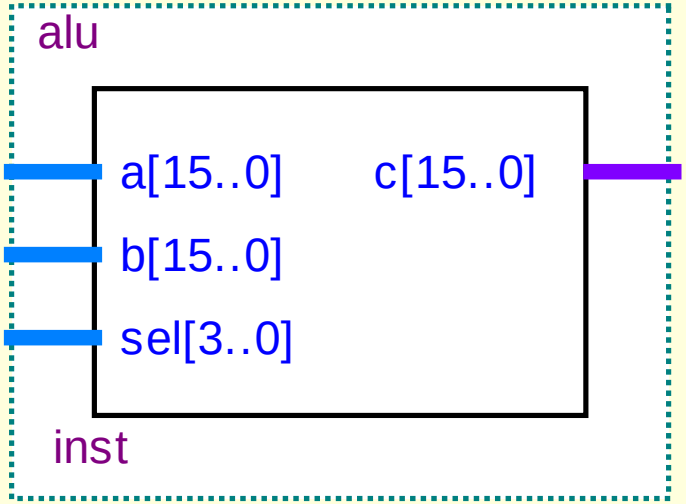

图6-5 运算器ALU结构图

【例6-3】alu.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use work.cpu_lib.all;
entity alu is
port( a, b : in bit16; sel : in t_alu; c : out bit16 );
end alu;
architecture rtl of alu is
begin
aluproc: process(a, b, sel)
begin
case sel is
when alupass=> c<=a after 1 ns;  when andOp => c<=a and b after 1 ns;
when orOp => c<= a or b after 1 ns; when xorOp => c<= a xor b after 1 ns;
when notOp => c<= not a after 1 ns; when plus => c<= a + b after 1 ns;
when alusub => c<= a - b after 1 ns;
when inc => c<= a + "0000000000000001" after 1 ns;
when dec => c<= a - "0000000000000001" after 1 ns;
when zero => c<= "0000000000000000" after 1 ns;
when others => c<= "0000000000000000" after 1 ns;
end case;end process;
end rtl;
```

6.2 CPU基本部件设计

6.2.1 运算器ALU

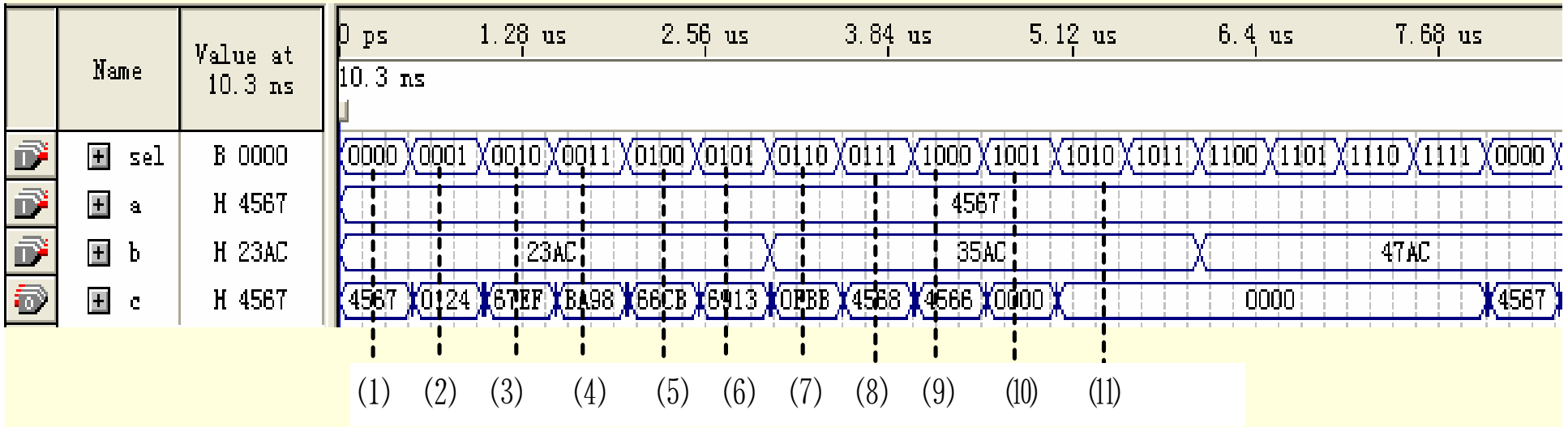

图6-6 运算器ALU的仿真波形

6.2 CPU基本部件设计

6.2.1 运算器ALU

表6-9 ALU运算仿真结果说明

工作点波形	功能选择 sel	运算类型	输入数据		运算结果 c
			a	b	
（1）	0000	通过PASS	4567		C=A , C=4567
（2）	0001	与	4567	23AC	C=A AND B= 4567∧23AC = 0124
（3）	0010	或	4567	23AC	C=A OR B = 4567∨23AC = 67EF
（4）	0011	非	4567		C=NOT A = BA98
（5）	0100	异或	45467	23AC	C=A XOR B = A⊕B = 4567⊕23AC = 66CB
（6）	0101	加法	4567	23AC	C=A + B= 4567+23AC= 6913
（7）	0110	减法	4567	35AC	C=A – B = 4567-35AC =0FBB
（8）	0111	加1	4567		C=A + 1 = 4567+1= 4568
（9）	1000	减1	4567		C=A – 1 = 4567-1 =4566
（10）	1001	清0	xxxx	xxxx	C=0000
（11）	1010~1111	其它	xxxx	xxxx	C=0000

6.2 CPU基本部件设计

6.2.2 比较器COMP

表6-10　　比较器的运算类型

t_comp	比较类型	操作说明
000	eq (等于)	若a=b，compout=1
001	neq (不等于)	若a<>b，compout=1
010	gt (大于)	若a>b，compout=1
011	gte (大于等于)	若a>=b，compout=1
100	lt (小于)	若a<b，compout=1
101	lte (小于等于)	若a<=b，compout=1
其他		compout=0

6.2 CPU基本部件设计

6.2.2 比较器COMP

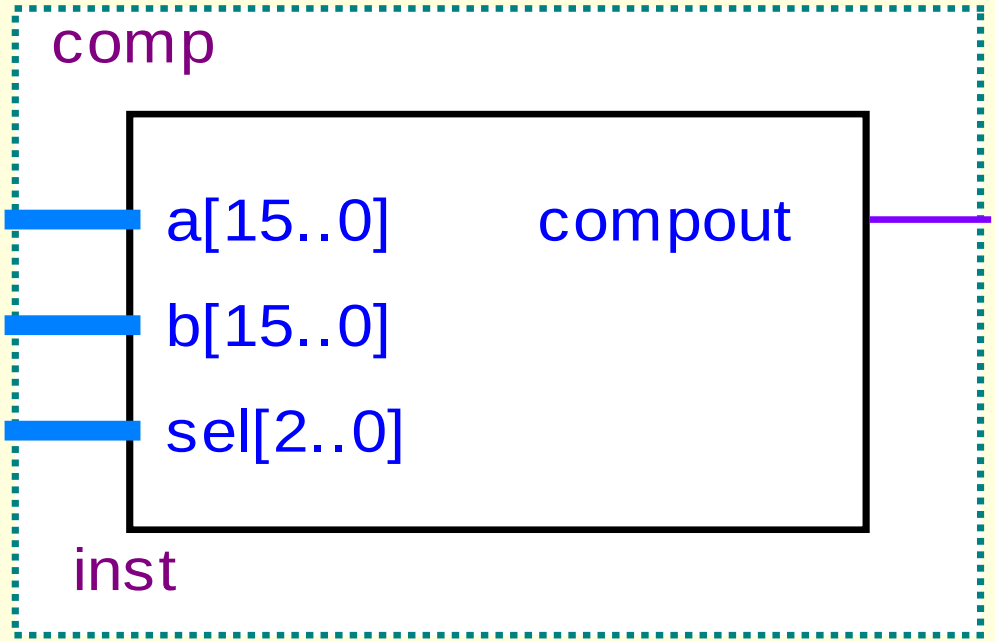

图6-7比较器结构图

【例6-4】COMP.VHD

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.cpu_lib.all;
entity comp is
port( a, b : in bit16；  sel : in t_comp；  compout : out std_logic)；
end comp;
architecture rtl of comp is
begin
compproc: process(a, b, sel)
begin
case sel is
when eq => if a = b then compout <= '1' after 1 ns ；
else compout <='0' after 1 ns ；  end if ；
when neq => if a /= b then compout <= '1' after 1 ns;
else compout <= '0' after 1 ns ；  end if；
when gt => if a > b then compout <= '1' after 1 ns;
else compout <= '0' after 1 ns ；  end if；
when gte => if a >= b then compout <= '1'after 1 ns；
else compout <= '0' after 1 ns ；  end if;
when lt => if a < b then compout <= '1' after 1 ns ；
else compout <= '0' after 1 ns ；  end if;
when lte => if a <= b then compout <= '1' after 1 ns ；
else compout <= '0' after 1 ns ；  end if;
end case ；
end process；
end rtl；
```

6.2 CPU基本部件设计

6.2.2 比较器COMP

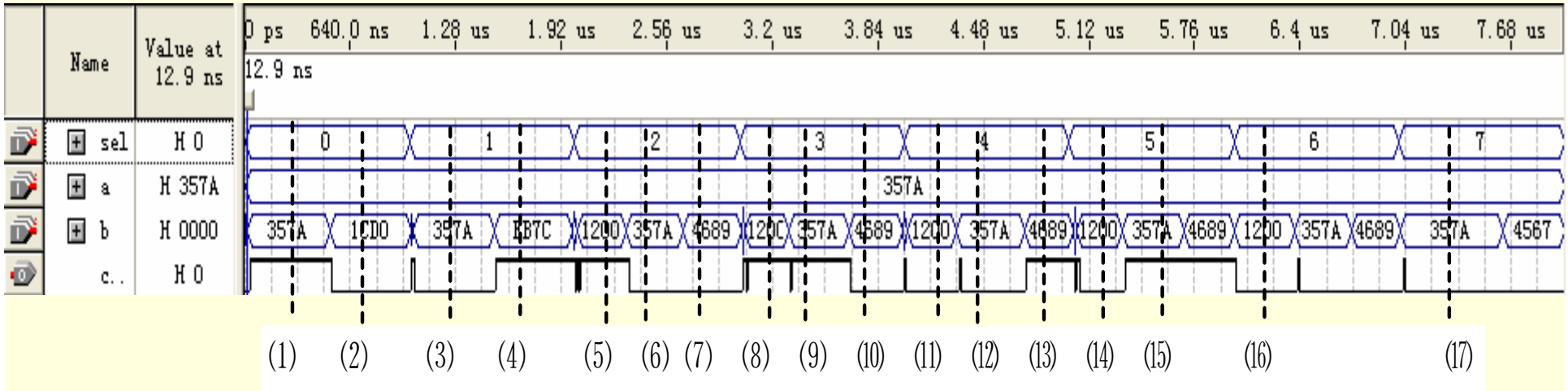

图6-8 比较器COMP的仿真波形图

表6-11　　比较器COMP的仿真波形说明

工作点波形	功能选择 sel	比较类型 t_comp	输入数据		比较运算结果 compout
			a	b	
（1）	0	等于	357A	357A	∵a=b，∴compout=1
（2）	0	等于	357A	1CD0	∵a>b，∴compout=0
（3）	1	不等于	357A	357A	∵a=b，∴compout=0
（4）	1	不等于	357A	EB7C	∵a<>b，∴ compout=1
（5）	2	大于	357A	1200	∵a>b，∴compout=1
（6）	2	大于	357A	357A	∵a=b，∴compout=0
（7）	2	大于	357A	4689	∵a<b，∴compout=0
（8）	3	大于等于	357A	1200	∵a>b，∴compout=1
（9）	3	大于等于	357A	357A	∵a=b，∴compout=1
（10）	3	大于等于	357A	4689	∵a<b，∴compout=0
（11）	4	小于	357A	1200	∵a>b，∴compout=0
（12）	4	小于	357A	357A	∵a=b，∴compout=0
（13）	4	小于	357A	4689	∵a<b，∴compout=1
（14）	5	小于等于	357A	1200	∵a>b，∴compout=0
（15）	5	小于等于	357A	357A	∵a=b，∴compout=1
（16）~(17)	6~7	其他	xxxx	xxxx	compout=0

6.2 CPU基本部件设计

6.2.3 控制器CONTROL

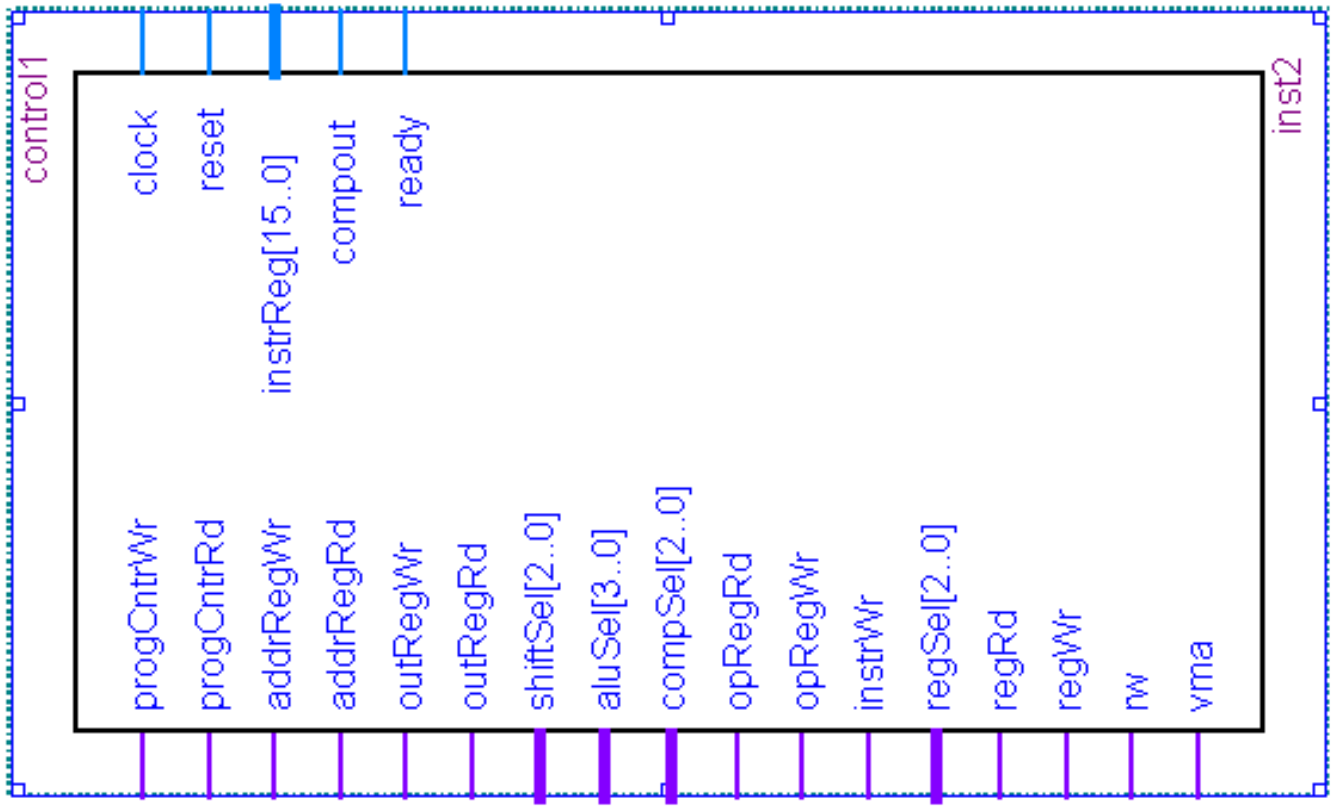

图6-9 控制器CONTROL的实体结构图

【例6-5】control.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use work.cpu_lib.all;
entity control is
port( clock，reset，ready，compout: in  std_logic; instrReg : in  bit16;
progCntrWr，progCntrRd，addrRegWr，addrRegRd，outRegWr，
outRegRd : out  std_logic;
shiftSel : out  t_shift; aluSel : out  t_alu; compSel : out  t_comp;
opRegRd，opRegWr，instrWr，regRd，regWr，rw，vma: out std_logic;
regSel : out  t_reg );
end control;
architecture rtl of control is
signal current_state, next_state : state;
begin
nxtstateproc: process( current_state, instrReg, compout,ready)
begin
progCntrWr <= '0'; progCntrRd <= '0'; addrRegWr <= '0'; outRegWr <= '0';
outRegRd <= '0'; shiftSel <= shftpass; aluSel <= alupass; compSel <= eq;
opRegRd <= '0'; opRegWr <= '0'; instrWr <= '0'; regSel <= "000";
regRd <= '0'; regWr <= '0'; rw <= '0'; vma <= '0';
case current_state is
when reset1=> aluSel<=zero after 1 ns; shiftSel<=shftpass; next_state<=reset2;
when reset2 => aluSel<=zero; shiftSel<=shftpass; outRegWr<='1';
```

（接下页）

```
                next_state<=reset3;
                when reset3 => outRegRd<='1'; next_state<=reset4;
                when reset4 => outRegRd<='1'; addrRegRd<='1'; progCntrWr<='1';
                addrRegWr<='1'; next_state<=reset5;
                when reset5 => vma<='1'; rw <= '0'; next_state <= reset6;
                when reset6 => vma<='1'; rw<='0';
                if ready = '1' then instrWr<='1'; next_state<=execute;
                else next_state <= reset6; end if;
                when execute => case instrReg(15 downto 11) is
                when "00000" => next_state <= incPc ;-- nop
                when "00001" => regSel<=instrReg(5 downto 3); regRd<='1';
next_state<=load2;
                when "00010" => regSel<=instrReg(2 downto 0); regRd<='1';
                 next_state<=store2;-- store
                when "00011" => regSel<=instrReg(5 downto 3); regRd<='1'; aluSel<=alupass;
                shiftSel<=shftpass;  next_state<=move2;
                when "00100" => progcntrRd<='1'; alusel<=inc; shiftsel<=shftpass;
                 next_state<=loadI2;
                when "00101" => progcntrRd<='1'; alusel<=inc; shiftsel<=shftpass;
                next_state<=braI2;
                when "00110" => regSel<=instrReg(5 downto 3); regRd<='1';
                next_state<=bgtI2;--BranchGTImm
                when "00111" => regSel<=instrReg(2 downto 0); regRd<='1'; alusel<=inc;
                shiftsel<=shftpass; next_state<=inc2;
                when others =>next  state <= incPc;
```

（接下页）

```
            end case;
            when load2 => regSel <= instrReg(5 downto 3); regRd <= '1';
            addrregWr <= '1'; next_state <= load3;
            when load3 => vma <= '1'; rw <= '0'; next_state <= load4;
            when load4 => vma <= '1'; rw <= '0'; regSel <= instrReg(2 downto 0);
            regWr <= '1'; next_state <= incPc;
            when store2 => regSel <= instrReg(2 downto 0); regRd <= '1';
            addrregWr <= '1'; next_state <= store3;
            when store3 => regSel <= instrReg(5 downto 3); regRd <= '1';
             next_state <= store4;
            when store4 => regSel <= instrReg(5 downto 3); regRd <= '1'; vma <= '1';
            rw <= '1'; next_state <= incPc;
            when move2 => regSel <= instrReg(5 downto 3); regRd <= '1';aluSel <=
alupass;
            shiftsel <= shftpass; outRegWr <= '1'; next_state <= move3;
            when move3 => outRegRd <= '1'; next_state <= move4;
            when move4 =>outRegRd <= '1';
            regSel <= instrReg(2 downto 0); regWr <= '1'; next_state <= incPc;
            when loadI2 => progcntrRd <= '1'; alusel <= inc; shiftsel <= shftpass;
            outregWr <= '1'; next_state <= loadI3;
            when loadI3 => outregRd <= '1'; next_state <= loadI4;
            when loadI4 => outregRd <= '1'; progcntrWr<='1'; addrregWr<='1';
            next_state<=loadI5;
            when loadI5 => vma <= '1'; rw <= '0'; next_state <= loadI6;
            when loadI6 => vma <= '1'; rw <= '0';
```

（接下页）

```
if ready = '1' then regSel <= instrReg(2 downto 0);
regWr <= '1'; next_state <= incPc;
else next_state <= loadI6; end if;
when braI2 => progcntrRd <= '1'; alusel <= inc; shiftsel <= shftpass;
outregWr <= '1'; next_state <= braI3;
when braI3 => outregRd <= '1'; next_state <= braI4;
when braI4 => outregRd<='1'; progcntrWr<='1'; addrregWr<='1';
next_state<=braI5;
when braI5 => vma<='1'; rw<='0'; next_state <= braI6;
when braI6 => vma <= '1'; rw <= '0';
if ready = '1' then progcntrWr <= '1'; next_state <= loadPc;
else next_state <= braI6; end if;
when bgtI2 => regSel <= instrReg(5 downto 3); regRd <= '1';
opRegWr <= '1'; next_state <= bgtI3;
when bgtI3 => opRegRd <= '1'; regSel <= instrReg(2 downto 0);
regRd <= '1'; compsel <= gt; next_state <= bgtI4;
when bgtI4 => opRegRd <= '1' after 1 ns;
regSel <= instrReg(2 downto 0); regRd <= '1'; compsel <= gt;
if compout = '1' then next_state <= bgtI5;
else next_state <= incPc; end if;
when bgtI5 => progcntrRd<='1'; alusel<=inc; shiftSel<=shftpass;
next_state<=bgtI6;
when bgtI6 => progcntrRd <= '1'; alusel <= inc; shiftsel <= shftpass;
outregWr <= '1'; next_state <= bgtI7;
```

（接下页）

```
when bgtI7 => outregRd <= '1'; next_state <= bgtI8;
when bgtI8 =>outregRd <= '1';
progcntrWr <= '1'; addrregWr <= '1'; next_state <= bgtI9;
when bgtI9 => vma <= '1'; rw <= '0'; next_state <= bgtI10;
when bgtI10 => vma <= '1'; rw <= '0';
if ready = '1' then progcntrWr <= '1'; next_state <= loadPc;
else next_state <= bgtI10; end if;
when inc2 => regSel <= instrReg(2 downto 0); regRd <= '1'; alusel <= inc;
shiftsel <= shftpass; outregWr <= '1'; next_state <= inc3;
when inc3 => outregRd <= '1'; next_state <= inc4;
when inc4 => outregRd <= '1';  regsel <= instrReg(2 downto 0);
regWr <= '1'; next_state <= incPc;
when loadPc => progcntrRd <= '1'; next_state <= loadPc2;
when loadPc2 => progcntrRd <= '1'; addrRegWr <= '1'; next_state <= loadPc3;
when loadPc3 => vma <= '1'; rw <= '0'; next_state <= loadPc4;
when loadPc4 => vma <= '1'; rw <= '0';
if ready = '1' then instrWr <= '1'; next_state <= execute;
else next_state <= loadPc4; end if;
when incPc => progcntrRd<='1'; alusel<=inc; shiftsel<=shftpass;
next_state<=incPc2;
when incPc2 => progcntrRd <= '1'; alusel <= inc; shiftsel <= shftpass;
outregWr <= '1'; next_state <= incPc3;
when incPc3 => outregRd <= '1'; next_state <= incPc4;
when incPc4 => outregRd<='1'; progcntrWr<='1';
```

（接下页）

```
addrregWr<='1'; next_state<=incPc5;
when incPc5 => vma <= '1'; rw <= '0'; next_state <= incPc6;
when incPc6 => vma <= '1'; rw <= '0';
if ready = '1' then instrWr <= '1'; next_state <= execute;
else next_state <= incPc6; end if;
when others => next_state <= incPc;
end case;
end process;
controlffProc:process(clock, reset)
begin
if reset = '1' then current_state <= reset1 after 1 ns;
elsif clock'event and clock = '1'
then current_state <= next_state after 1 ns; end if;
end process;
end rtl;
```

6.2 CPU基本部件设计

6.2.4 寄存器与寄存器阵列

1. 寄存器REG

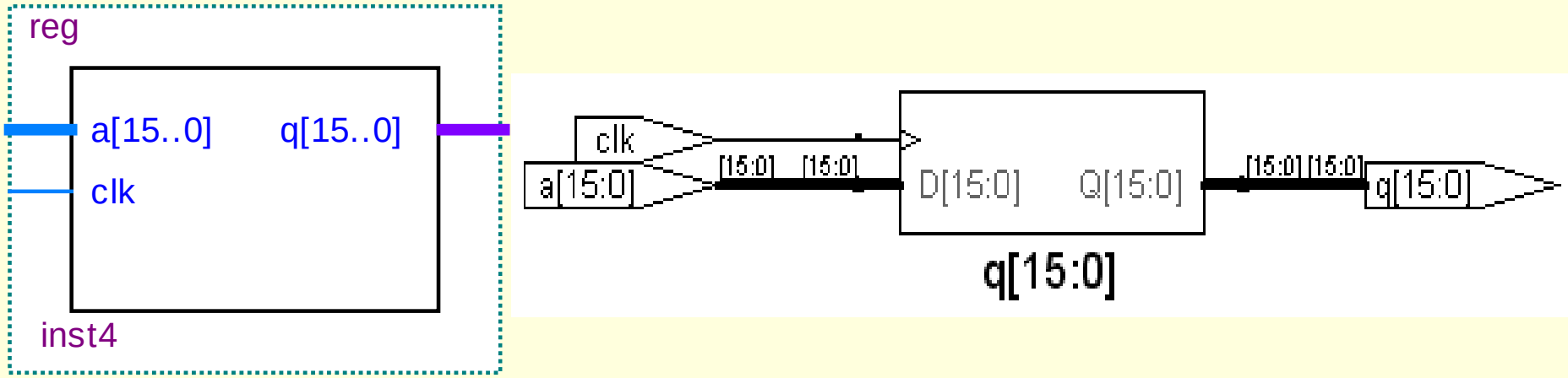

图6-10 寄存器REG的实体结构和RTL图

6.2 CPU基本部件设计

6.2.4 寄存器与寄存器阵列

1. 寄存器REG

【例6-6】reg.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use work.cpu_lib.all;
entity reg is
port( a : in bit16; clk : in std_logic; q : out bit16);
end reg;
architecture rtl of reg is
begin
regproc: process
begin
wait until clk' event and clk = '1';
q <= a after 1 ns;
end process;
end rtl;
```

【例6-7】regarray.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use work.cpu_lib.all;
entity regarray is
port( data : in bit16; sel : in t_reg; en，clk : in std_logic;
q : out bit16);
end regarray;
architecture rtl of regarray is
type t_ram is array (0 to 7) of bit16;
signal temp_data : bit16;
begin
process(clk,sel)
variable ramdata : t_ram;
begin
if clk'event and clk = '1' then ramdata(conv_integer(sel)) := data;
end if;
temp_data <= ramdata(conv_integer(sel)) after 1 ns;
end process;
process(en, temp_data)
begin
if en = '1' then q <= temp_data after 1 ns;
else q <="ZZZZZZZZZZZZZZZZ" after 1 ns; end if;
end process;
end rtl;
```

6.2 CPU基本部件设计

6.2.4 寄存器与寄存器阵列

2. 寄存器阵列RegArray

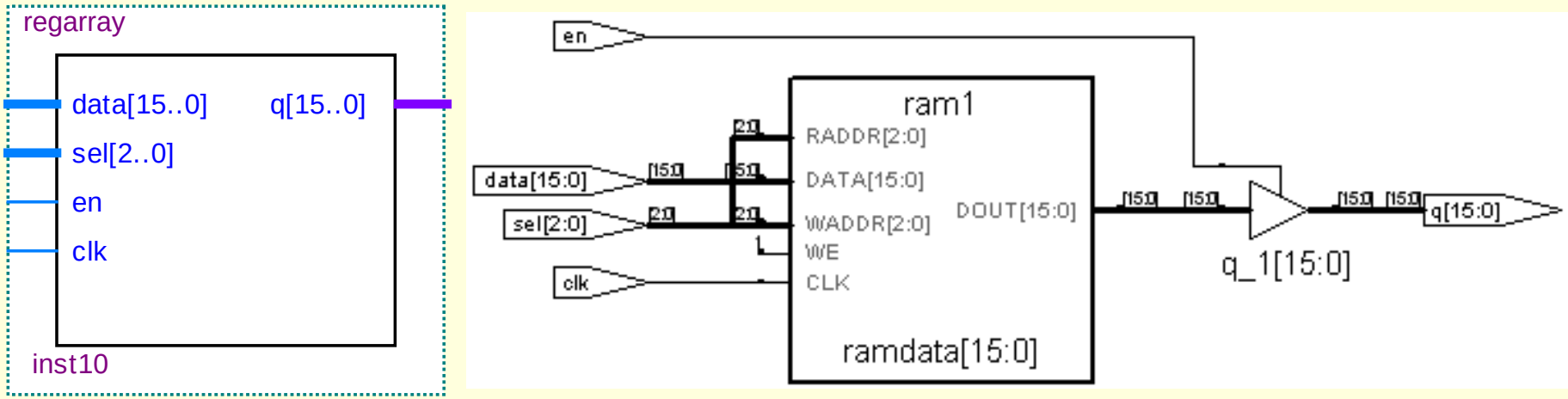

图6-11 寄存器阵列RegAarray的结构图和RTL图

6.2 CPU基本部件设计

6.2.4 寄存器与寄存器阵列

2. 寄存器阵列RegArray

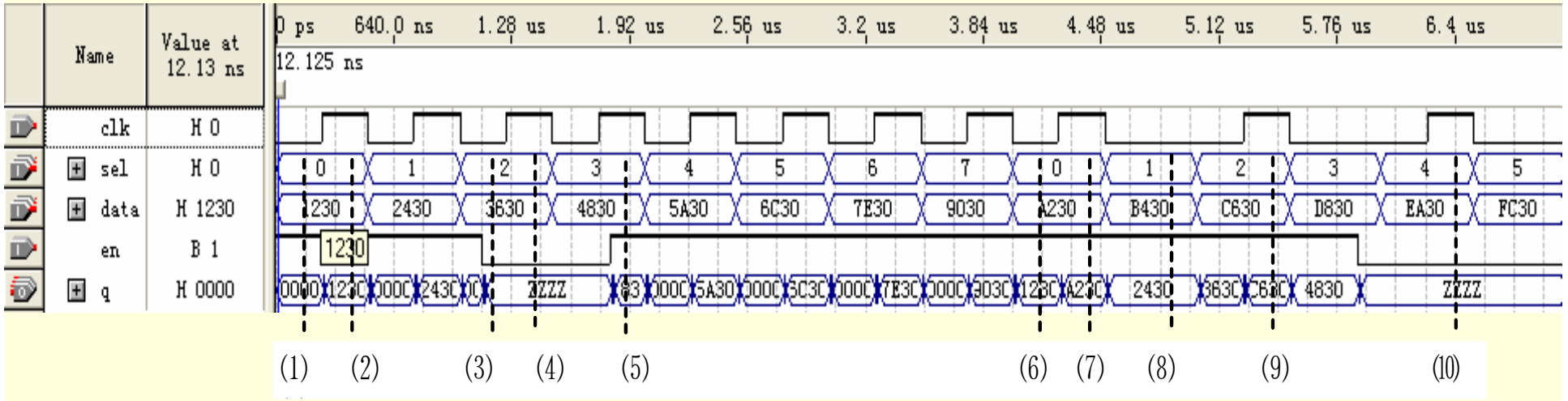

图6-12 寄存器阵列regarray.VHD的仿真波形

6.2.5 移位寄存器SHIFT

【例6-8】sheft.VHD

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use work.cpu_lib.all;
entity shift is
port ( a : in bit16; sel : in t_shift ; y : out bit16);
end shift;
architecture rtl of shift is
begin
shftproc: process(a, sel)
begin
case sel is
when shftpass =>y <= a after 1 ns;
when sftl =>y <= a(14 downto 0) & '0' after 1 ns;
when sftr =>y <= '0' & a(15 downto 1) after 1 ns;
when rotl =>y <= a(14 downto 0) & a(15) after 1 ns;
when rotr =>y <= a(0) & a(15 downto 1) after 1 ns;
when others =>y <= "0000000000000000" after 1 ns;
end case;
end process;
end rtl;
```

6.2 CPU基本部件设计

6.2.5 移位寄存器SHIFT

表6-12　　SHIFT移位运算类型说明

t_shift	移位类型
000	直通
001	无符号数左移
010	无符号数右移
011	循环左移
100	循环右移
其他	输出 0

6.2 CPU基本部件设计

6.2.5 移位寄存器SHIFT

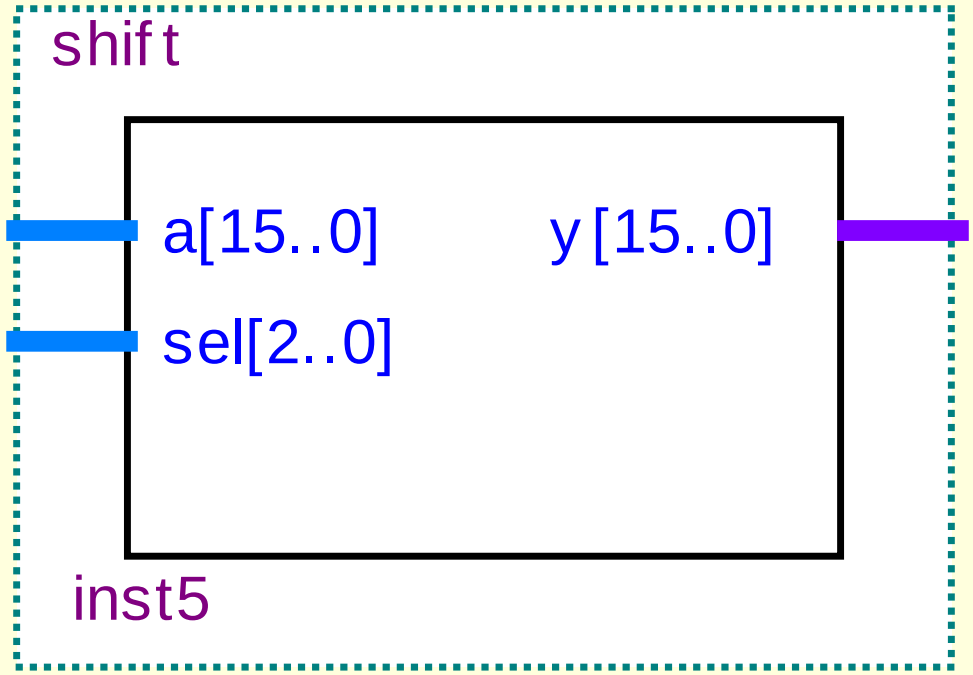

图6-13 移位寄存器的结构图

6.2 CPU基本部件设计

6.2.5 移位寄存器SHIFT

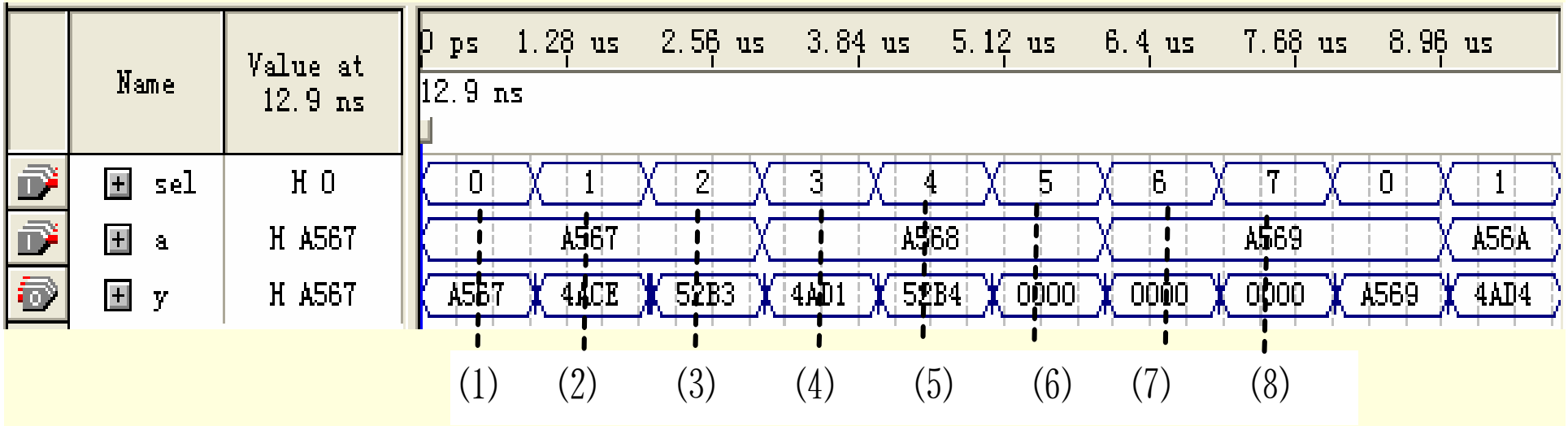

图6-14 移位寄存器SHIFT的仿真波形图

6.2 CPU基本部件设计

6.2.6 三态寄存器TRIREG

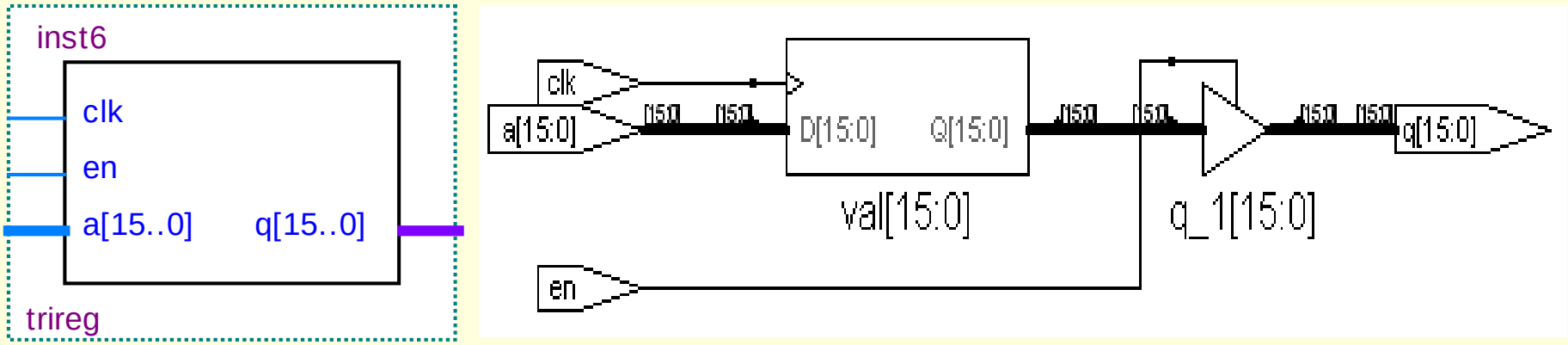

图6-15 三态寄存器triReg的结构图和RTL图

【例6-9】triReg.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use work.cpu_lib.all;
entity trireg is
port( a : in bit16; en，clk : in std_logic; q : out bit16);
end trireg;
architecture rtl of trireg is
signal val : bit16;
begin
triregdata: process
begin
wait until clk'event and clk = '1';
val <= a;
end process;
trireg3st: process(en, val)
begin
if en = '1' then q <= val after 1 ns;
elsif en = '0' then q <= "ZZZZZZZZZZZZZZZZ" after 1 ns;
else q <= "XXXXXXXXXXXXXXXX" after 1 ns; -- exemplar_translate_on
end if;
end process;
end rtl;
```

6.3 CPU的时序仿真与实现

6.3.1 编辑仿真波形文件

1. 建立仿真波形VWF文件

通过仿真波形分析，可以了解CPU在执行指令过程中，各信号的工作时序是否符合设计要求。将工程**TOP**的端口信号节点加入波形编辑器中。top.vwf的波形文件存入文件夹/cpu_16/top中。

将重要的端口节点**RST**、**CLK0**、**STEP**、**Addr**、**data**、**alu**和其它重要的控制信号分别加入到波形编辑窗中。在仿真波形文件中，输入信号有**RST**、**CLK0**和**STEP**。

图6-15 三态寄存器triReg的结构图和RTL图

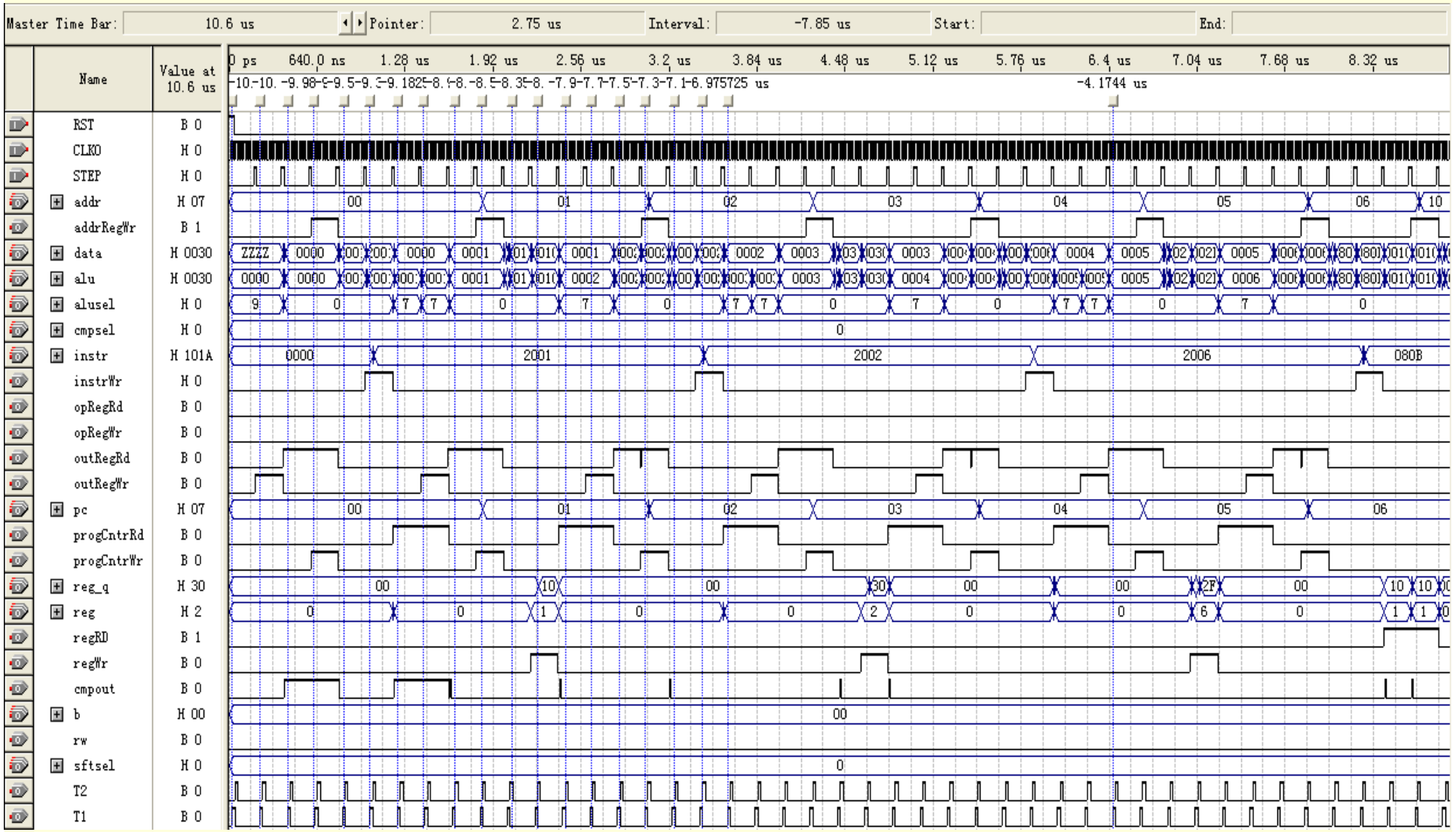
Master Time Bar: 10.6 us
Pointer: 2.75 us
Interval: -7.85 us
Start:
End:
Name
Value at 10.6 us
RST
CLKO
STEP
addr
addrRegWr
data
alu
alusel
cmpsel
instr
instrWr
opRegRd
opRegWr
outRegRd
outRegWr
pc
progCntrRd
progCntrWr
reg_q
reg
regRD
regWr
cmpout
b
rw
sftsel
T2
T1

图6-16 仿真输出波形

6.3 CPU的时序仿真与实现

6.3.1 编辑仿真波形文件

1. 建立仿真波形VWF文件

表6-13 图6-16仿真波形对应的程序

地址addr	机器码	指令	说明
0000	2001	LOADI R1，0010H	源地址送R1
0001	0010		
0002	2002	LOADI R2，0030H	目标地址送R2
0003	0030		
0004	2006	LOADI R6，002FH	结束地址送R6
0005	002F		
0006	……		

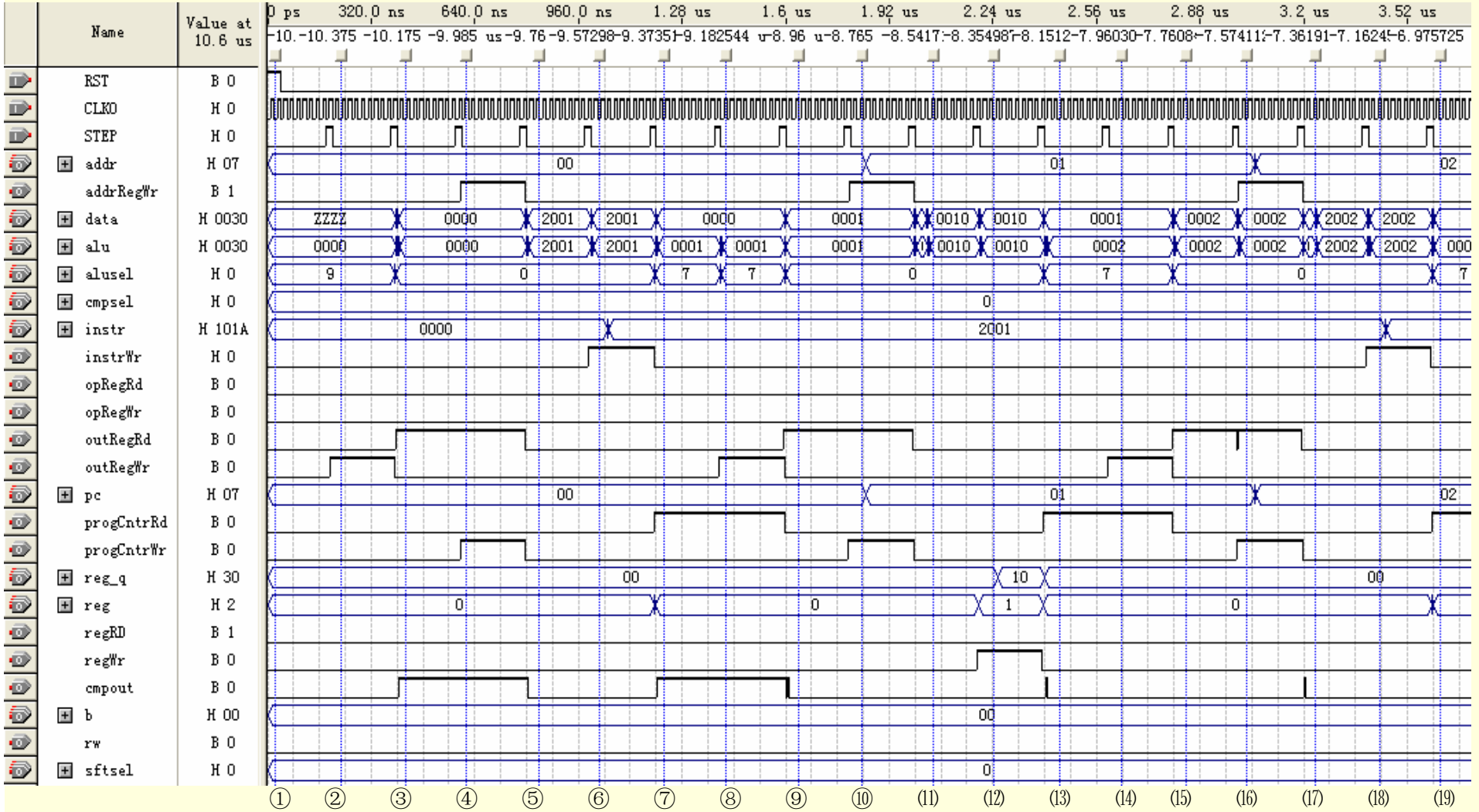

图6-17 CPU复位和第1条指令的仿真波形

6.3 CPU的时序仿真与实现

6.3.1 编辑仿真波形文件

2. CPU的RTL电路结构图

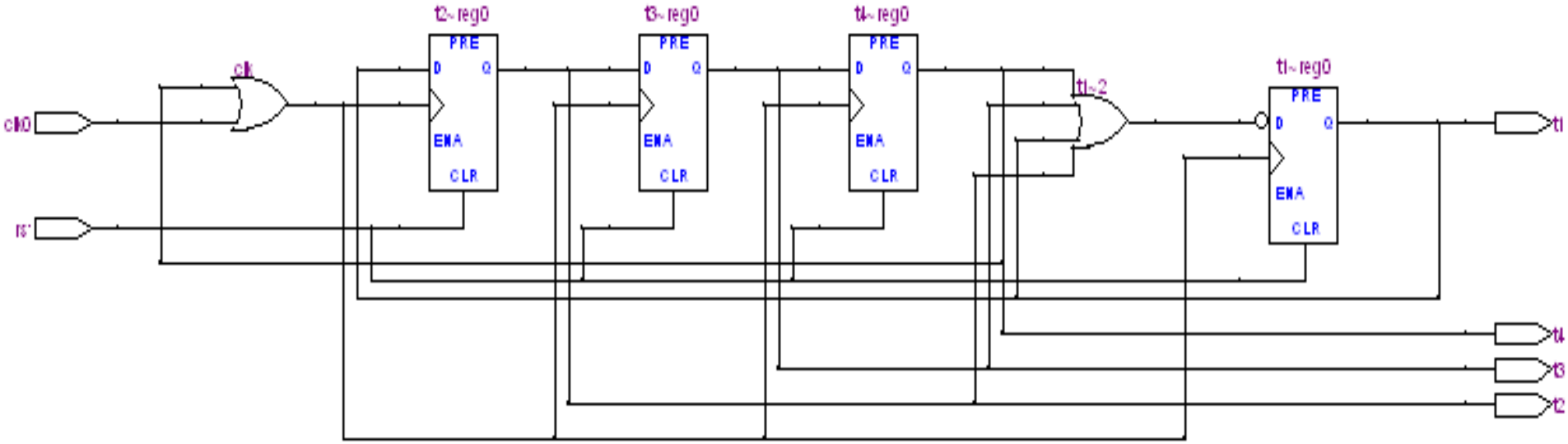

图6-18 STEP时序控制器的RTL电路图

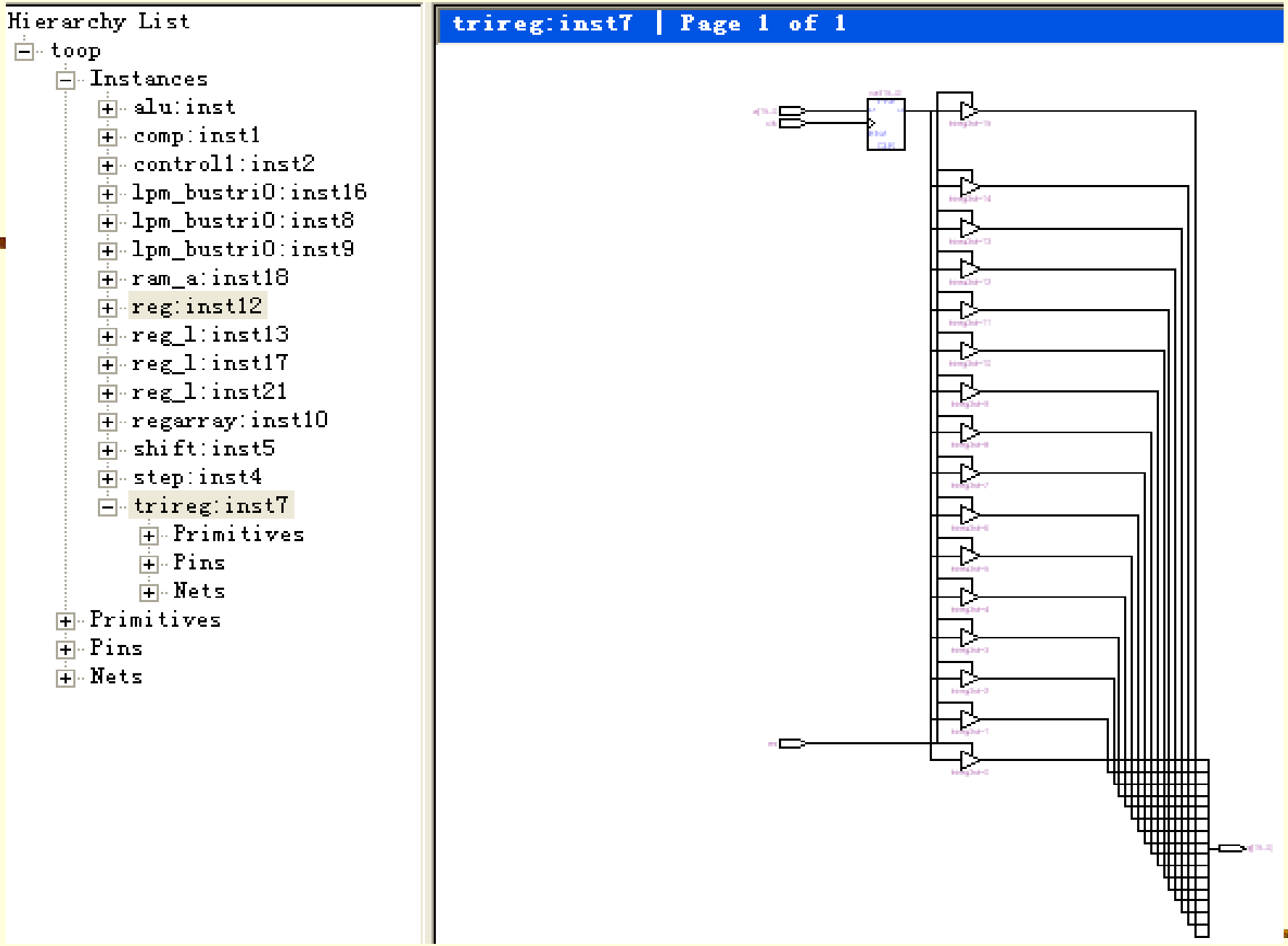

图6-19 三态触发寄存器的RTL电路图

6.3 CPU的时序仿真与实现

6.3.2 16位CPU的调试运行

6.3.3 应用嵌入式逻辑分析仪调试CPU

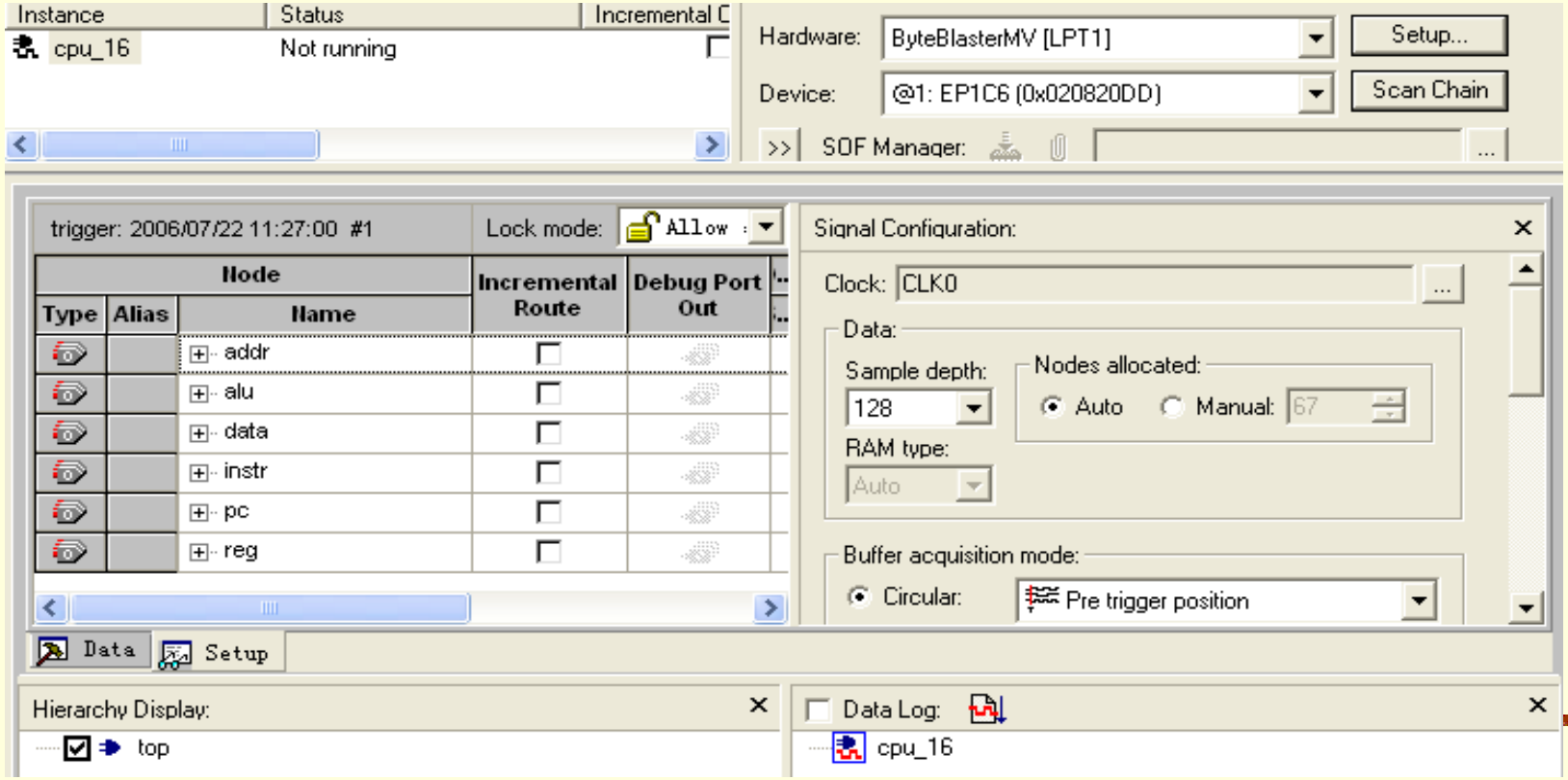

图6-20 信号调入观察窗，存盘

6.3 CPU的时序仿真与实现

6.3.3 应用嵌入式逻辑分析仪调试CPU

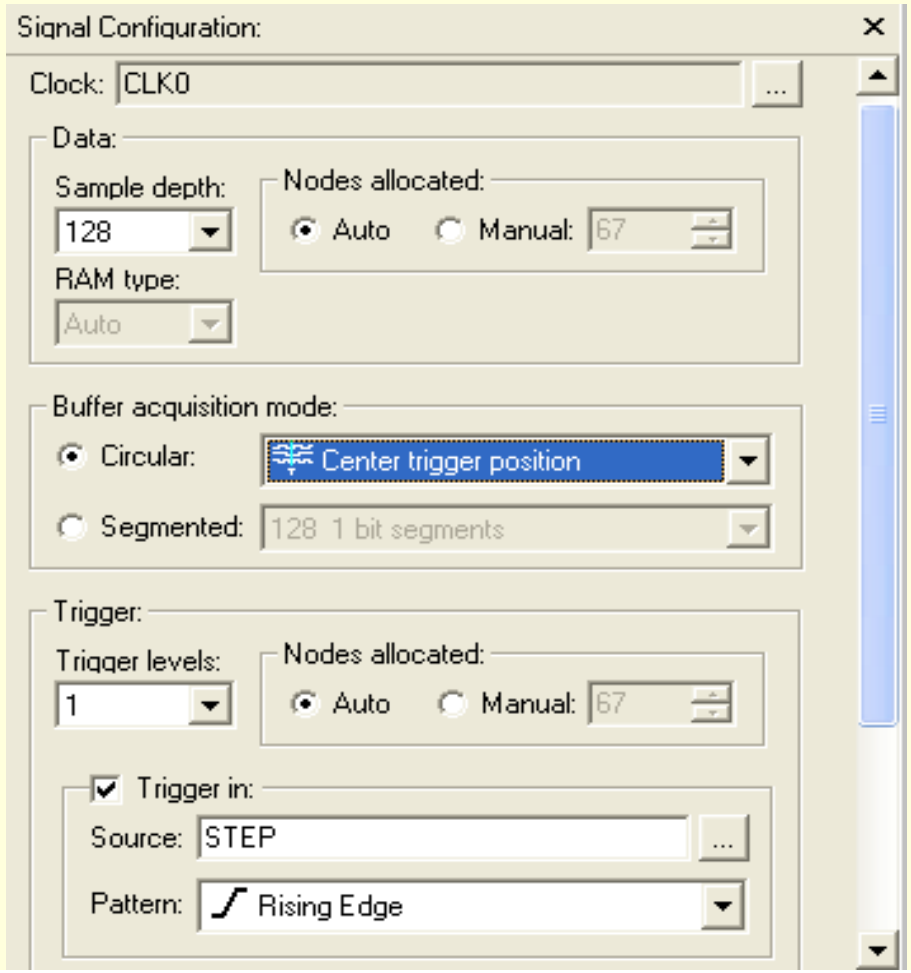

图6-21 的全屏编辑窗

6.3 CPU的时序仿真与实现

6.3.3 应用嵌入式逻辑分析仪调试CPU

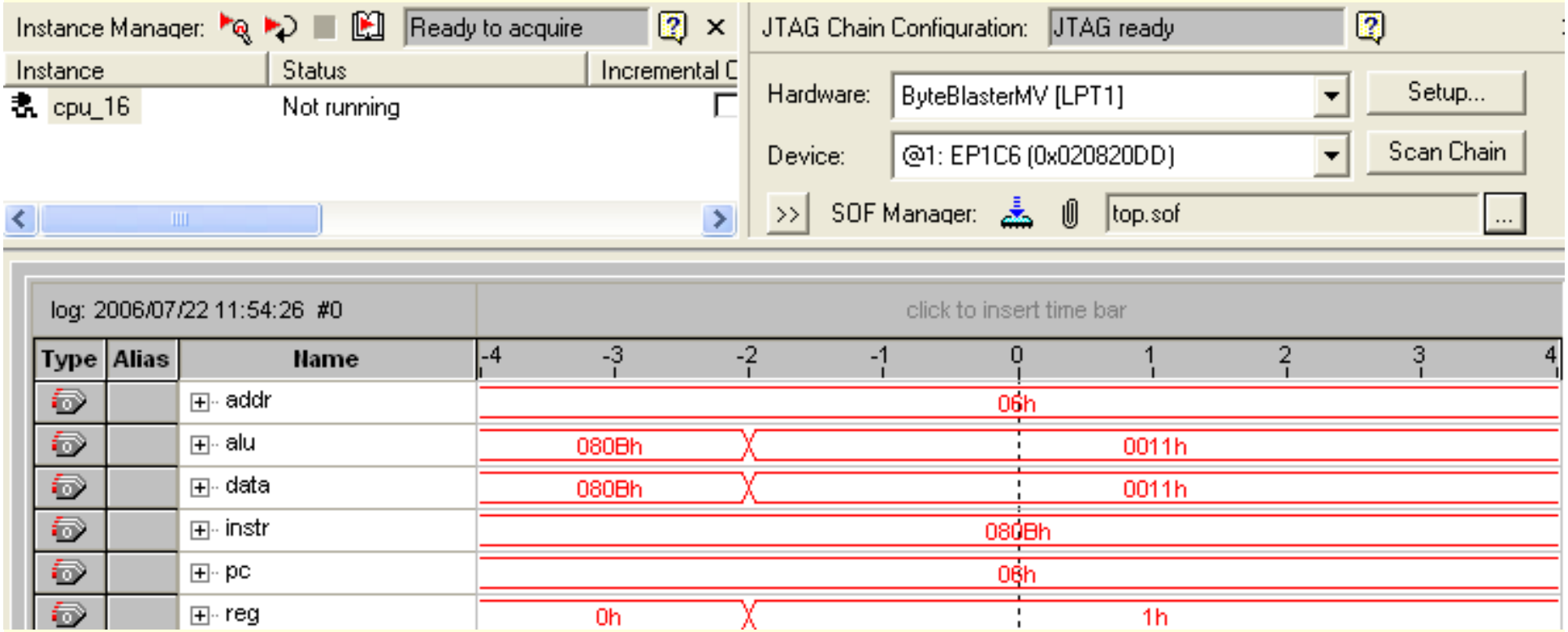

图6-22 SignalTapII数据窗的实时信号

6.3 CPU的时序仿真与实现

6.3.4 对配置器件编程

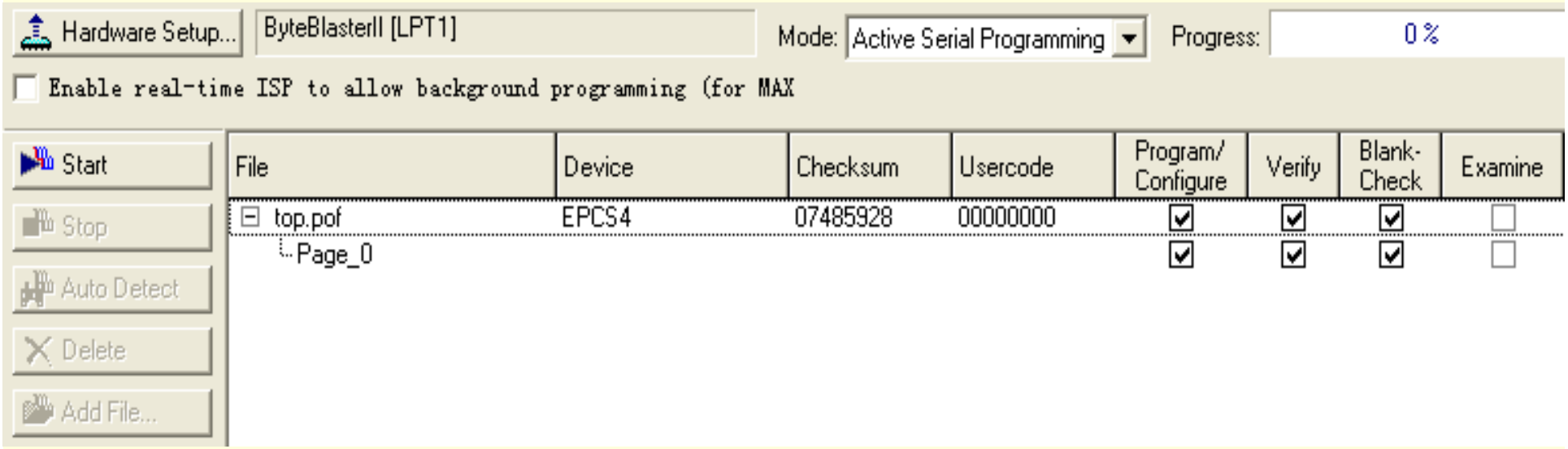

图6-24 ByteBlasterII接口AS模式编程窗口

6.4 应用程序设计实例

6.4.1 乘法算法及其硬件实现

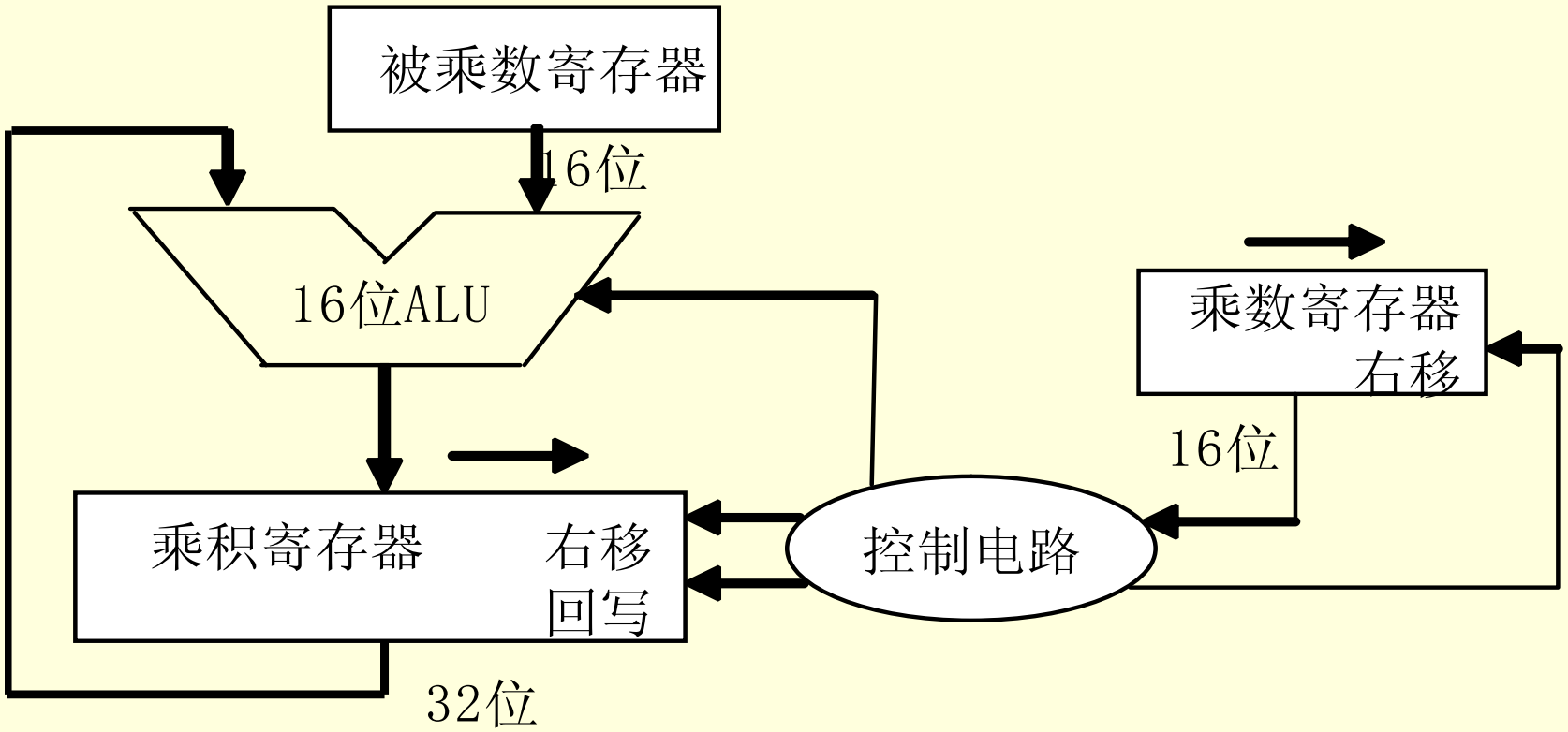

图6-25 乘法算法1的硬件实现

6.4 应用程序设计实例

6.4.1 乘法算法及其硬件实现

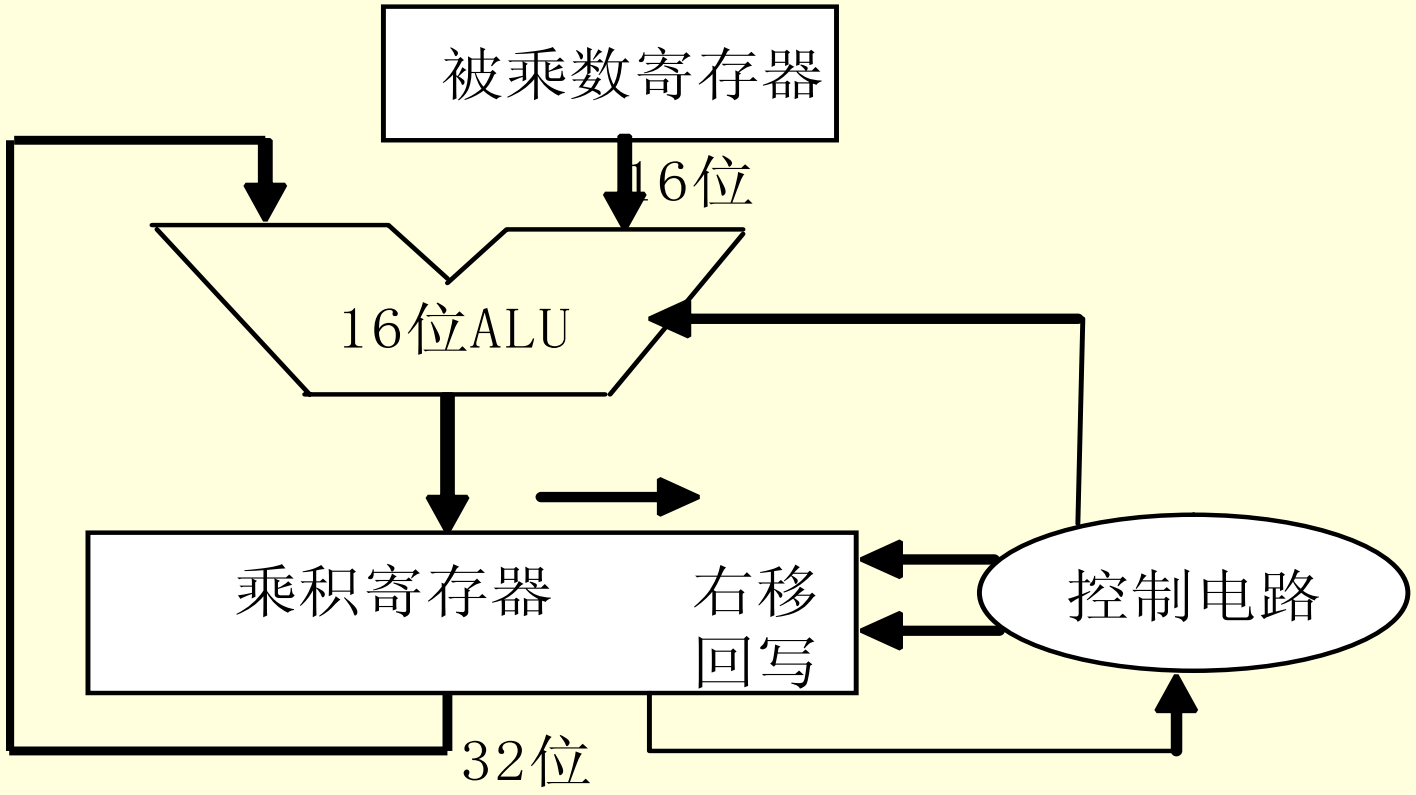

图6-26 改进后的乘法算法2的硬件实现

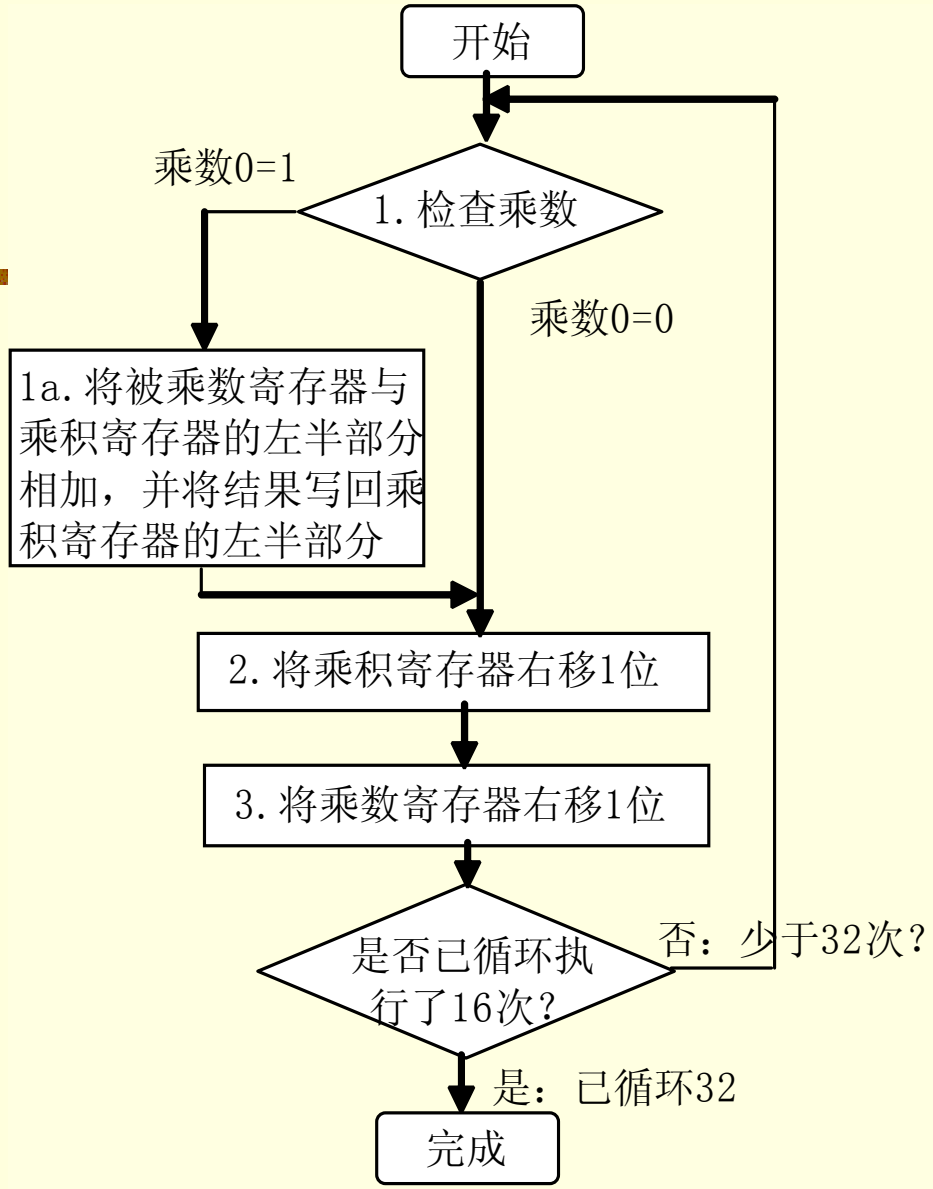

图6-27 乘法算法1的流程图

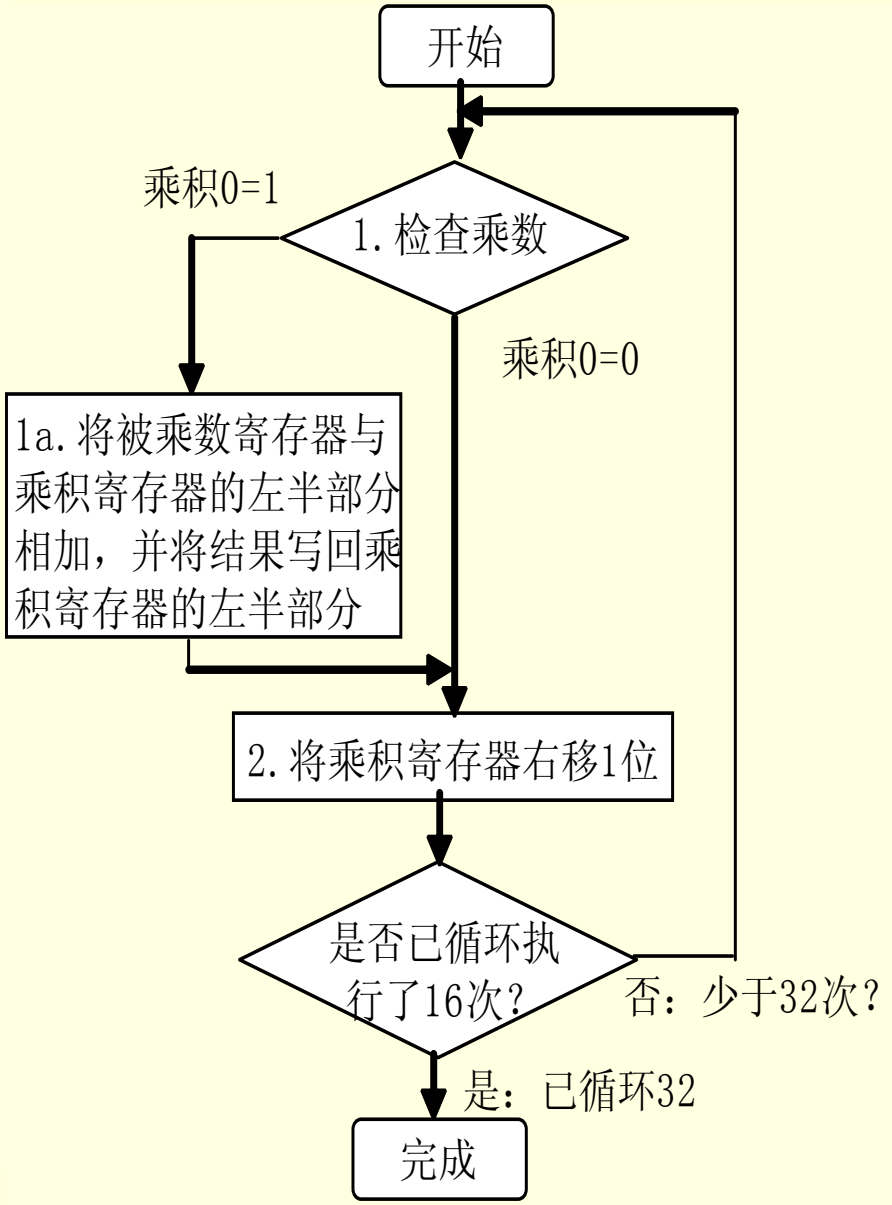

图6-28 乘法算法2的流程图

6.4 应用程序设计实例

6.4.2 除法算法及其硬件实现

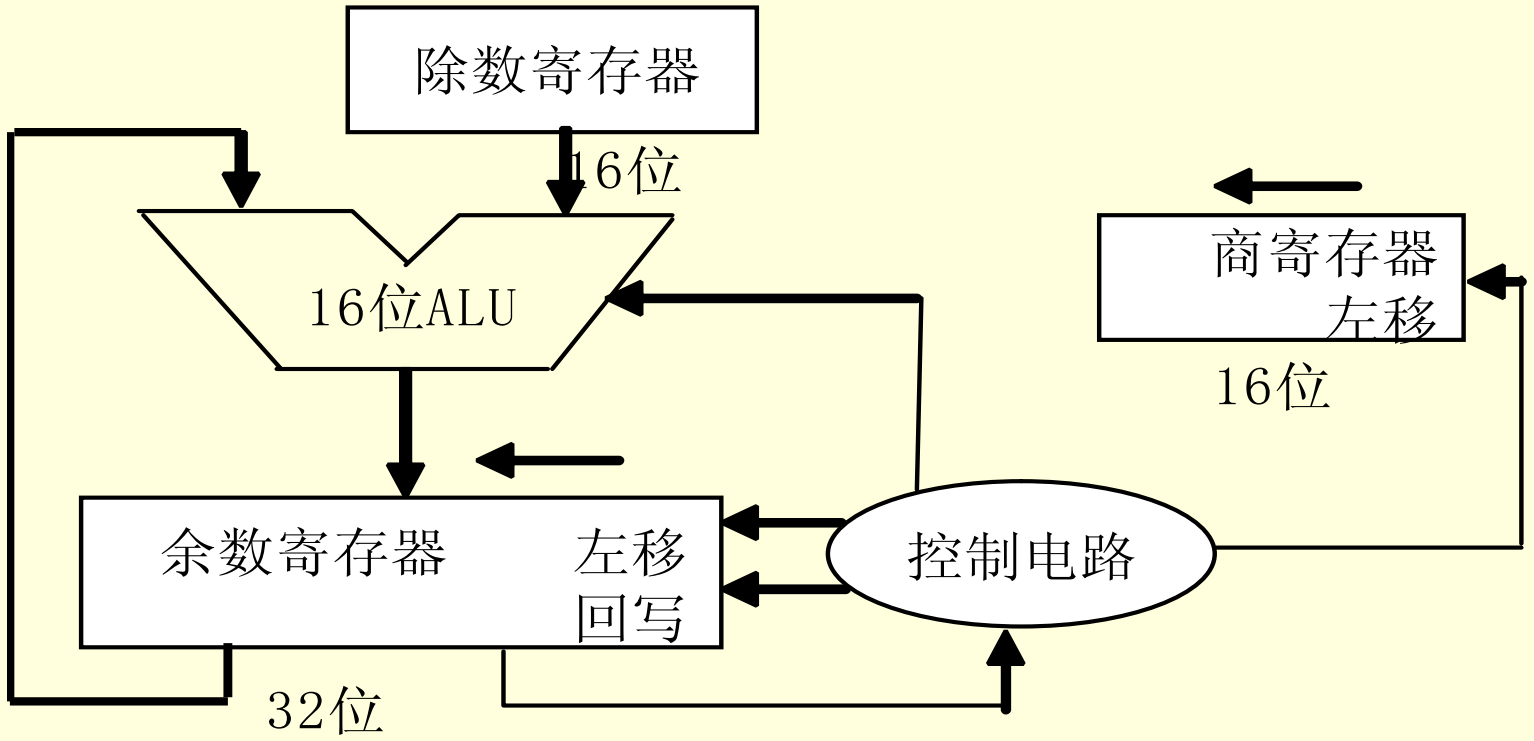

图 6-29 除法算法1的硬件结构

6.4 应用程序设计实例

6.4.2 除法算法及其硬件实现

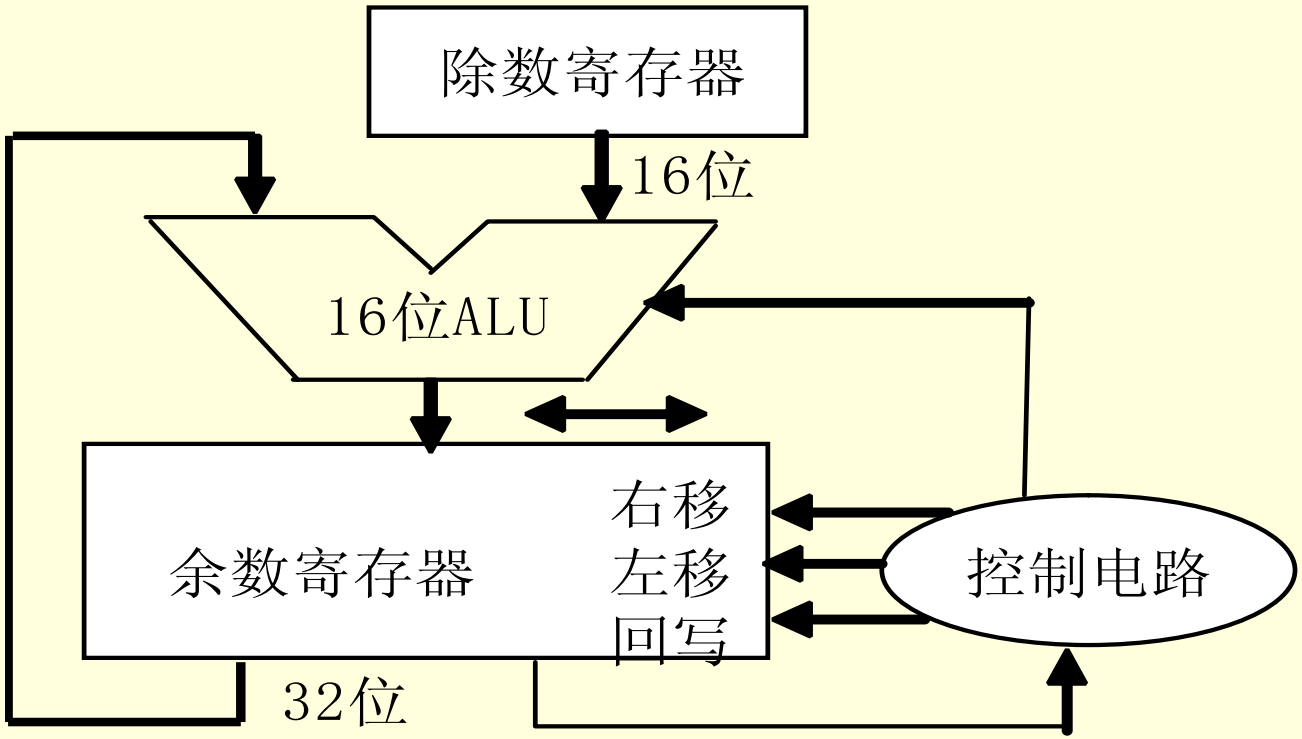

图 6-30 除法算法2的硬件结

习 题

6-1. 对CPU进行修改，为其增加一个状态寄存器FLAG，FLAG可以中保存进位标志和零标志。

6-2. 修改CPU，为其加入一条带进位加法指令ADDC，给出ADDC指令的运算流程，对控制器的控制程序作相应的修改。

6-3. 说明在16位CPU中，PC←PC+1操作是如何执行的？

6-4. 根据图6-26和图6-27的电路结构和流程图,设计乘法应用程序，进行计算机仿真验证程序功能，并在16位CPU上调试运行。

6-5. 根据图6-30和图6-31的电路结构和流程图,设计除法应用程序，进行计算机仿真验证程序功能，并在16位CPU上调试运行。

6-6. 请说明CONTROL.vhd中的两个进程各自的作用，两个进程之间是如何发生联系的？

6-7. 简要说明16位CPU的复位过程。

6-8. 根据control.vhd中状态机的描述，说明指令MOVE R1，R2 的执行过程。

实验与设计

实验6-1. 16位计算机基本部件实验

参考实验示例和实验课件：

/CMPUT_EXPMT/CH6_Expt/DEMO_61/ 和 实验6_1.ppt 。

实验6-2. 16位CPU设计综合实验

参考实验示例和实验课件：

/CMPUT_EXPMT/CH6_Expt/DEMO_62/TOP 和 实验6_2.ppt 。

实验与设计

表6-14　　汇编语言指令格式

地址	机器码	指令	功能说明
0000H 0001H	2001H　0021H	LOADI　R1,0021H	源操作数首地址（0021）送 R1
0002H 0003H	2002H　0058H	LOADI　R2,0058H	目的操作数首地址（0058）送 R2
0004H 0005H	2006H　0040H	LOADI　R6,0040H	结束地址（0040）送 R6
0006H	080BH	LOAD　R3，[R1]	取数，从 R1 指定的存储单元取数送 R3
0007H	101AH	STORE　[R2]，R3	存数，将 Reg3 的内容存入 R2 指定的存储单元
0008H 0009H	300EH　0000H	BRANCHGTI　[0000]	比较 R1>R6 ？，若 yes，则转向地址[0000]
000AH	3801H	INC　R1	修改源指针 R1<=R1+1
000BH	3802H	INC　R2	修改目的指针 R2<=R2+1
000CH	2800H　0006H	BRANCHI　[0006]	这是一个绝对地址转移指令，goto 到地址[0006]，执行此处指令，实现循环。

实验与设计

top - [RAM_16.mif]

ocessing Tools Window Help

top

Addr	+0	+1	+2	+3	+4	+5	+6	+7
00	2001	0021	2002	0058	2006	0040	080B	101A
08	300E	0000	3801	3802	2800	0006	0000	0000
10	0000	0000	0000	0000	0000	0000	0000	0000
18	0000	0000	0000	0000	0000	0000	0000	0000
20	0000	FF00	4321	5432	6543	4433	5544	2DFF
28	1234	2345	3456	2030	3033	3068	2D2D	3466
30	A1A2	B2B3	C3C4	D5D6	E6E7	F8F9	ABCD	EF01
38	1212	2323	3434	5656	7878	8989	ABAB	CDCD
40	EFEF	0000	0000	0000	0000	0000	0000	0000
48	0000	0000	0000	0000	0000	0000	0000	0000
50	0000	0000	0000	0000	0000	0000	0000	0000
58	0000	0000	0000	0000	0000	0000	0000	0000
60	0000	0000	0000	0000	0000	0000	0000	0000
68	0000	0000	0000	0000	0000	0000	0000	0000
70	0000	0000	0000	0000	0000	0000	0000	0000
78	0000	0000	0000	0000	0000	0000	0000	0000

图6-31 在QUARTUS II环境下编辑ram_16.mif文件

实验与设计

Instance | Ready to acquire

I...	Instance ID	Status	Width	Depth	Type	Mode
0	RAM	Not run...	16	128	RAM/ROM	Read/Write

JTAG Chain | JTAG ready

Hardware: ByteBlasterMV [LPT1]

Device: @1: EP1C6 (0x020820DD)

0 RAM:

000000	20 01	00 21	20 02	00 58	20 06	00 40	08 0B	10 1A	30 0E	00 00	38 01	..! ..X .
00000B	38 02	28 00	00 06	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	8.(.....
000016	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	.....
000021	FF 00	43 21	54 32	65 43	44 33	55 44	2D FF	12 34	23 45	34 56	20 30	..C!T2eCD3
00002C	30 33	30 68	2D 2D	34 66	A1 A2	B2 B3	C3 C4	D5 D6	E6 E7	F8 F9	AB CD	030h--4f..
000037	EF 01	12 12	23 23	34 34	56 56	78 78	89 89	AB AB	CD CD	EF EF	00 00	....##44VV
000042	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	.....
00004D	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	.....
000058	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	.....
000063	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	.....
00006E	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	.....
000079	00 00	00 00	00 00	00 00	00 00	00 00	00 00					.....

图6-32用In-System Memory Content Editor读取的数据文件

实验与设计

0 RAM:

000000	20 01	00 21	20 02	00 58	20 06	00 40	08 0B	10 1A	30 0E	00 00	38 01	..! ..X ..@....0...8.
00000B	38 02	28 00	00 06	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	8.(.....
000016	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	.....
000021	FF 00	43 21	54 32	65 43	44 33	55 44	2D FF	12 34	23 45	34 56	20 30	..C!T2eCD3UD-..4#E4V 0
00002C	30 33	30 68	2D 2D	34 66	A1 A2	B2 B3	C3 C4	D5 D6	E6 E7	F8 F9	AB CD	030h--4f.....
000037	EF 01	12 12	23 23	34 34	56 56	78 78	89 89	AB AB	CD CD	EF EF	00 00	....##44VVxx.....
000042	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	.....
00004D	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	.....
000058	FF 00	43 21	54 32	65 43	44 33	55 44	2D FF	12 34	23 45	34 56	20 30	..C!T2eCD3UD-..4#E4V 0
000063	30 33	30 68	2D 2D	34 66	A1 A2	B2 B3	C3 C4	D5 D6	E6 E7	F8 F9	AB CD	030h--4f.....
00006E	EF 01	12 12	23 23	34 34	56 56	78 78	89 89	AB AB	CD CD	EF EF	00 00	....##44VVxx.....
000079	00 00	00 00	00 00	00 00	00 00	00 00	00 00	.....

图6-33数据搬运完毕后的In-System Memory Content Editor窗

实验与设计

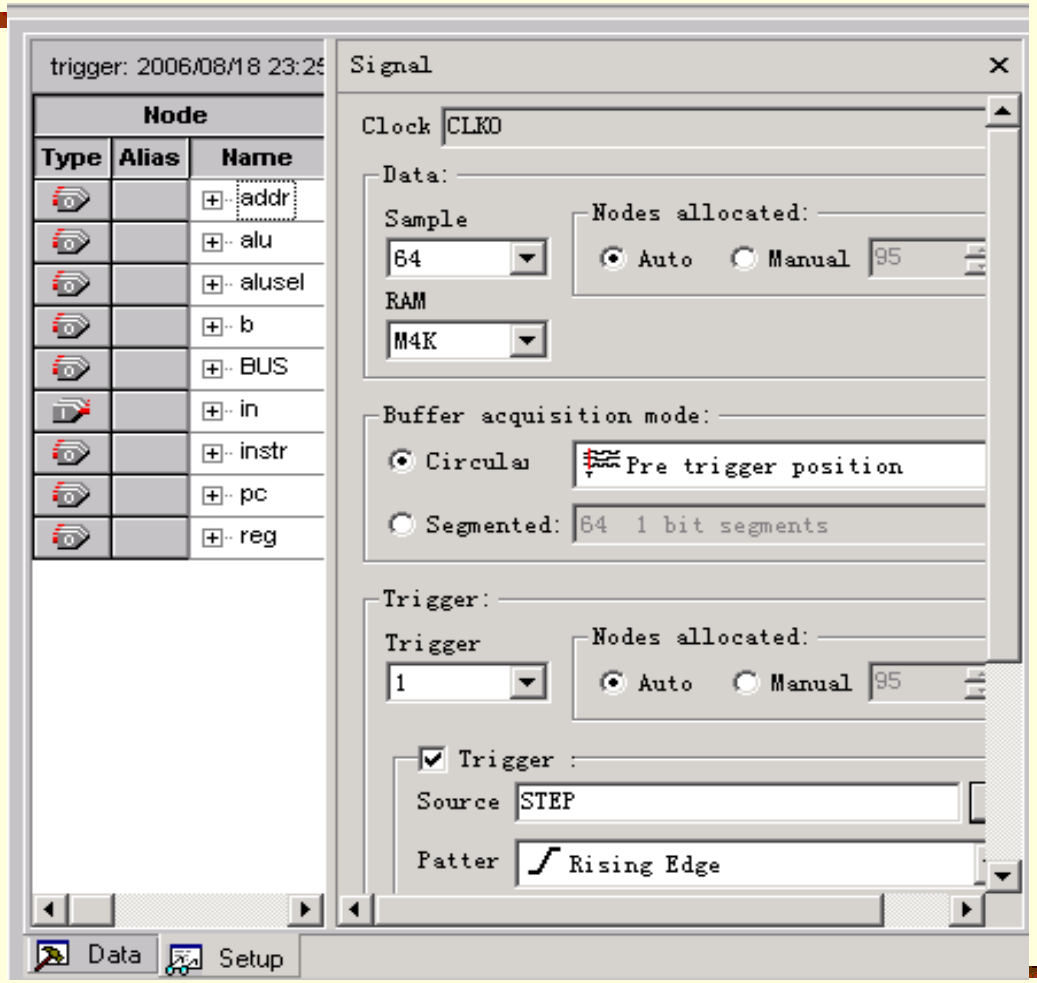

图6-34 嵌入式逻辑分析仪设置情况图

实验与设计

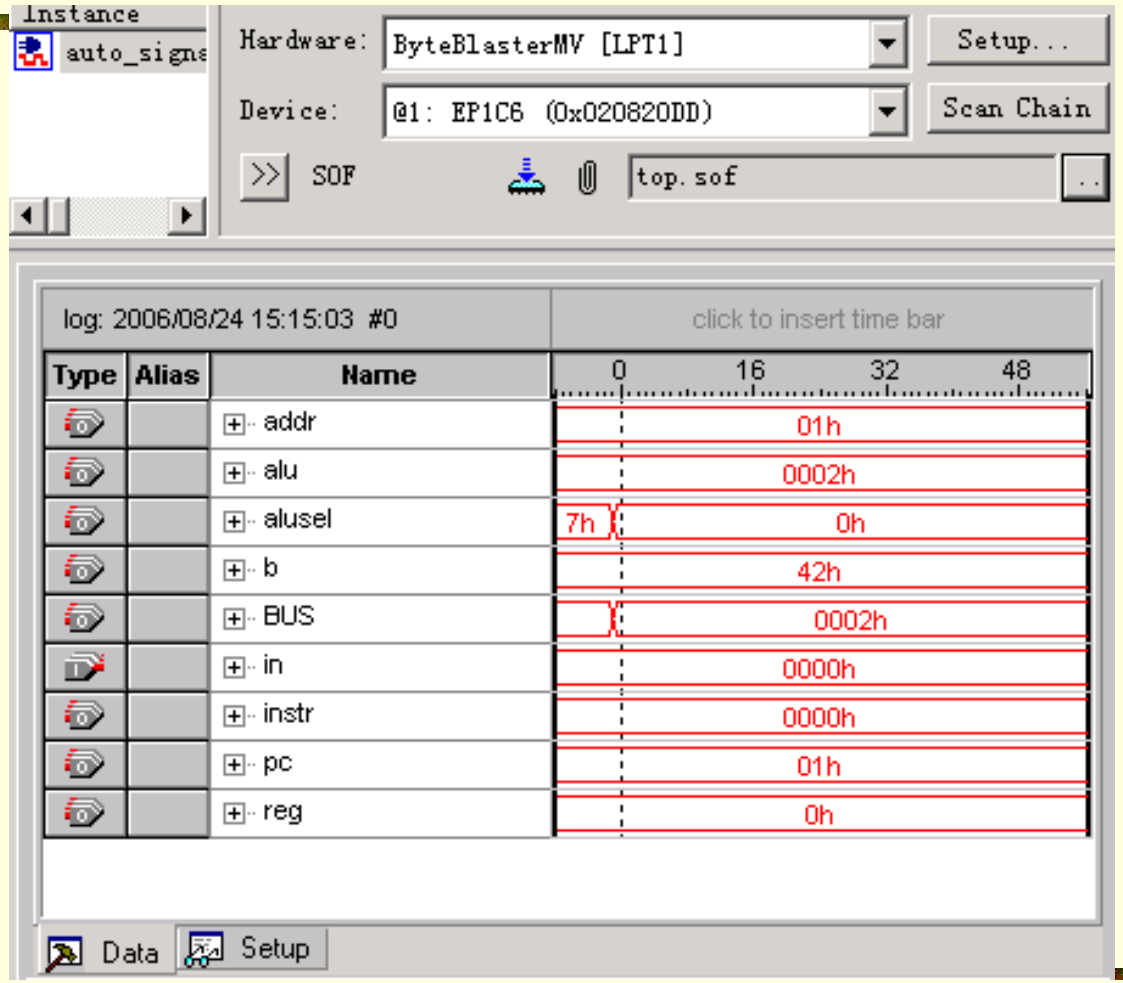

图6-35 CPU运行时逻辑分析仪显示波形

实验与设计

表6-15　　LCD液晶显示屏显示数据说明

名称	作 用	名称	作 用
IN	输入单元 INPUT	OUT	输出单元 OUTPUT
ALU	算术逻辑单元	BUS	数据总线
DR	数据寄存器 R	REG	寄存器阵列
AR	地址寄存器	PC	程序计数器
RAM	程序/数据存储器	IR	指令寄存器

实验与设计

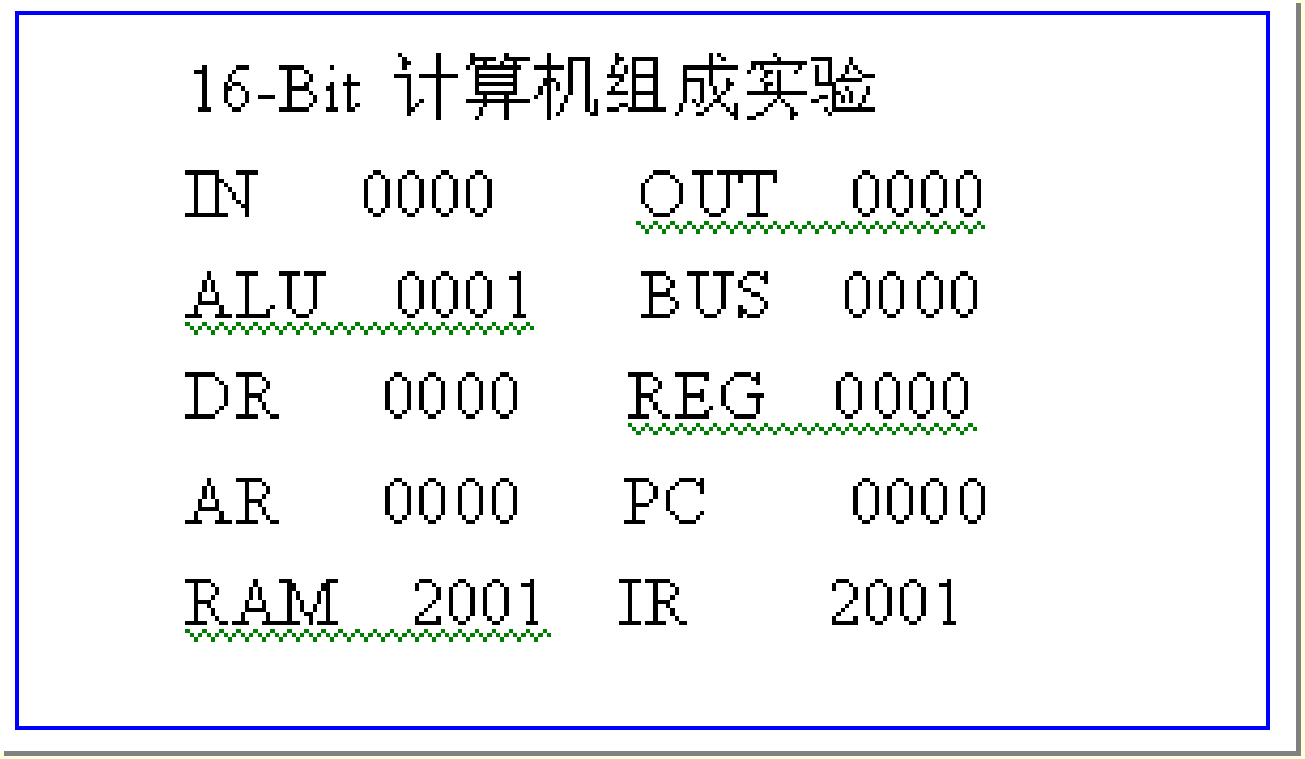

图6-36 LCD液晶显示屏

实验与设计

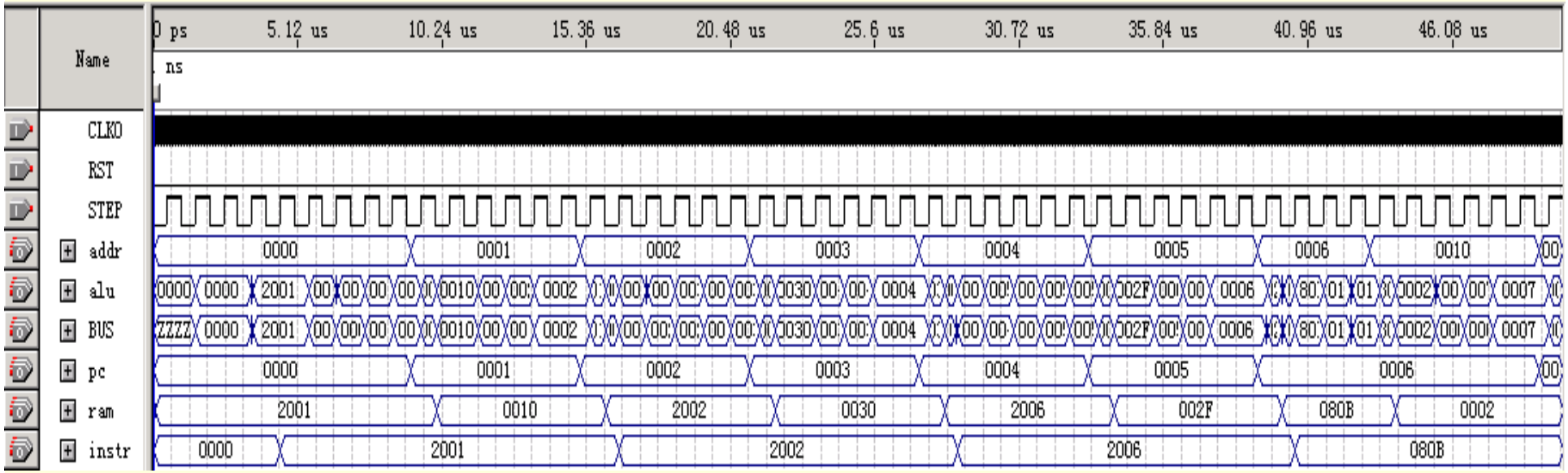

图6-37 CPU_16仿真波形