现代计算机组成原理

潘 明　潘 松 编著

科学出版社

第 8 章

16位流水线CPU设计

8.1 流水线CPU的结构

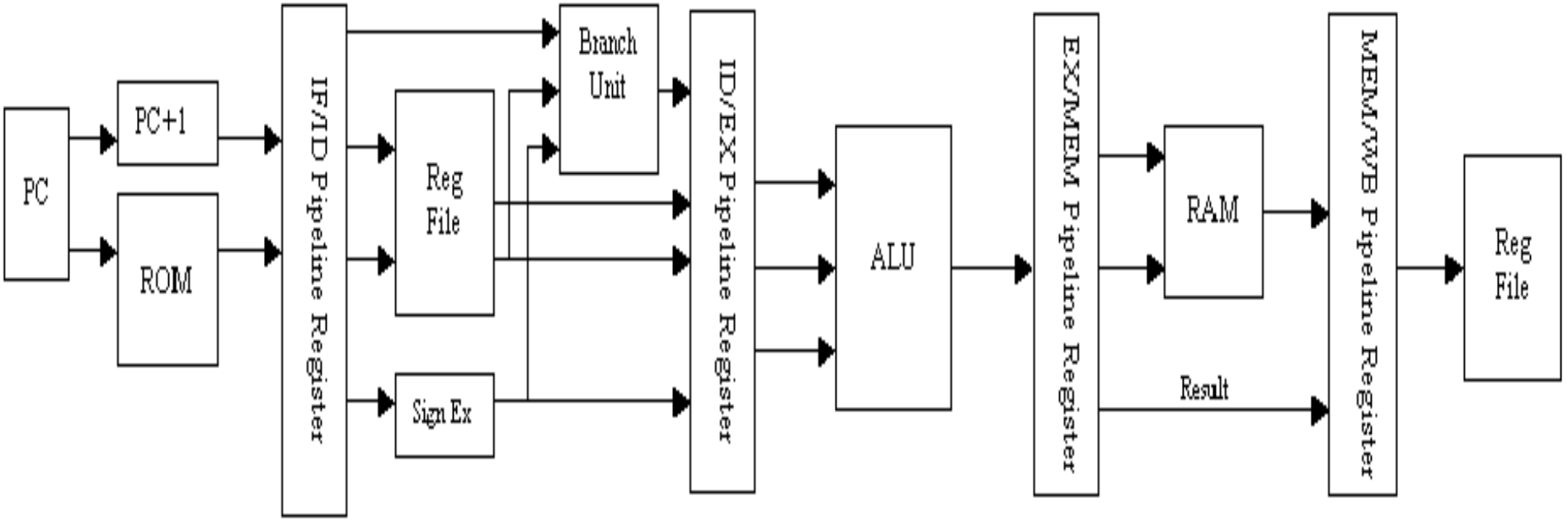


图8-1 流水线CPU的结构

8.2 指令系统设计

- 指令系统的完备性
- 指令系统的高效性
- 指令系统的规整性
- 指令系统的兼容性

图8-1 流水线CPU的结构

操作码	指令	形式	功能描述	功能码 func
0000	NOP	R	无操作	000
	ADD R1, R2, R3	R	有符号数加法 (R1 = R2 + R3)	010
	ADDu R1, R2, R3	R	无符号数加法(R1 = R2 + R3)	001
	SUB R1, R2, R3	R	有符号数减法(R1 = R2 - R3)	011
	SUBu R1, R2, R3	R	无符号数减法 (R1 = R2 - R3)	100
	SLT R1, R2, R3	R	有符号数比较，小于时置位 (R1=0 if R2<R3 else R1=1)	101
	SLTu R1, R2, R3	R	无符号数比较，小于时置位	110
0001	NOT R1, R2	R	逻辑非 (R1 = NOT R2)	
	AND R1, R2, R3	R	逻辑与	000
	OR R1, R2, R3	R	逻辑或	001
	XOR R1, R2, R3	R	逻辑异或	010
	NOR R1, R2, R3	R	逻辑或非	011
	SLL R1, R2, R3	S	有符号数逻辑左移 (R1 = R2 shifted by R3)	100
	SRL R1, R2, R3	S	有符号数逻辑右移	101
	SRA R1, R2, R3	S	有符号数算术右移	110
	ROR R1, R2, R3	S	有符号数循环右移	111
0010	IN R1	R	从端口输入	000
	OUT R1	R	输出到端口	001

图8-1 流水线CPU的结构

	BZ R1, R2	R	为 0 时转移 (If R1=0 jump to loc R2)	010
	BNZ R1, R2	R	不为 0 时转移(If R1 not 0 jump to loc R2)	011
	EI data6	R	允许中断 (data6 中的每 1 位代表 1 个中断是打开还是关闭)	Other
0011	JAL R1,R2	R	跳转与链接	000
	RET	R	子程序返回	001
	RETI	R	中断子程序返回	010
0100	MVIL R1, data8	I	将立即数装入低字节 (将 data8 装入 R1 的低字节)	0
	MVIH R1, data8	I	将立即数装入高字节	1
0101	BZI R1, data8	I	为 0 时，以 PC 作相对转移(If R1=0 jump to loc PC+data8)	0
	BNZI R1, data8	I	不为 0 时，以 PC 作相对转移(If R1=1 jump to loc PC+data8)	1
0111	SLLI R1, R2, data5	SI	按立即数 data5,有符号数逻辑左移	0
	SRLI R1, R2, data5	SI	按立即数 data5,有符号数逻辑右移	1
1000	SRAI R1, R2, data5	SI	按立即数 data5,有符号算术辑右移	0
	RORI R1, R2, data5	SI	按立即数 data5,有符号数循环右移	1
1001	ADDI R1, R2, data6	RI	有符号数与立即数相加 (R1 = R2 + data6)	
1010	SUBI R1, R2, data6	RI	有符号数与立即数相减	
1011	LW R1, R2, data6	RI	装入字	
1100	SW R1, R2, data6	RI	存储字	
说明	Operation 为 4-bit， R1、 R2、 R3 均为 3-bit， data5、 data6、 data8 分别为 5、 6、 8-bit.			

8.2 指令系统设计

8.2.1寄存器型 (R-型)

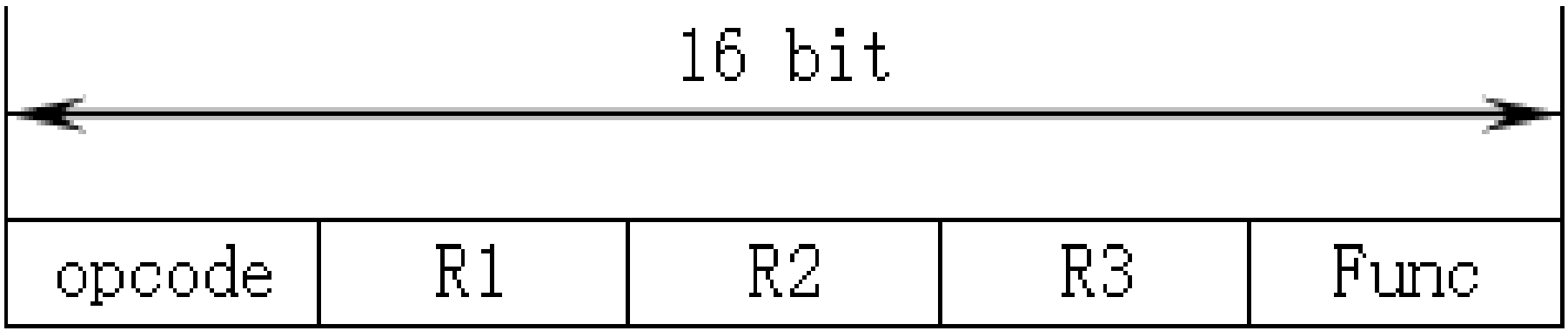


图8-2 R-型指令 ADD R1,R2,R3

8.2 指令系统设计

8.2.2 寄存器立即数型(RI-型)

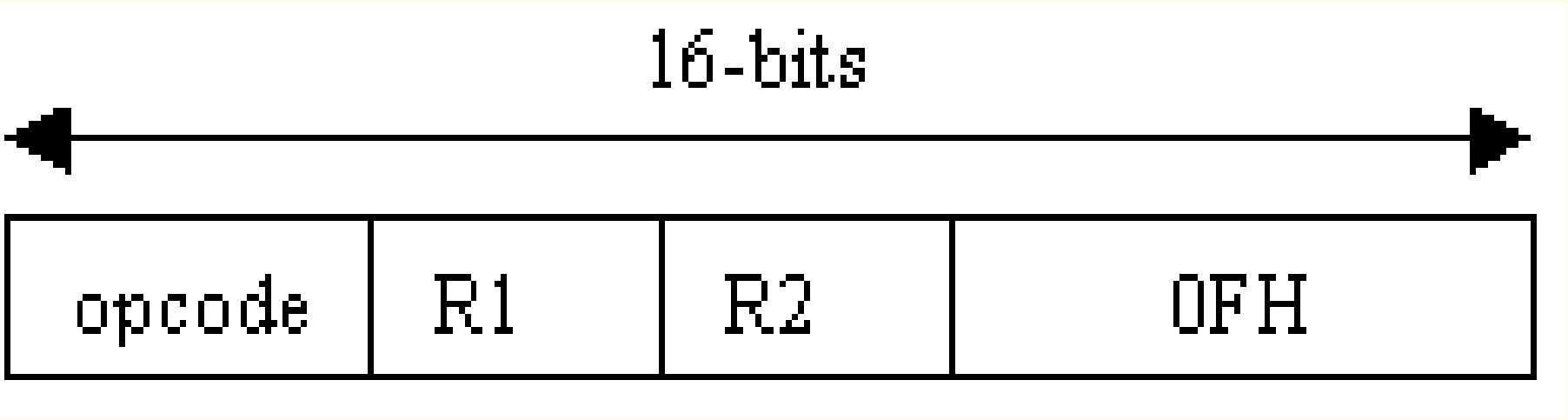


图8-3 RI-型指令 ADDI R1,R2,0FH

8.2 指令系统设计

8.2.3 立即数型 (I-型)

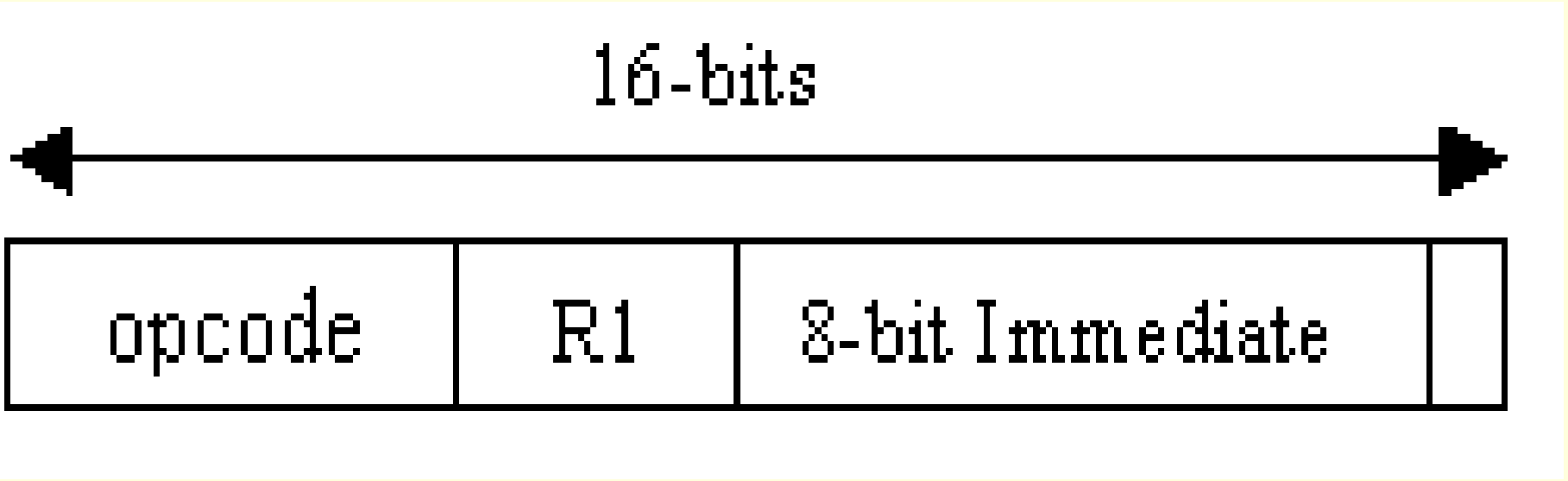


图8-4 I-型指令 MVIL R1,FFH

8.2 指令系统设计

8.2.4 立即移位型 (SI-型)

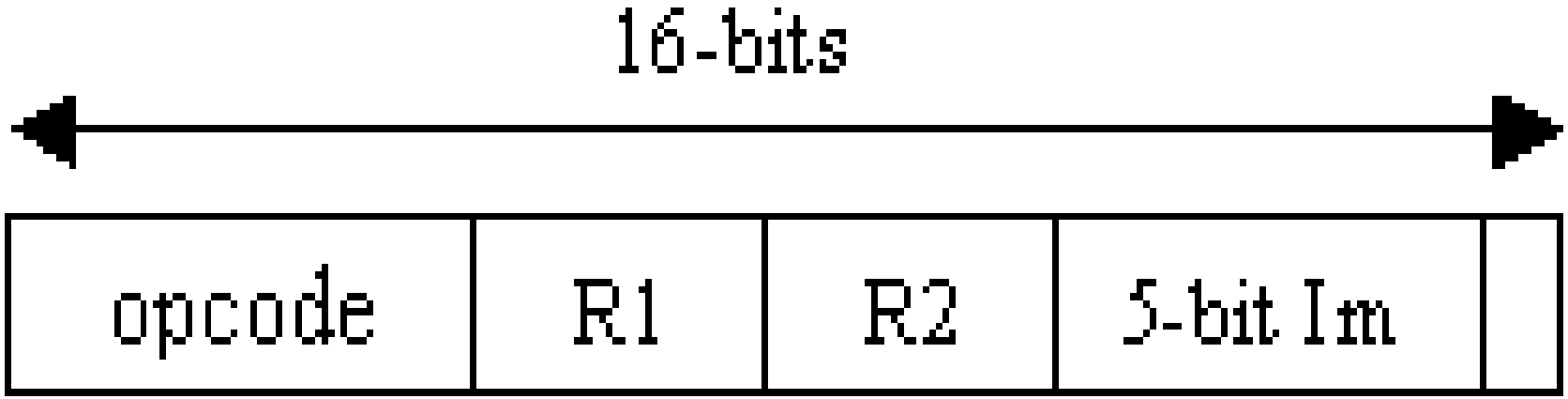


图8-5 SI-型指令 RORI R1,R2,2H

8.3 数据通路设计

8.3.1 R-型数据通路

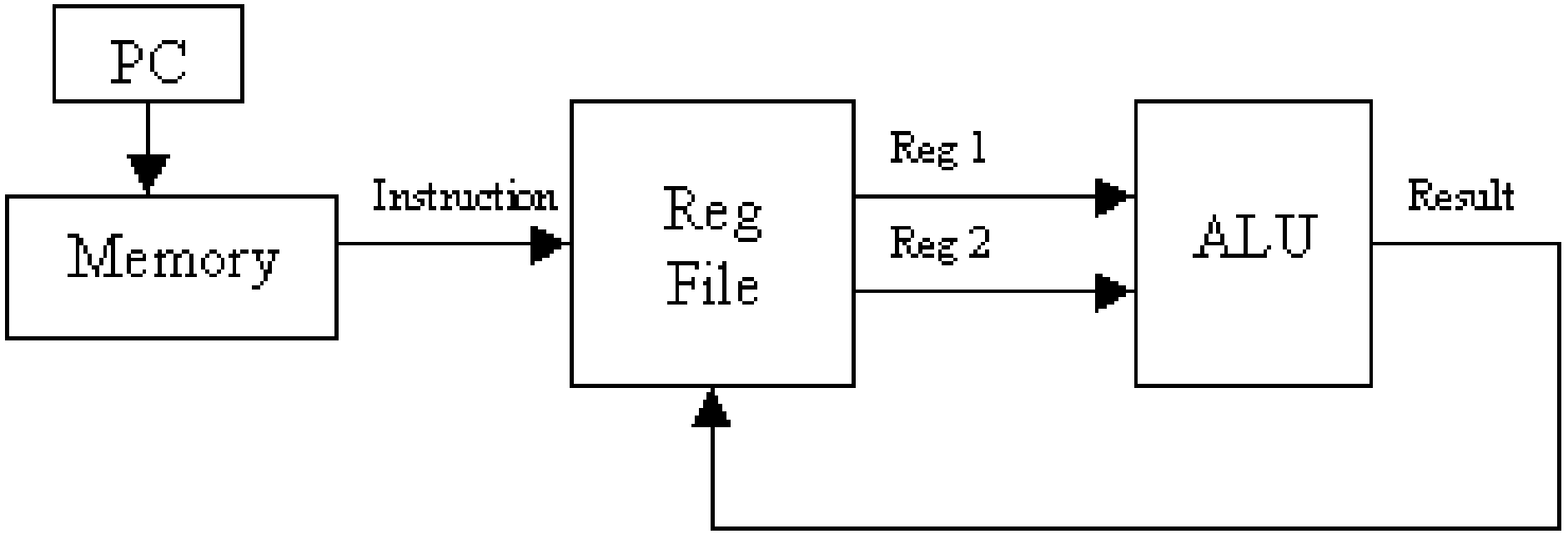


图8-6 R-型 ALU 指令的数据通路

8.3 数据通路设计

8.3.2 RI-型数据通路

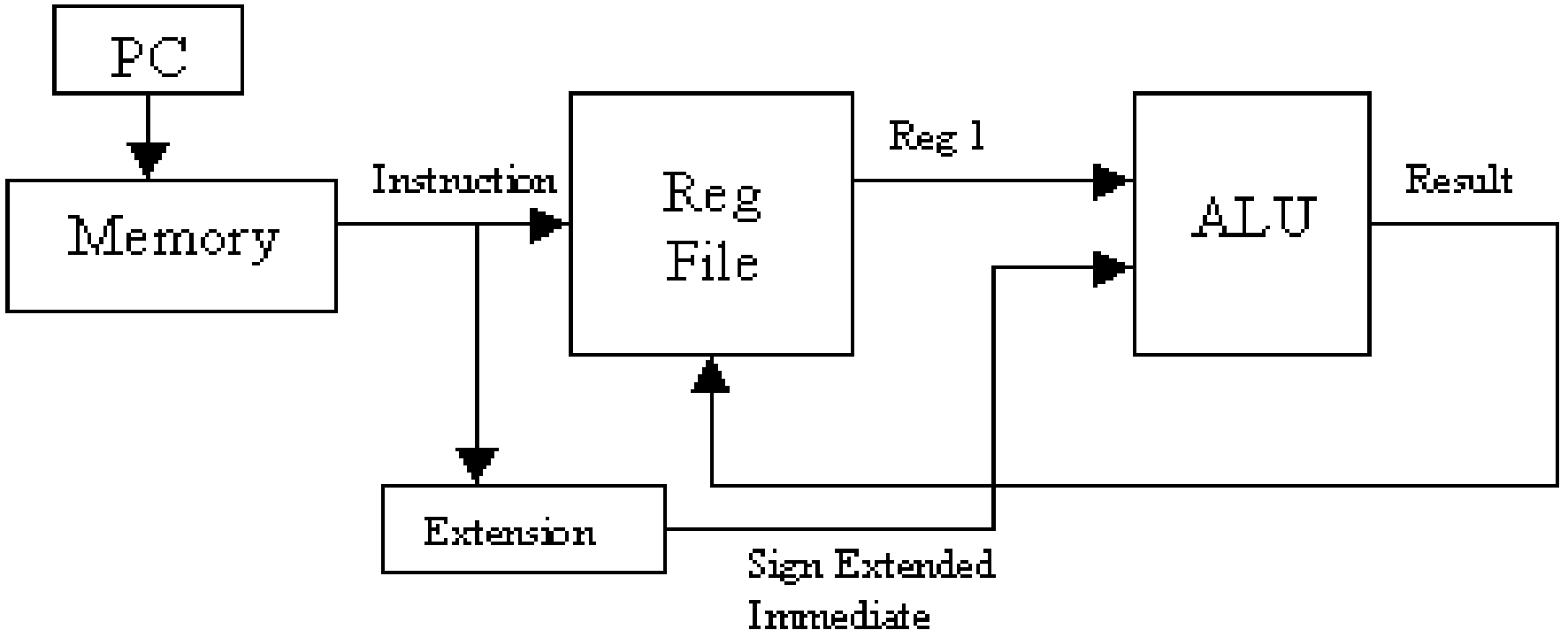


图 8-7 RI-型 ALU 指令的数据通路

8.3 数据通路设计

8.3.3装入字数据通路

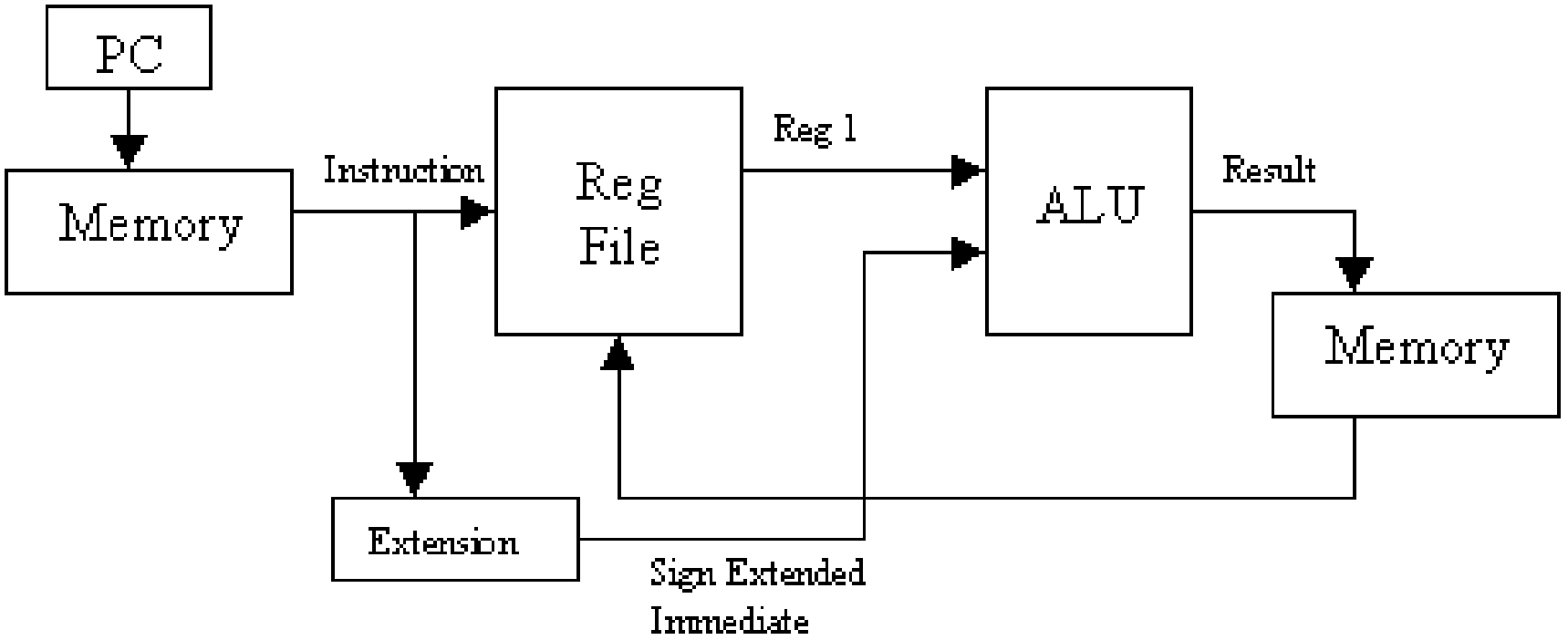


图8-8 装入字数据通路

8.3 数据通路设计

8.3.4存储字数据通路

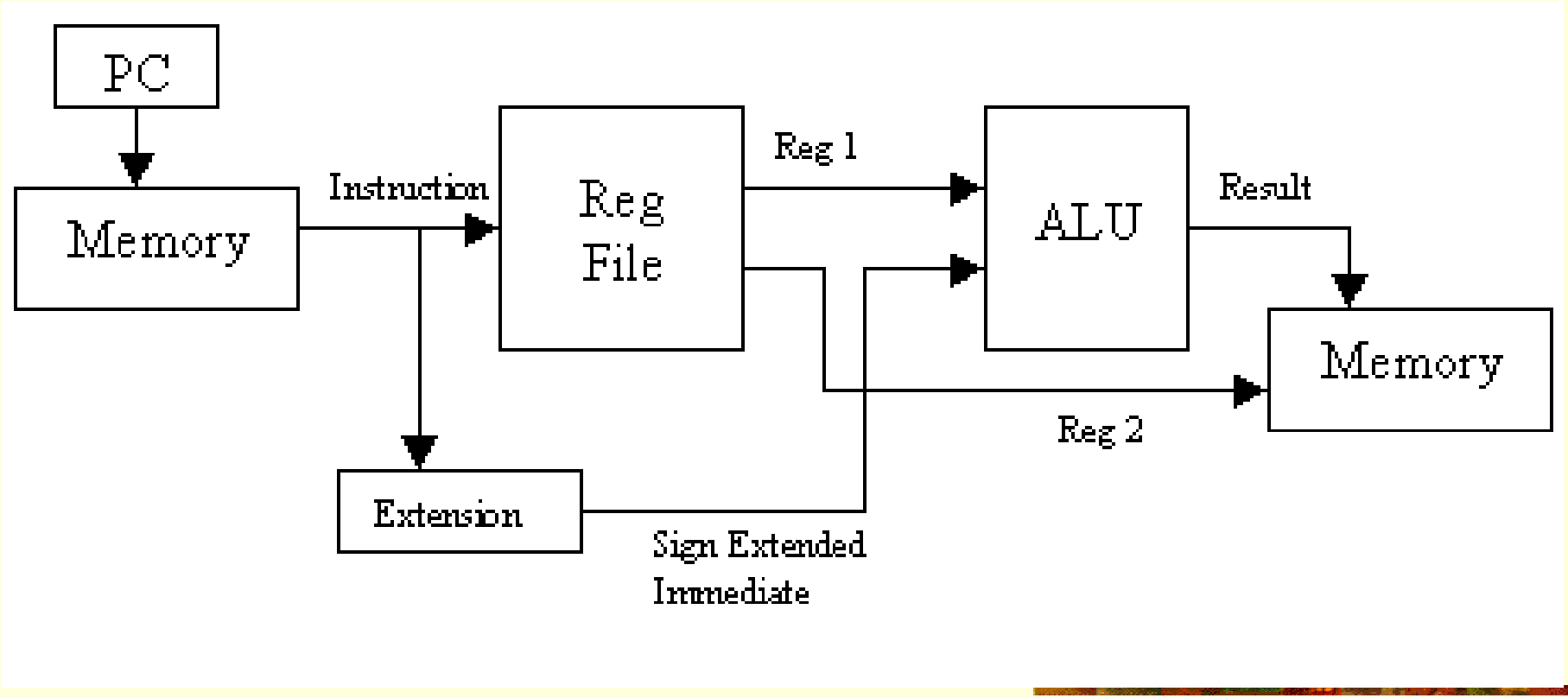


图8-9 存储指令的数据通路

8.3 数据通路设计

8.3.5寄存器转移数据通路

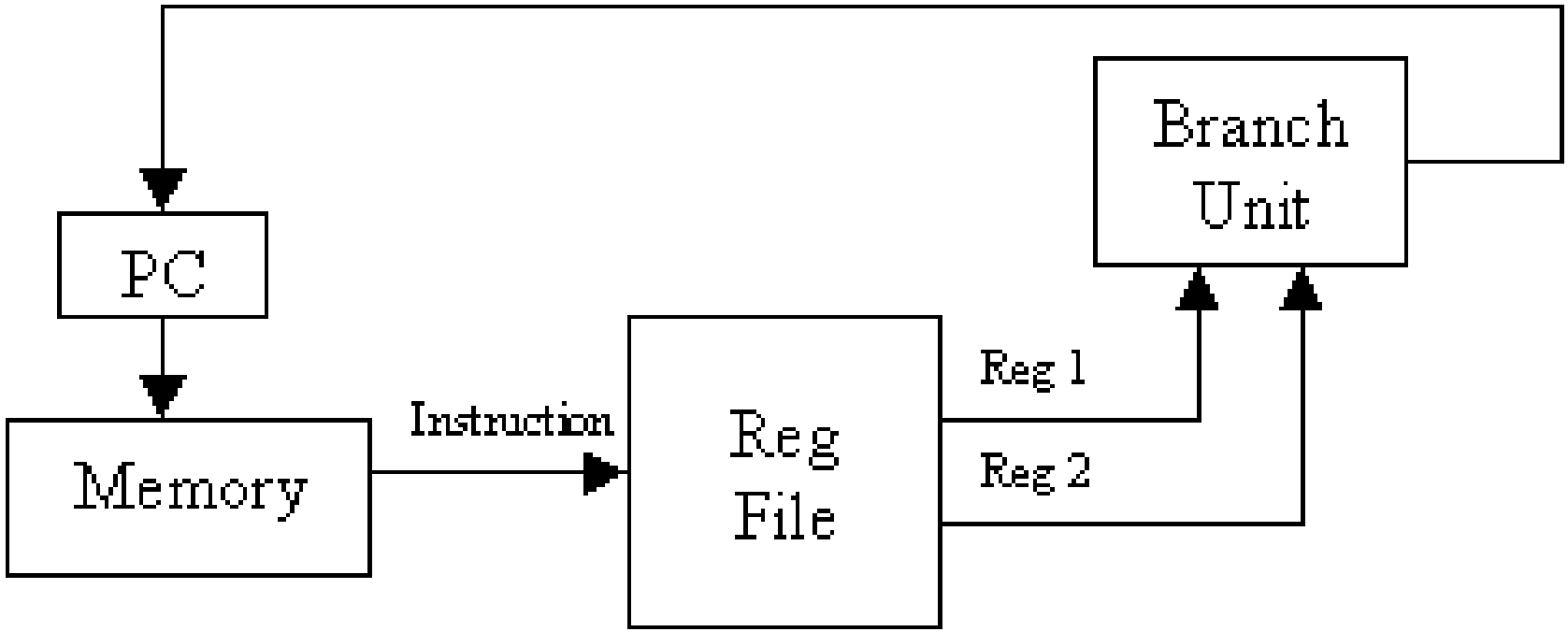


图8-10 转移指令数据通路

8.4流水线各段设计和功能描述

8.4.1 Stage 1取指令段

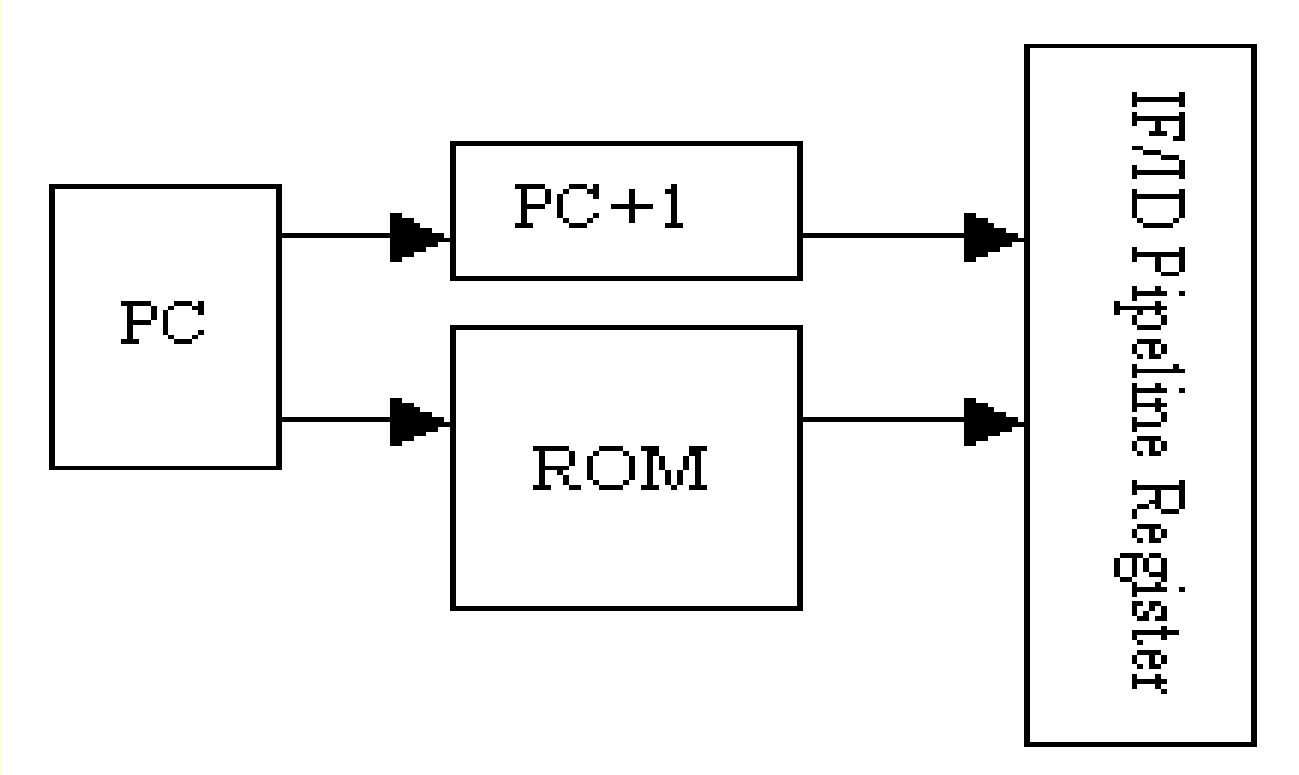


图 8-11 IF Stage 1的结构

8.4流水线各段设计和功能描述

8.4.1 Stage 1取指令段

1. 功能描述

（1）取指令及锁存。

（2）地址计算。

（3）检验指令的合法性。

（4）同步控制。

2. 模块划分和实现

【例8-1】

```
library ieee;
use ieee.std_logic_arith.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_unsigned.all;
ENTITY pcselector IS
   PORT(nextpc, branchpc, retpc, retipc   : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
        sel                       : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        newpc                     : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
END ENTITY pcselector;
ARCHITECTURE sel_behav OF pcselector IS
name : PROCESS(sel) IS
BEGIN
CASE sel IS
WHEN "0000" =>  newpc <= nextpc;
WHEN "0001" =>  newpc <= branchpc;
WHEN "0010" =>  newpc <= retpc;
WHEN "0011" =>  newpc <= retipc;
WHEN "0100" =>  newpc <= "1111111111111111";
WHEN "0101" =>  newpc <= "1111111111110000";
WHEN "0110" =>  newpc <= "0000000000001000";
WHEN "0111" =>  newpc <= "0000000000001100";
WHEN "1000" =>  newpc <= "0000000000001100";
WHEN "1001" =>  newpc <= "0000000000001110";
WHEN "1010" =>  newpc <= "0000000000010000";
WHEN OTHERS =>  newpc <= "0000000000010010";
END CASE;
END PROCESS name;
END ARCHITECTURE sel_behav;
```

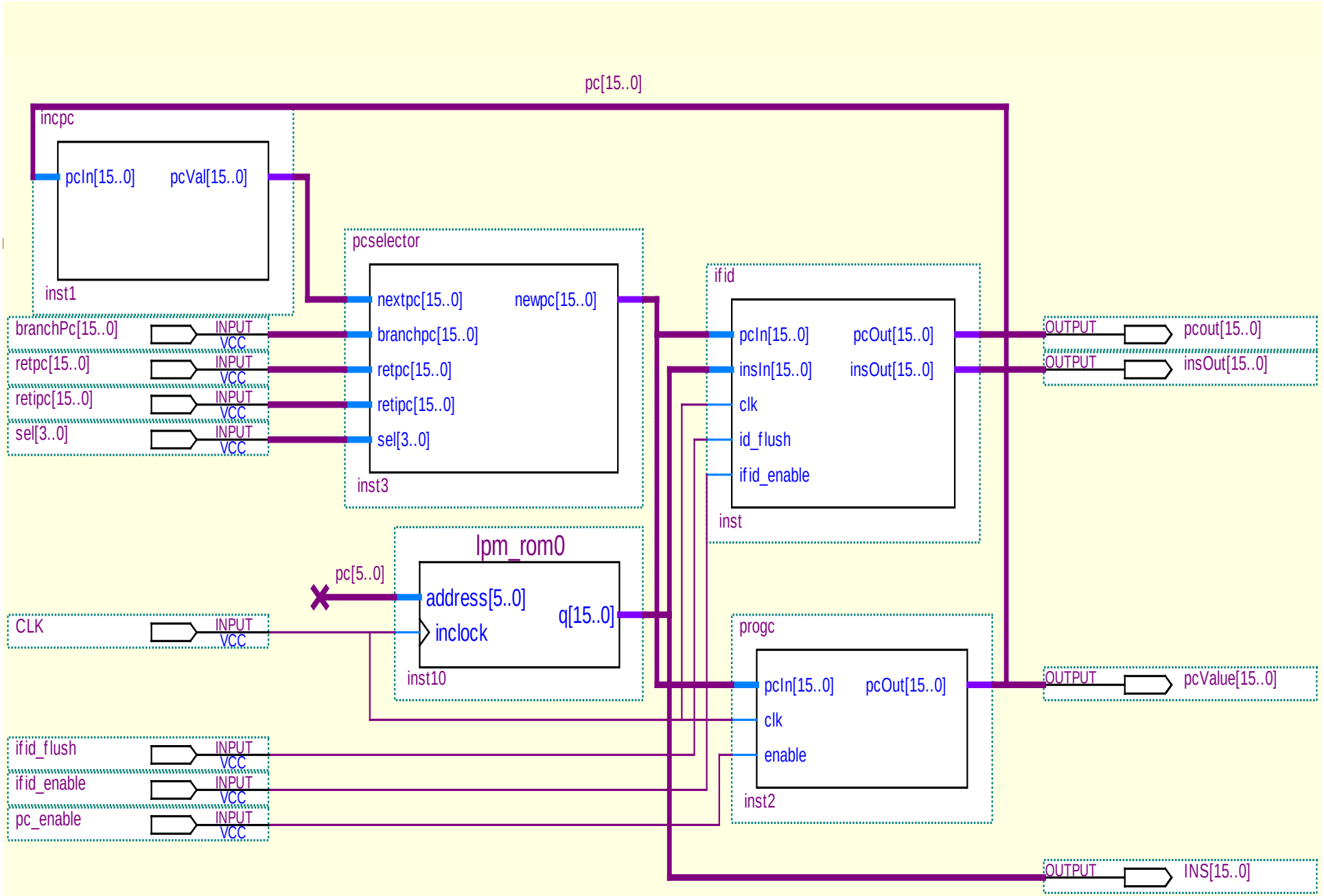


图8-12 Stage 1取指令段的各模块信号连接电路

8.4流水线各段设计和功能描述

8.4.1 Stage 1取指令段

2. 模块划分和实现

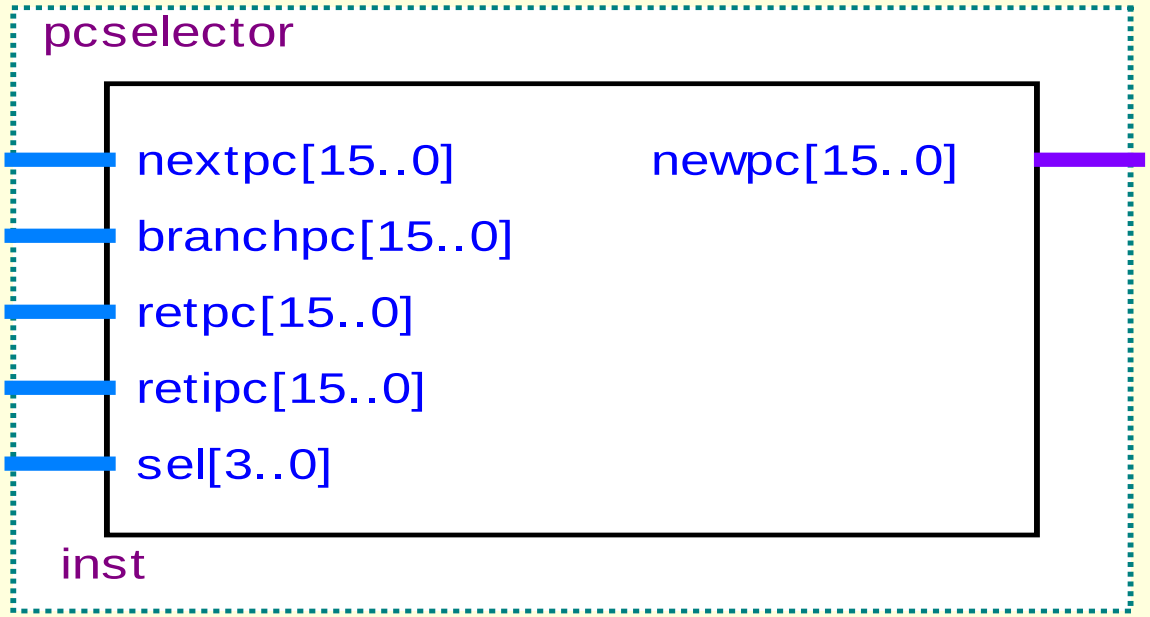


图8-13 pcselector的实体结构

8.4流水线各段设计和功能描述

8.4.1 Stage 1取指令段

2. 模块划分和实现

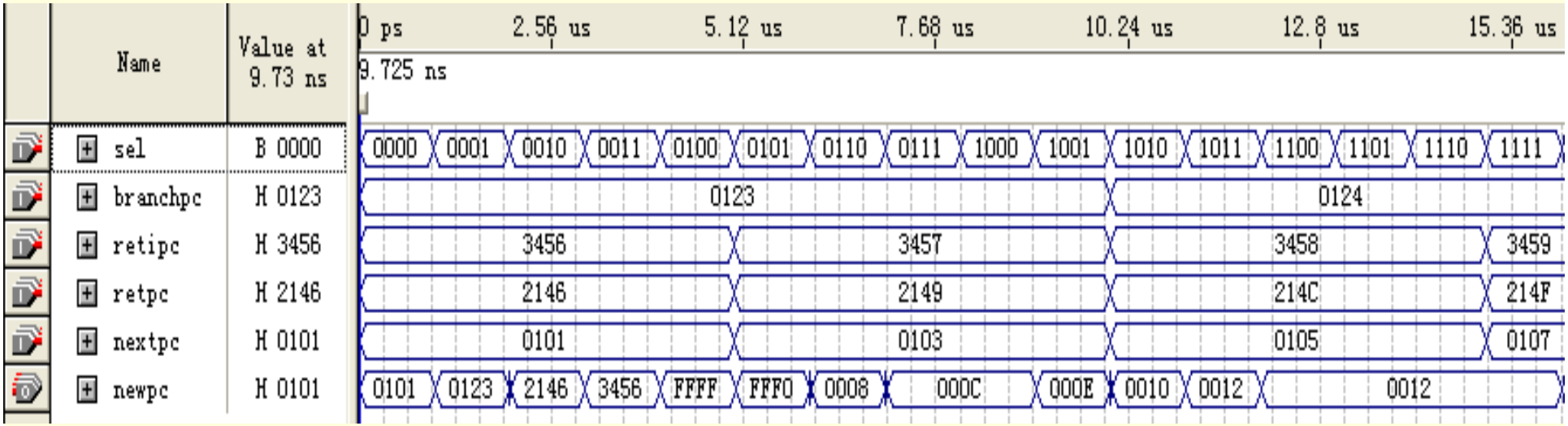


图8-14 pcselector的时序仿真波形图

【例8-2】

```
library ieee;
use ieee.std_logic_1164.all;
ENTITY progc IS
  PORT(pcIn  : IN STD_LOGIC_VECTOR(15 DOWNTO 0); clk,enable : IN
STD_LOGIC;
       pcOut : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
END ENTITY progc;
ARCHITECTURE progc_behav OF progc IS
BEGIN
   name : PROCESS(clk) IS
   VARIABLE regValue : STD_LOGIC_VECTOR(15 DOWNTO 0);
   BEGIN
      IF(clk='1') THEN
         IF(enable='1') THEN   regValue := pcin;  END IF;
      END IF;
      pcOut <= regValue;
   END PROCESS name;
END ARCHITECTURE progc_behav;
```

8.4流水线各段设计和功能描述

8.4.1 Stage 1取指令段

2. 模块划分和实现

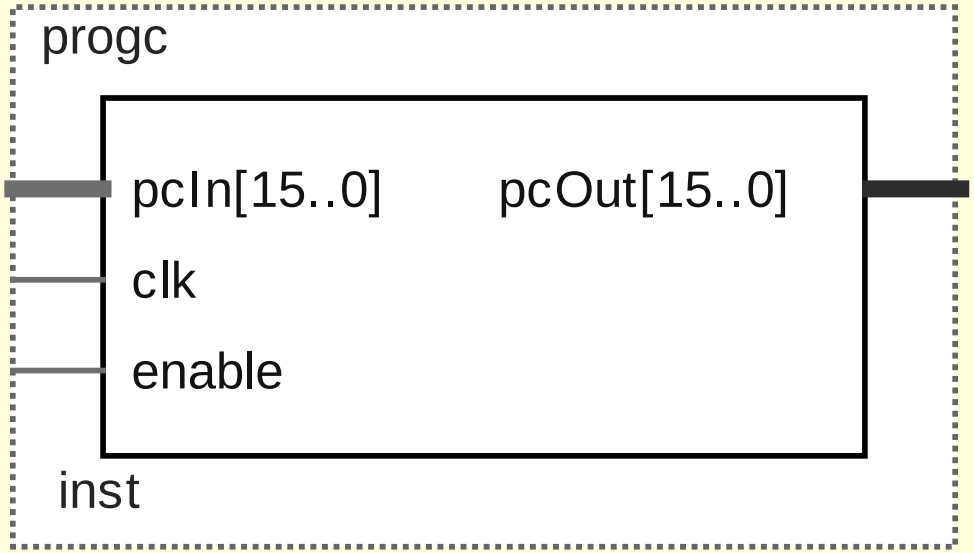


图8-15 progc的实体结构

8.4流水线各段设计和功能描述

8.4.1 Stage 1取指令段

2. 模块划分和实现

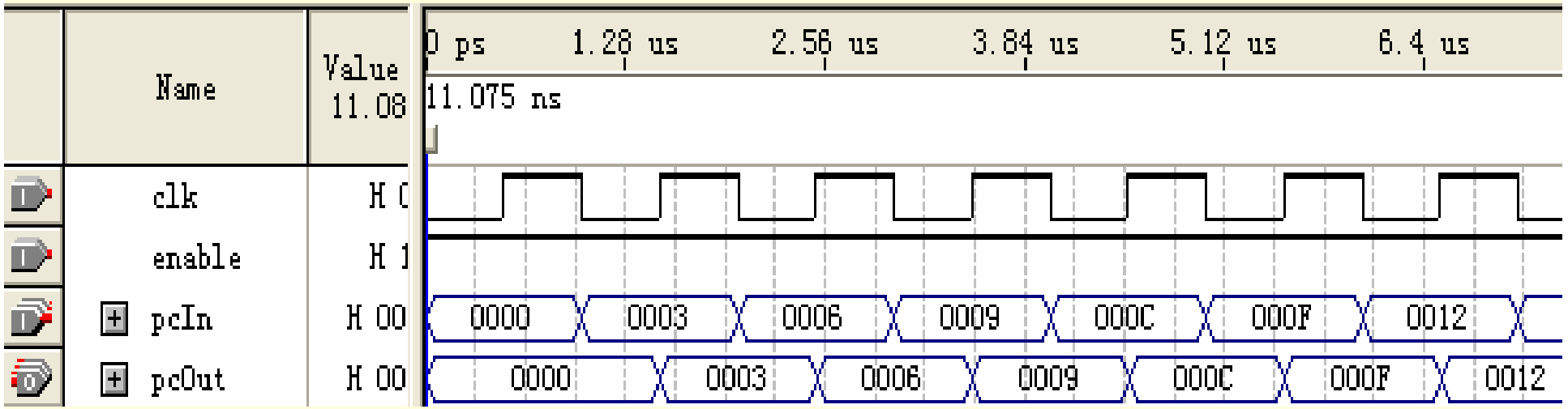


图8-16 progc的时序仿真图

8.4流水线各段设计和功能描述

8.4.1 Stage 1取指令段

2. 模块划分和实现

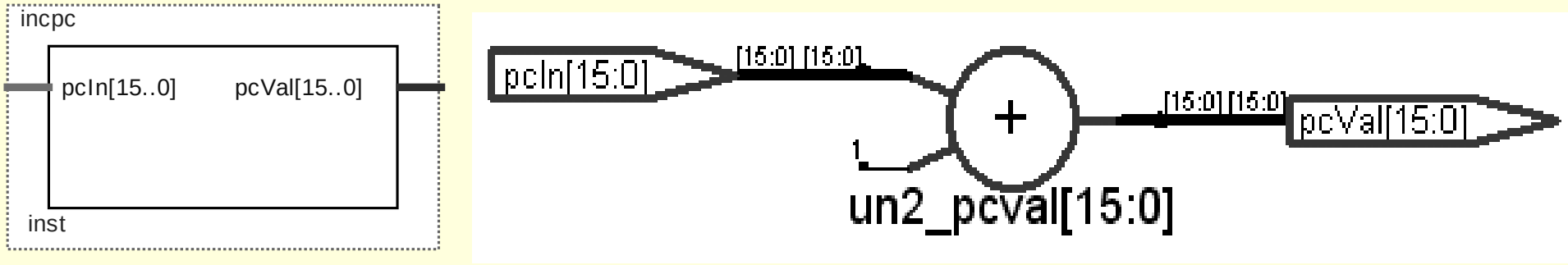
【例8-3】

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
ENTITY incpc IS
    PORT(pcIn  : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
         pcVal  : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
END ENTITY incpc;
ARCHITECTURE pc_behav OF incpc IS
BEGIN
    name : PROCESS(pcIn) IS
    BEGIN
        pcVal <= pcIn +1;
    END PROCESS name;
END ARCHITECTURE pc_behav;
```

8.4流水线各段设计和功能描述

8.4.1 Stage 1取指令段

2. 模块划分和实现



(a) incpc的实体结构　　(b) incpc的RTL结构

图8-17 程序计数器加1模块

8.4流水线各段设计和功能描述

8.4.1 Stage 1取指令段

2. 模块划分和实现

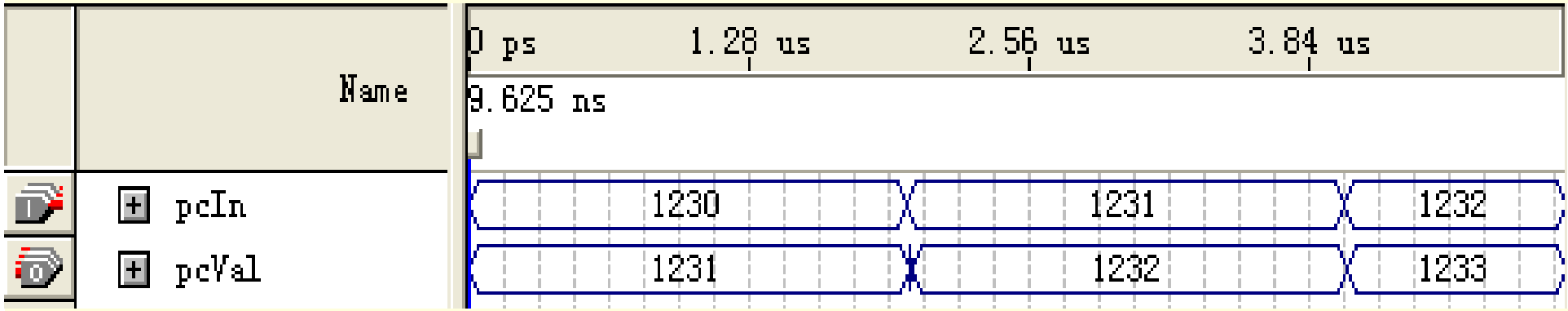


图8-18 程序计数器加1模块的仿真波形

【例8-4】

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY altera_mf;
USE altera_mf.altera_mf_components.all;
ENTITY lpm_rom0 IS
        PORT (address : IN STD_LOGIC_VECTOR (5 DOWNTO 0);
Inclock : IN STD_LOGIC ; q : OUT STD_LOGIC_VECTOR (15 DOWNTO 0) );
END lpm_rom0;
ARCHITECTURE SYN OF lpm_rom0 IS
        SIGNAL sub_wire0          : STD_LOGIC_VECTOR (15 DOWNTO 0);
        COMPONENT altsyncram
        GENERIC ( address_aclr_a : STRING;
                  init_file : STRING;
                  intended_device_family                : STRING;
                  lpm_hint          : STRING;
                  lpm_type          : STRING;
                  numwords_a                : NATURAL;
                  operation_mode            : STRING;
                  outdata_aclr_a            : STRING;
                  outdata_reg_a             : STRING;
                  widthad_a                 : NATURAL;
                  width_a           : NATURAL;
                  width_byteena_a           : NATURAL  );
```

接下页

```
        PORT ( clock0      : IN STD_LOGIC ;
                    address_a          : IN STD_LOGIC_VECTOR (5 DOWNTO 0);
                              q_a      : OUT STD_LOGIC_VECTOR (15 DOWNTO 0) );
        END COMPONENT;
  BEGIN
        q    <= sub_wire0(15 DOWNTO 0);
        altsyncram_component : altsyncram
        GENERIC MAP ( address_aclr_a => "NONE",  init_file => "rom_2.mif",
                    intended_device_family => "Cyclone",
                    lpm_hint                =>              "ENABLE_RUNTIME_MOD=YES,
INSTANCE_NAME=rom3",
                    lpm_type => "altsyncram",  numwords_a => 64,
                    operation_mode => "ROM",  outdata_aclr_a => "NONE",
                    outdata_reg_a => "UNREGISTERED",  widthad_a => 6,
                    width_a => 16,  width_byteena_a => 1  )
        PORT MAP ( clock0 => inclock,address_a => address,q_a => sub_wire0     );
  END SYN;
```

8.4流水线各段设计和功能描述

8.4.1 Stage 1取指令段

2. 模块划分和实现

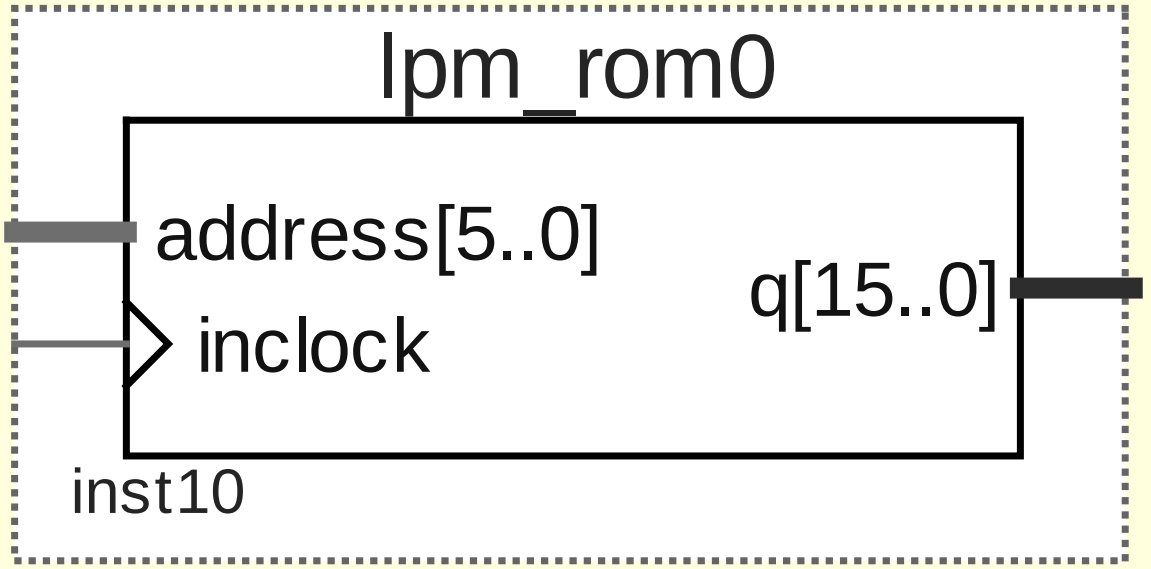


图8-19 lpm_rom的实体结构图

【例8-5】

```
library ieee;
use ieee.std_logic_1164.all;
ENTITY ifid IS
   PORT(pcIn, insIn              : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
        clk,id_flush,ifid_enable      : IN STD_LOGIC;
        pcOut, insOut             : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
END ENTITY ifid;
ARCHITECTURE ifid_behav OF ifid IS
BEGIN
   name : PROCESS(clk) IS
      VARIABLE regValue : STD_LOGIC_VECTOR(31 DOWNTO 0);
   BEGIN
    IF(clk='1') THEN
     IF(ifid_enable='1') THEN
       IF(id_flush='1') THEN
        regValue(31 DOWNTO 0) := regValue(31 DOWNTO 16) & "00000000000000000";
        ELSE regValue(31 DOWNTO 0) := pcIn(15 DOWNTO 0) & insIn(15 DOWNTO 0);
             END IF;
          END IF;
       END IF;
      pcOut <= regValue(31 DOWNTO 16);    insOut <= regValue(15 DOWNTO 0);
   END PROCESS name;
END ARCHITECTURE ifid_behav;
```

8.4流水线各段设计和功能描述

8.4.1 Stage 1取指令段

2. 模块划分和实现

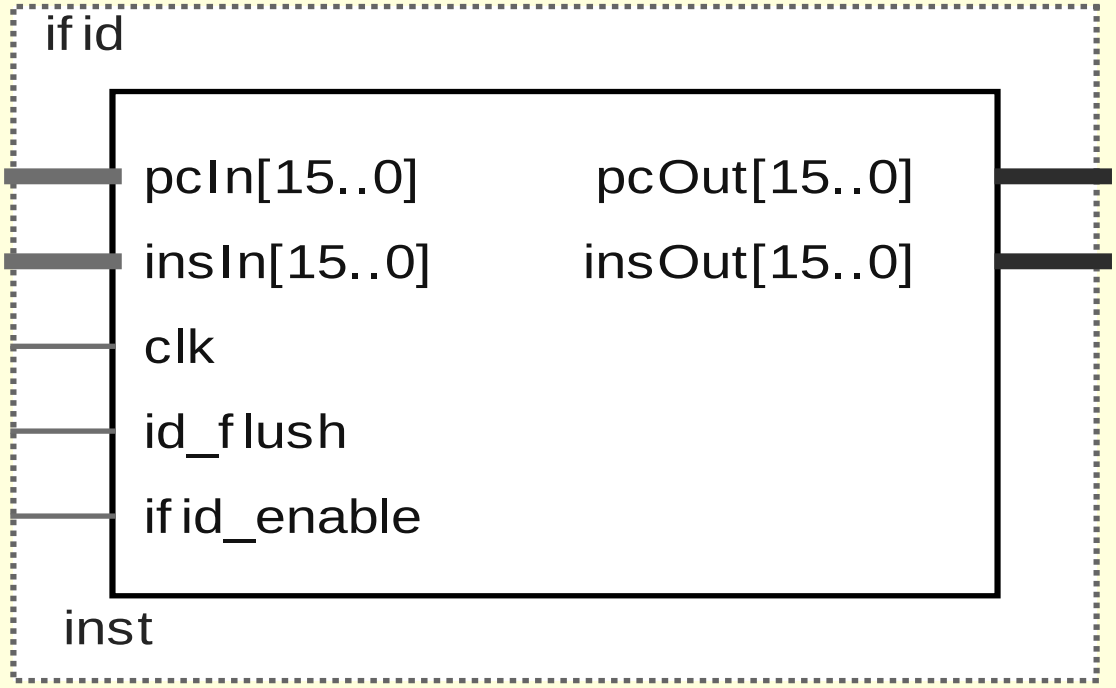


图8-20 ifid的实体结构图

8.4流水线各段设计和功能描述

8.4.1 Stage 1取指令段

2. 模块划分和实现

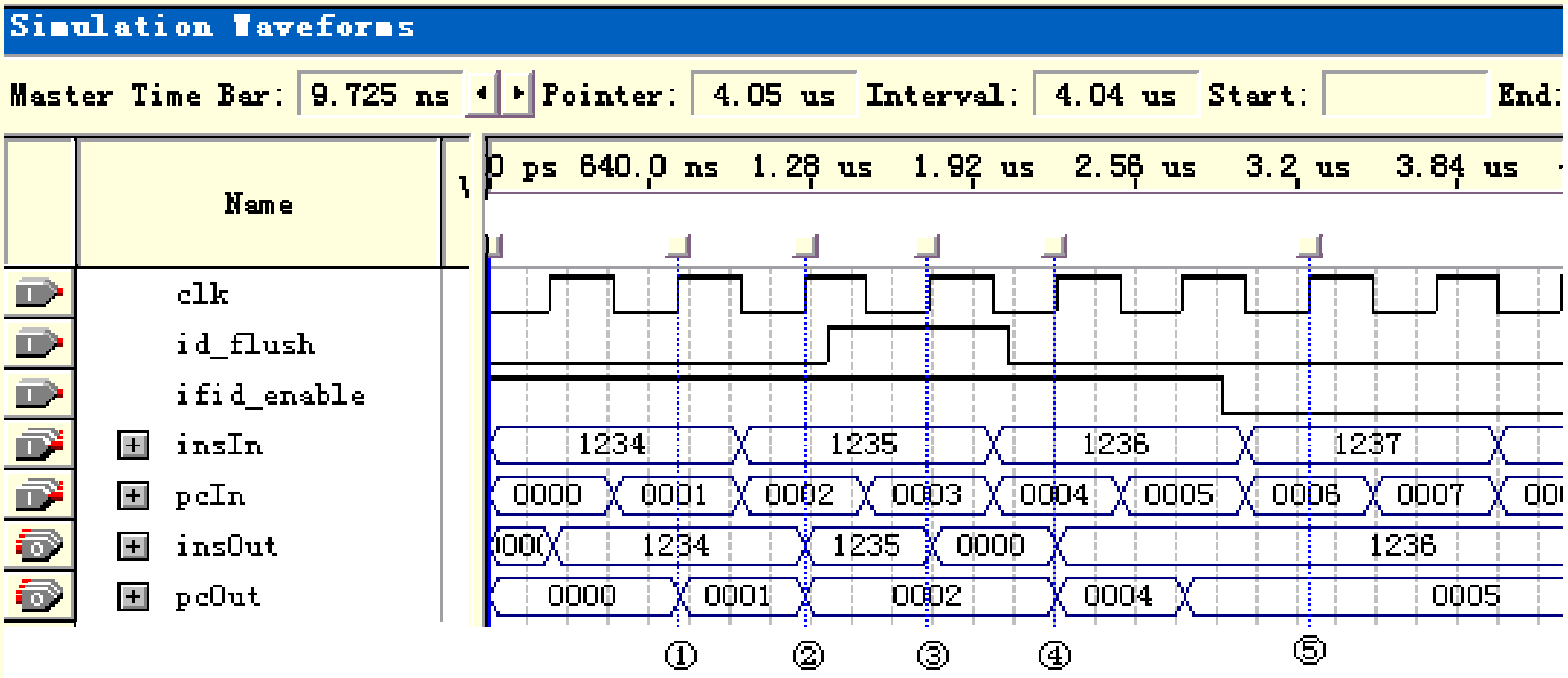


图8-21 ifid流水线寄存器的时序仿真波形图

8.4流水线各段设计和功能描述

8.4.2 Stage 2译码段ID

2. 模块划分和实现

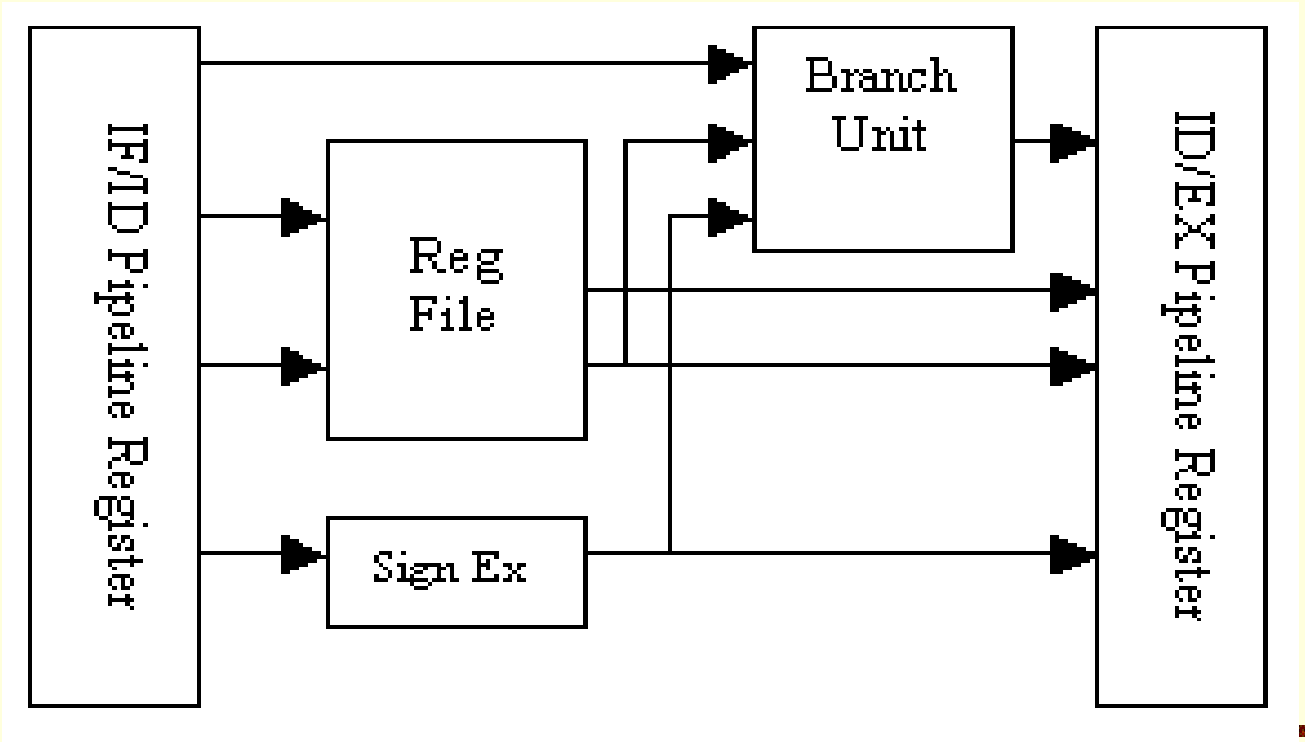


图 8-22 stage 2指令译码段ID的基本结构

8.4流水线各段设计和功能描述

8.4.2 Stage 2译码段ID

1. stage2的功能描述

本段的主要功能如下：

访问寄存器文件。从寄存器文件中读取寄存器操作数并送往id latch段；

向寄存器文件回写数据。把流水线中已经执行完毕的指令所需要回写寄存器文件的执行结果，写入到寄存器文件当中；

符号位扩展。对指令中的8位或6位立即数操作数进行零扩展或者符号位扩展，把扩展后得到的16位操作数送往id latch段；

相关性检测。对正在**ID**段进行处理的指令和流水线中**EXE, MEM, WB**段中的指令进行数据相关检测和控制相关检测。如果检测到数据相关性，就对流水线各段发出**pipeline flush**信号；如果检测到分支指令，就对流水线各段发出**control flush**信号。

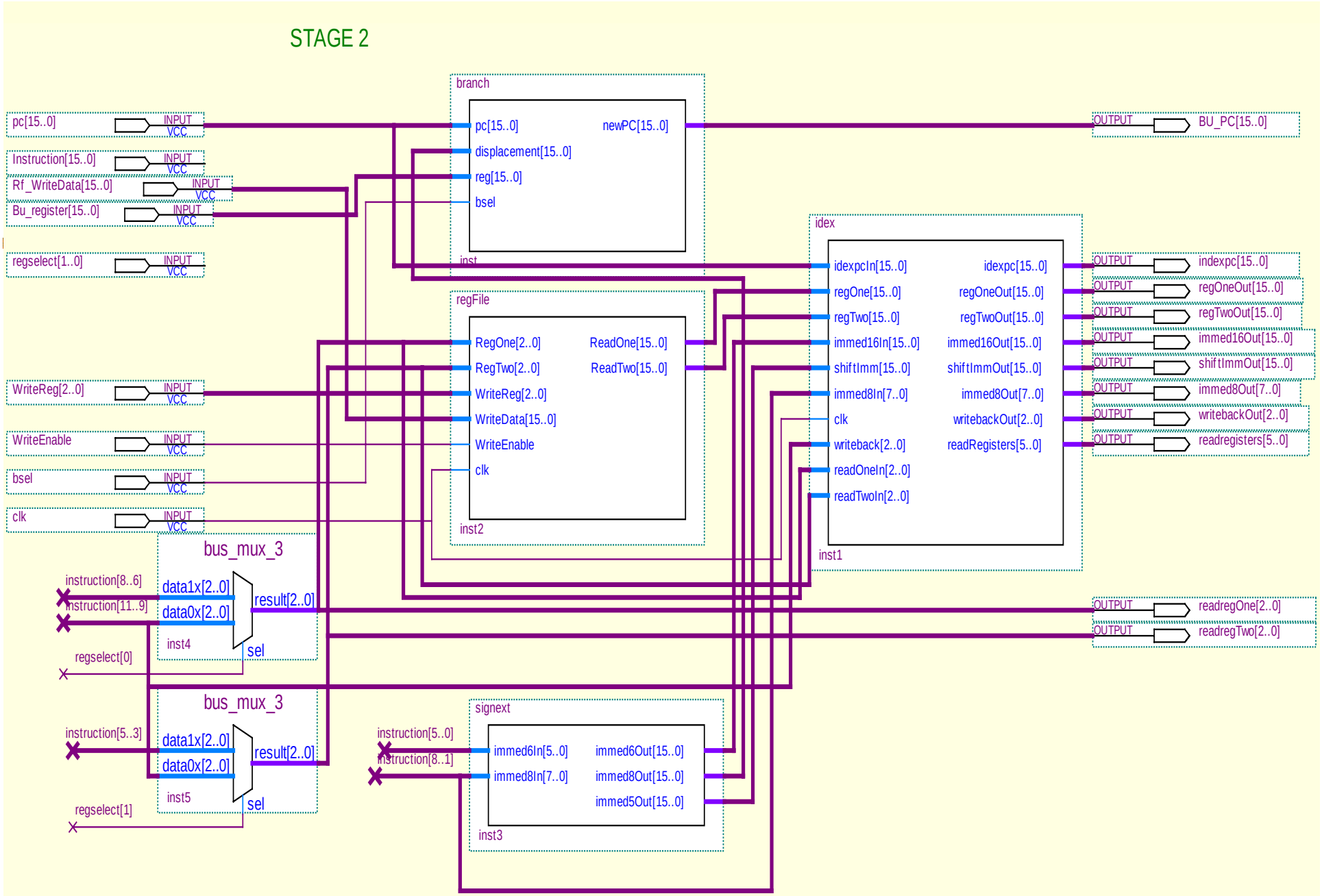


图 8-23 Stage 2 译码段ID的各模块信号连接电路

【例8-6】

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
ENTITY signext IS
   PORT(immed6In  : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
        immed8In   : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        immed6Out, immed8Out,immed5Out : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
END ENTITY signext;
ARCHITECTURE sign_behav OF signext IS
BEGIN
   name: PROCESS(immed6In, immed8In) IS
   VARIABLE tempInt : INTEGER;
   BEGIN
   IF(immed6In(5)='0') THEN immed6Out<="0000000000" & immed6In(5 DOWNTO 0);
      ELSE immed6Out <= "1111111111" & immed6In(5 DOWNTO 0); END IF;
   IF(immed8In(7)='0') THEN immed8Out<="00000000" & immed8In(7 DOWNTO 0);
      ELSE   immed8Out <= "11111111" & immed8In(7 DOWNTO 0); END IF;
   IF(immed8In(4)='0') THEN immed5Out<="00000000000" & immed8In(5 DOWNTO 1);
      ELSE  immed5Out <= "11111111111" & immed8In(5 DOWNTO 1); END IF;
   END PROCESS name;
END ARCHITECTURE sign_behav;
```

【例8-7】

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
ENTITY regFile IS
   port(RegOne, RegTwo, WriteReg  : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
        WriteData    : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
        WriteEnable  : IN STD_LOGIC;
        clk          : IN STD_LOGIC;
        ReadOne, ReadTwo  : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
END ENTITY regFile;
ARCHITECTURE behav OF regFile IS
BEGIN
   reg : PROCESS (clk) IS
      TYPE regArray IS ARRAY (INTEGER RANGE 0 TO 7) OF
                STD_LOGIC_VECTOR(15 downto 0);
      VARIABLE register_file : regArray;
   BEGIN
     IF(clk='1') THEN
      IF (WriteEnable = '1') THEN
       register_file(CONV_INTEGER(UNSIGNED(WriteReg))) := WriteData;
         register_file(0) := "0000000000000000";  END IF;
```

(接下页）

```
END IF;
    IF(WriteEnable='1') THEN
     IF(WriteReg = RegOne) THEN   ReadOne <= WriteData;
      ELSE  ReadOne <= register_file(CONV_INTEGER(UNSIGNED(RegOne)));
        END IF;
    IF(WriteReg = RegTwo) THEN  ReadTwo <= WriteData;
     ELSE  ReadTwo <= register_file(CONV_INTEGER(UNSIGNED(RegTwo)));
      END IF;
      ELSE  ReadOne <= register_file(CONV_INTEGER(UNSIGNED(RegOne)));
        ReadTwo <= register_file(CONV_INTEGER(UNSIGNED(RegTwo)));
   END IF;
 END PROCESS reg;
END ARCHITECTURE behav;
```

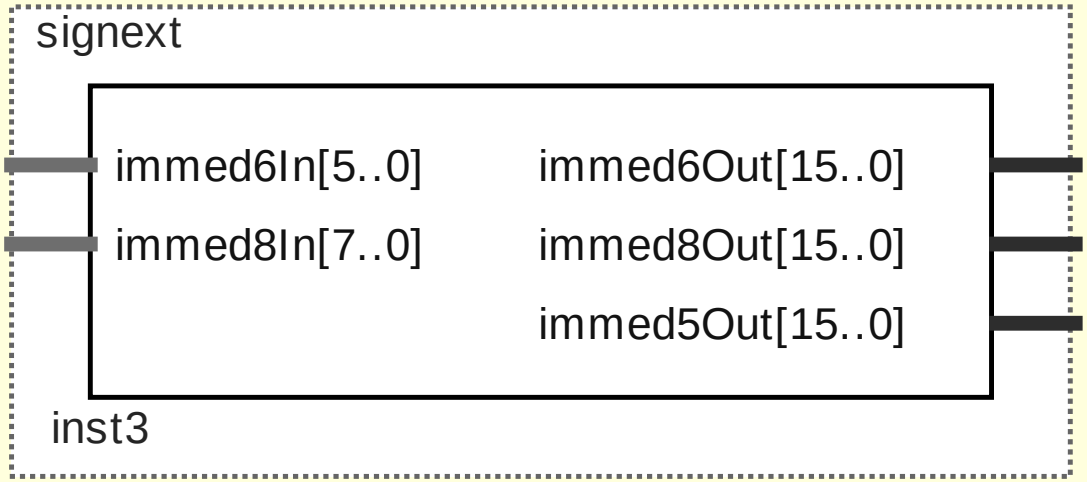


图8-24 符号扩展模块signext的结构图

8.4流水线各段设计和功能描述

8.4.2 Stage 2译码段ID

2. 模块划分和实现

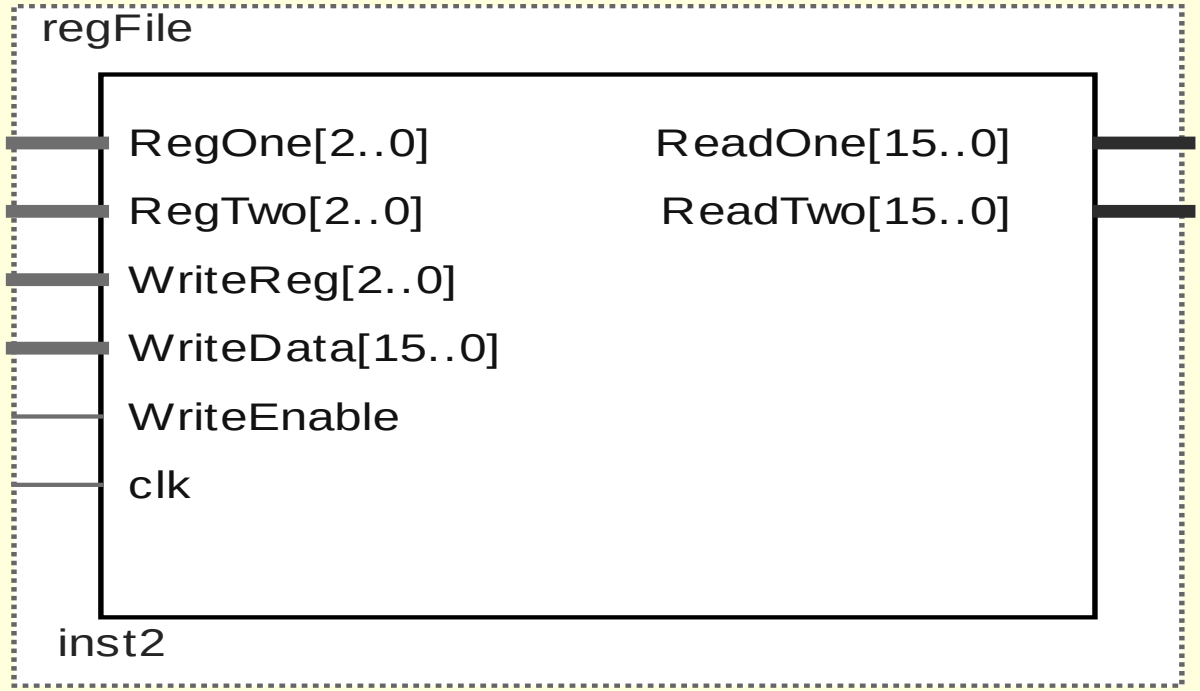


图8-25　寄存器文件模块regFile的实体结构图

8.4流水线各段设计和功能描述

8.4.2 Stage 2译码段ID

2. 模块划分和实现

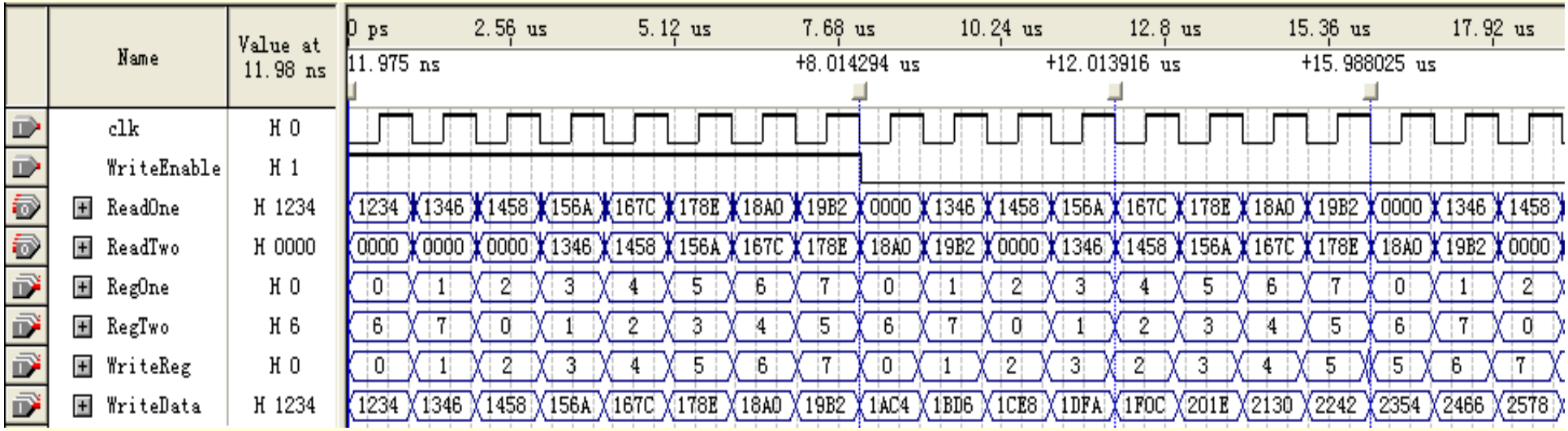


图8-26　regFile的时序仿真图

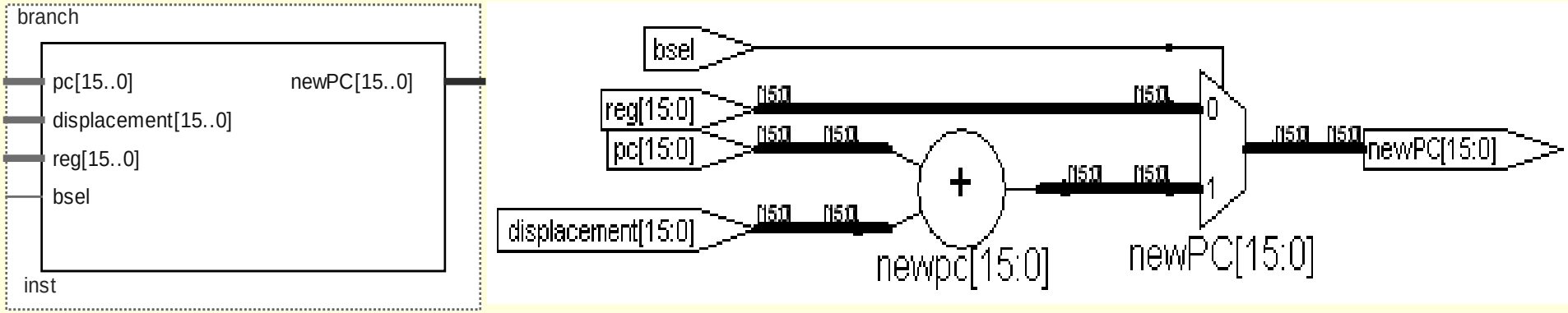
【例8-8】

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_unsigned.all;
ENTITY branch IS
   PORT(pc, displacement, reg  : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
        bsel  : IN STD_LOGIC;
        newPC  : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
END ENTITY branch;
ARCHITECTURE branch_behav OF branch IS
BEGIN
   name : PROCESS(pc, displacement,reg,bsel) IS
   BEGIN
      IF(bsel='1') THEN  newPC <= pc + displacement;
      ELSE   newPC <= reg;   END IF;
   END PROCESS name;
END ARCHITECTURE branch_behav;
```

8.4流水线各段设计和功能描述

8.4.2 Stage 2译码段ID

2. 模块划分和实现



(a) 实体结构图　　(b) RTL结构图

图8-27 分支转移branch模块

8.4流水线各段设计和功能描述

8.4.2 Stage 2译码段ID

2. 模块划分和实现

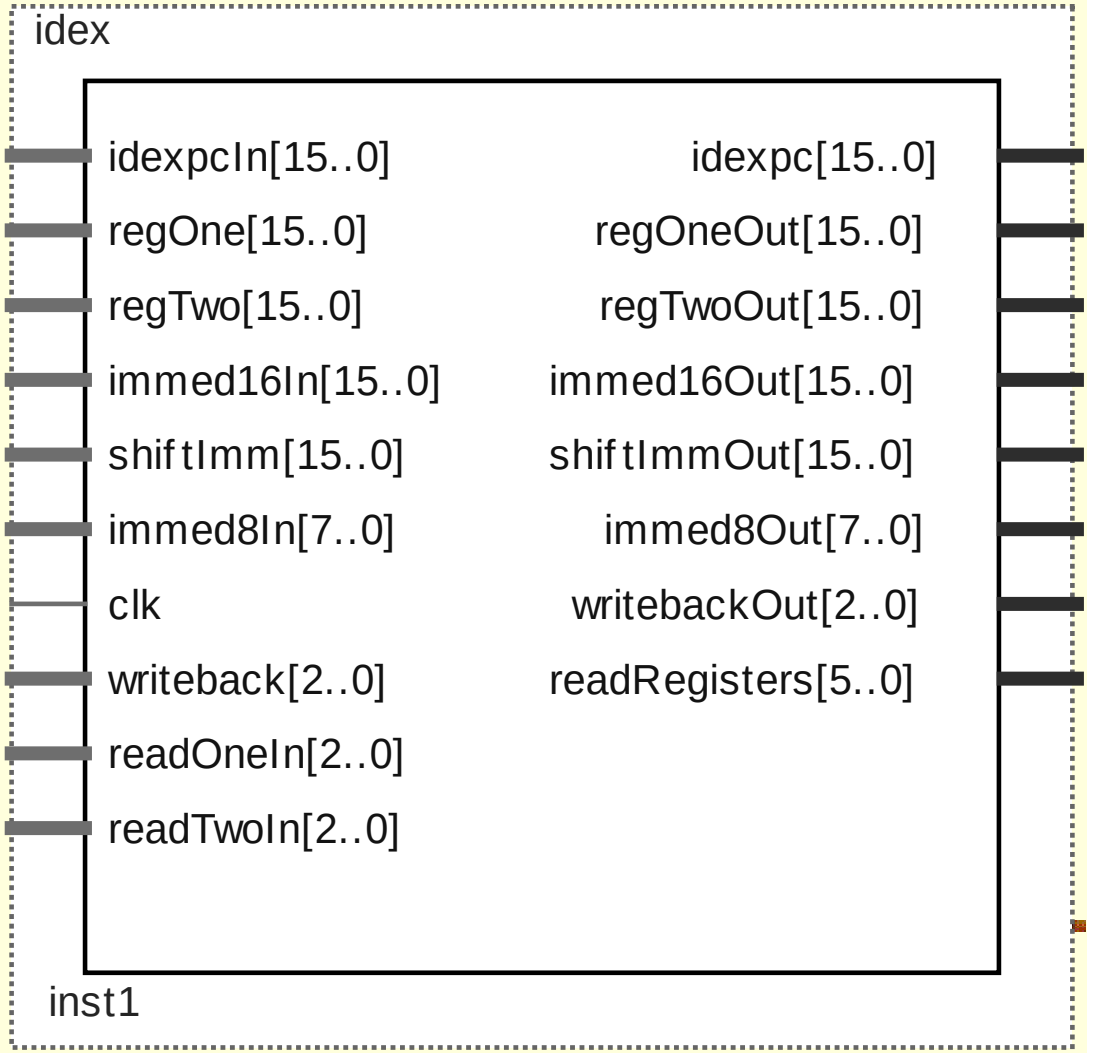


图8-28 stage 2的idex结构图

8.4流水线各段设计和功能描述

8.4.2 Stage 2译码段ID

2. 模块划分和实现

【例8-9】

```
library ieee;
use ieee.std_logic_1164.all;
ENTITY idex IS
 PORT(idexpcIn,regOne,  regTwo,  immed16In  :  IN  STD_LOGIC_VECTOR(15
DOWNTO 0);
      shiftImm  : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
      immed8In  : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
      clk    : IN STD_LOGIC;
      writeback  : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
      readOneIn, readTwoIn      : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
      idexpc,regOneOut, regTwoOut  : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
      immed16Out, shiftImmOut     : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
      immed8Out                   : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
```

（接下页）

```
   writebackOut                : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
         readRegisters                : OUT STD_LOGIC_VECTOR(5 DOWNTO 0));
   END ENTITY idex;
   ARCHITECTURE idex_behav OF idex IS
   BEGIN
      name : PROCESS(clk) IS
         VARIABLE regValue : STD_LOGIC_VECTOR(96 DOWNTO 0);
      BEGIN
       IF(clk='1') THEN regValue(96 DOWNTO 0) := idexpcIn(15 DOWNTO 0) &
        readOneIn(2 DOWNTO 0) & readTwoIn(2 DOWNTO 0) &
        shiftImm(15 DOWNTO 0) & regOne(15 DOWNTO 0) & regTwo(15 DOWNTO 0) &
        immed16In(15 DOWNTO 0) & immed8In(7 DOWNTO 0) & writeback(2 DOWNTO
0);
        END IF;
        idexpc<=regValue(96 DOWNTO 81); readRegisters<=regValue(80 DOWNTO 75);
          shiftImmOut <= regValue(74 DOWNTO 59);
       regOneOut<=regValue(58 DOWNTO 43); regTwoOut<=regValue(42 DOWNTO 27);
      immed16Out<=regValue(26 DOWNTO 11); immed8Out<=regValue(10 DOWNTO 3);
         writebackOut <= regValue(2 DOWNTO 0);
      END PROCESS name;
   END ARCHITECTURE idex_behav;
```

8.4流水线各段设计和功能描述

8.4.3 Stage 3执行有效地址计算段(EXE)

1. 功能描述

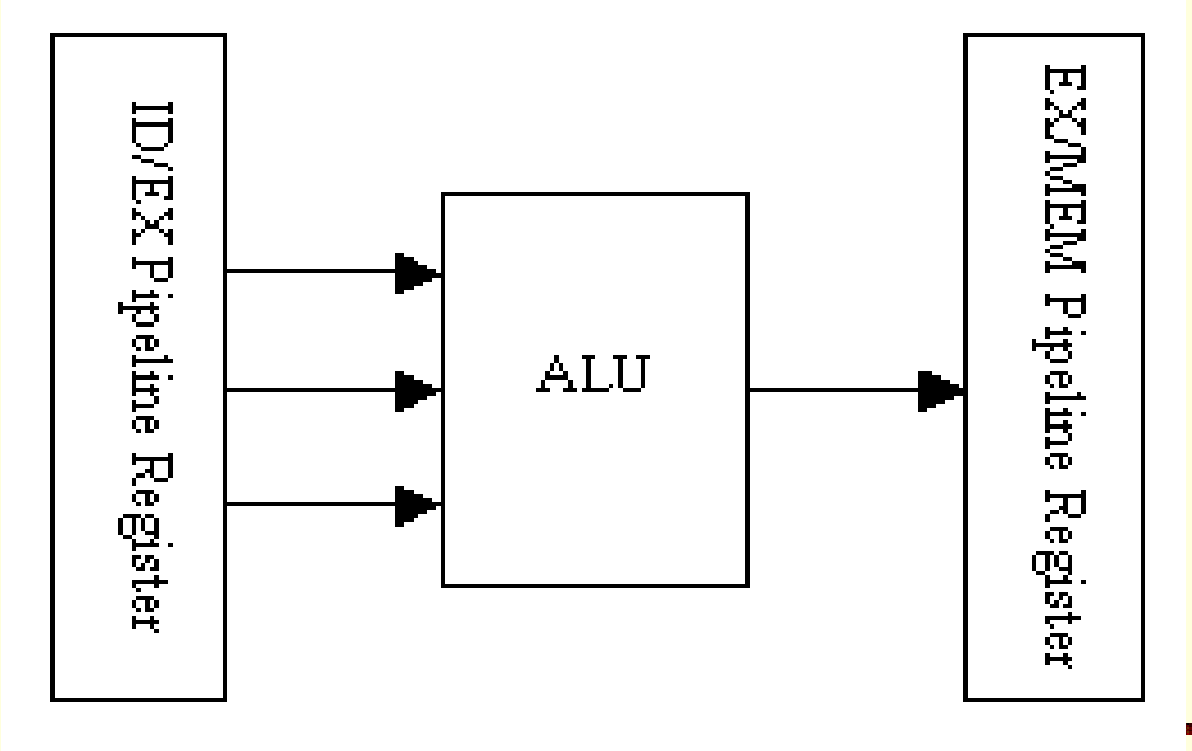


图 8-29 stage 3执行EXE段的基本结构

2. 模块划分和实现

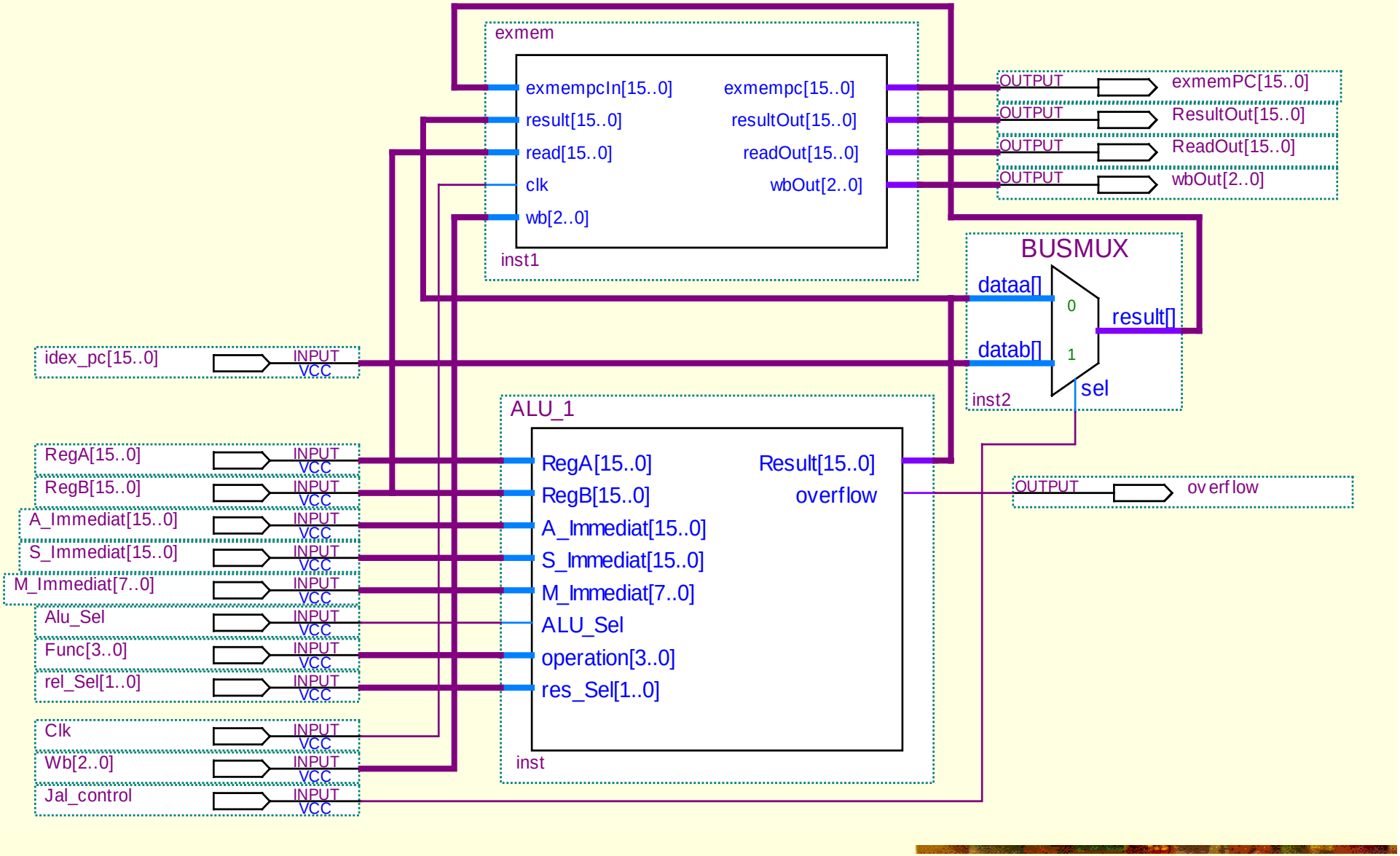


图 8-30 Stage 3 EXE执行段的各模块信号连接电路

【例8-10】

```
library ieee;
use ieee.std_logic_1164.all;
ENTITY exmem IS
   PORT(exmempcIn, result, read    : STD_LOGIC_VECTOR(15 DOWNTO 0);
     clk  : IN STD_LOGIC;
     wb   : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
     exmempc, resultOut, readOut  : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
     wbOut   : OUT STD_LOGIC_VECTOR(2 DOWNTO 0));
END ENTITY exmem;
ARCHITECTURE exmem_behav OF exmem IS
BEGIN
   name : PROCESS(clk) IS
   VARIABLE regValue : STD_LOGIC_VECTOR(50 DOWNTO 0);
   BEGIN
    IF(clk='1') THEN  regValue(50 DOWNTO 0) := exmempcIn(15 DOWNTO 0) &
     result(15 DOWNTO 0) & read(15 DOWNTO 0) & wb(2 DOWNTO 0); END IF;
     exmempc <= regValue(50 DOWNTO 35);    -- EX段的PC值
     resultOut <= regValue(34 DOWNTO 19);  --来自ALU的运算结果
     readOut <= regValue(18 DOWNTO 3);     --从ALU读出的数据
     wbOut <= regValue(2 DOWNTO 0);        --回写的寄存器
   END PROCESS name;
END ARCHITECTURE exmem_behav;
```

8.4流水线各段设计和功能描述

8.4.3 Stage 3执行有效地址计算段(EXE)

2. 模块划分和实现

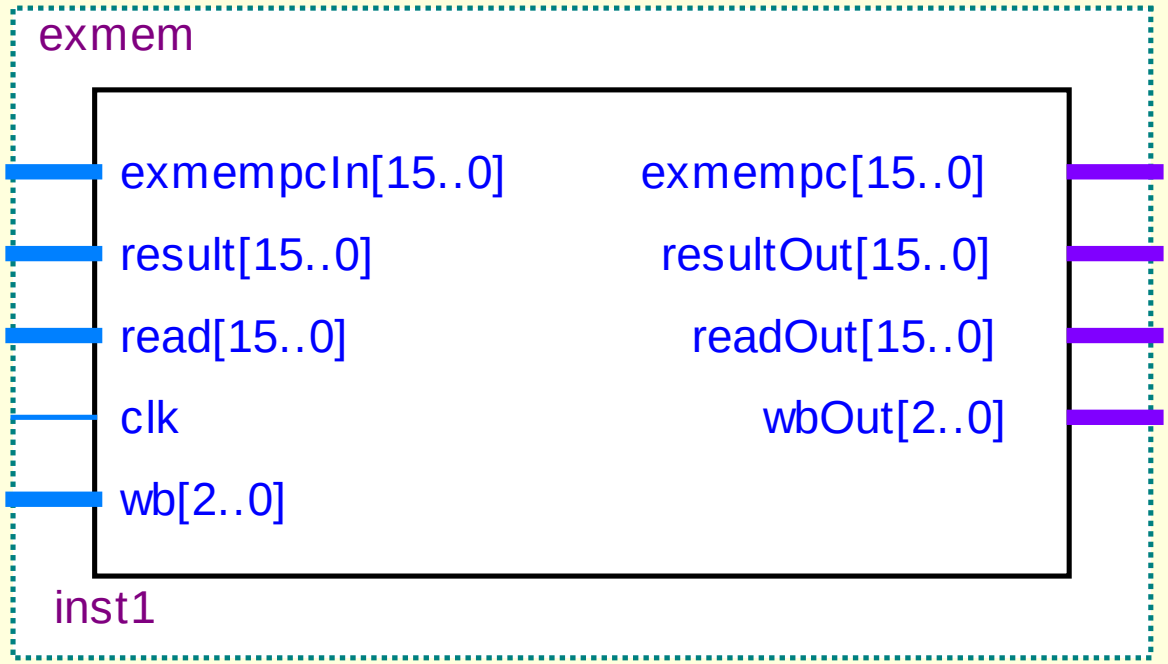


图8-31 Stage 3 EXMEM流水线寄存器

8.4流水线各段设计和功能描述

8.4.3 Stage 3执行有效地址计算段(EXE)

2. 模块划分和实现

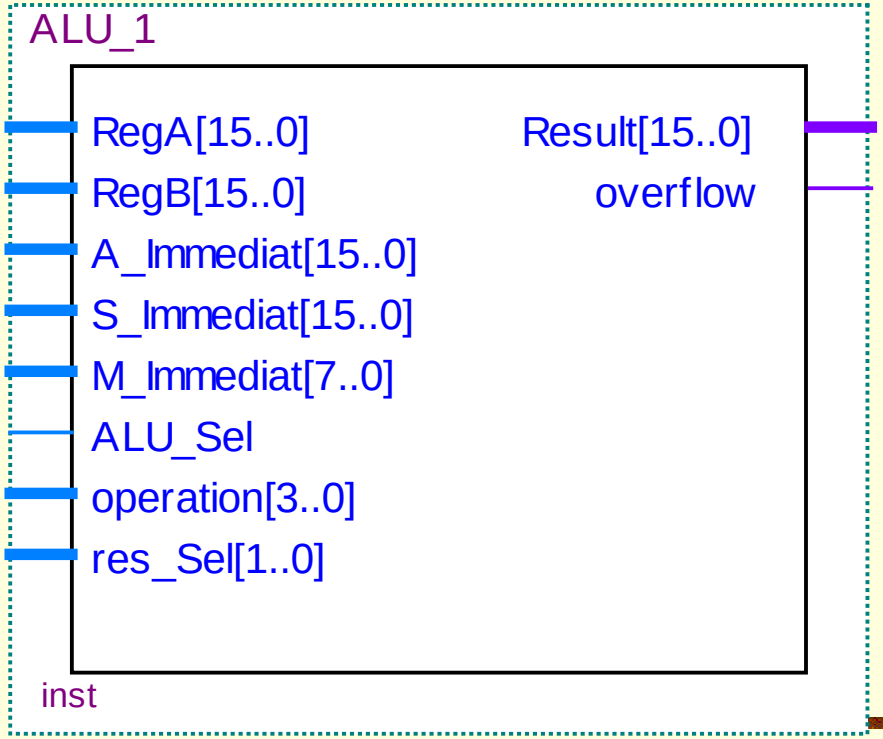


图8-32 Stage 3的运算模块ALU_1

【例8-11】

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_unsigned.all;
ENTITY alu_16 IS
   PORT(a, b   : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
        func    : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        overflow  : OUT STD_LOGIC;
        c   : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
END ENTITY alu_16;
ARCHITECTURE alu_behav OF alu_16 IS
BEGIN
   name : PROCESS(a, b, func) IS
   VARIABLE signedResult          : SIGNED(15 DOWNTO 0);
   VARIABLE unsignedResult        : UNSIGNED(16 DOWNTO 0);
   VARIABLE temp                  : STD_LOGIC_VECTOR(15 DOWNTO 0);
   VARIABLE a1,b1                 : STD_LOGIC_VECTOR(16 DOWNTO 0);
BEGIN
      CASE func IS
         WHEN "0000" =>  c <= a and b; overflow <= '0'; --AND_WORD
         WHEN "0001" => c <= a or b; overflow <= '0'; --OR
         WHEN "0010" => c <= a xor b; overflow <= '0'; --XOR
         WHEN "0011" => c <= a nor b; overflow <= '0'; --NOR
         WHEN "0100" => c <= not a; overflow <= '0'; --NOT
         WHEN "0101" => signedResult := conv_signed(conv_integer(signed(a))
```

(接下页)

```
+conv_integer(signed(b)),16); --ADD
            temp := conv_std_logic_vector(signed(a) + signed(b), 16);
             c <= conv_std_logic_vector(signedResult,16);   overflow <= '0';
          WHEN "0110" =>  signedResult := signed(a) - signed(b); --SUB
             c <= conv_std_logic_vector(signedResult,16);  overflow <= '0';
          WHEN "0111" => unsignedResult := unsigned(a) + unsigned(b);--ADD
a1 := "0" & a ; b1 :="0" & b;
             unsignedResult := unsigned(a1) + unsigned(b1);
             c <= conv_std_logic_vector(unsignedResult,16);
            IF(conv_integer(unsignedResult) >= 65536) then  overflow <= '1';
             ELSE  overflow <= '0';  END IF;
          WHEN "1000" =>  a1 := "0" & a ; b1 :="0" & b; --SUBu
             unsignedResult := unsigned(a1) - unsigned(b1);
           c <=conv_std_logic_vector(unsignedResult,16);  overflow <= '0';
    WHEN "1001" => if(conv_integer(unsigned(a))<=conv_integer(unsigned(b)))
             Then  c <= "0000000000000000";
             else  c <= "0000000000000001";   end if;
             overflow <= '0';
   WHEN "1010" => if(conv_integer(signed(a))<=conv_integer(signed(b)))
Then  c<="0000000000000000";
             else   c<="0000000000000001";  end if;
             overflow <= '0';
          WHEN others =>  c <= a;  overflow <= '0';
       END CASE;
    END PROCESS name;
END ARCHITECTURE alu_behav;
```

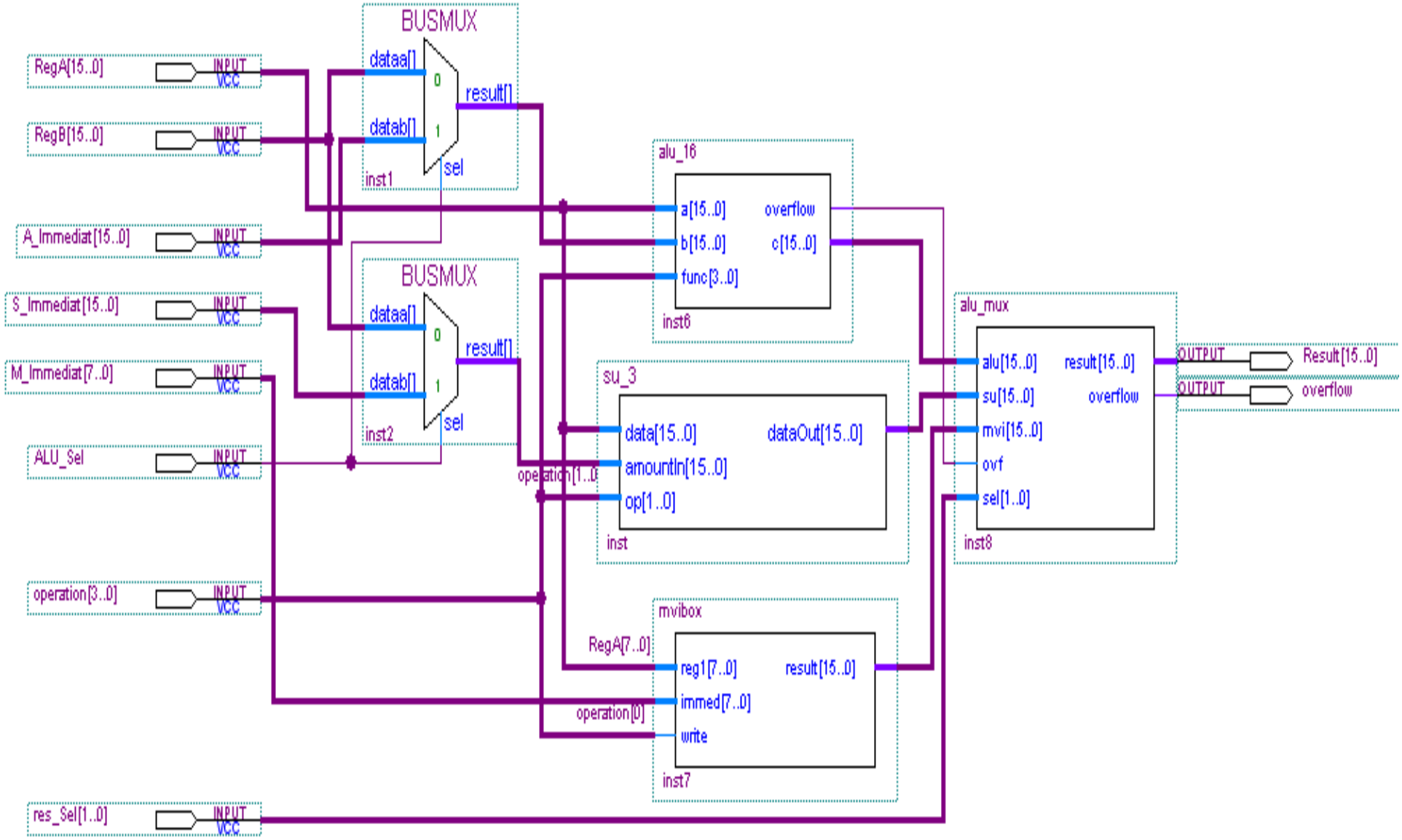


图 8-33　运算模块ALU的内部结构图

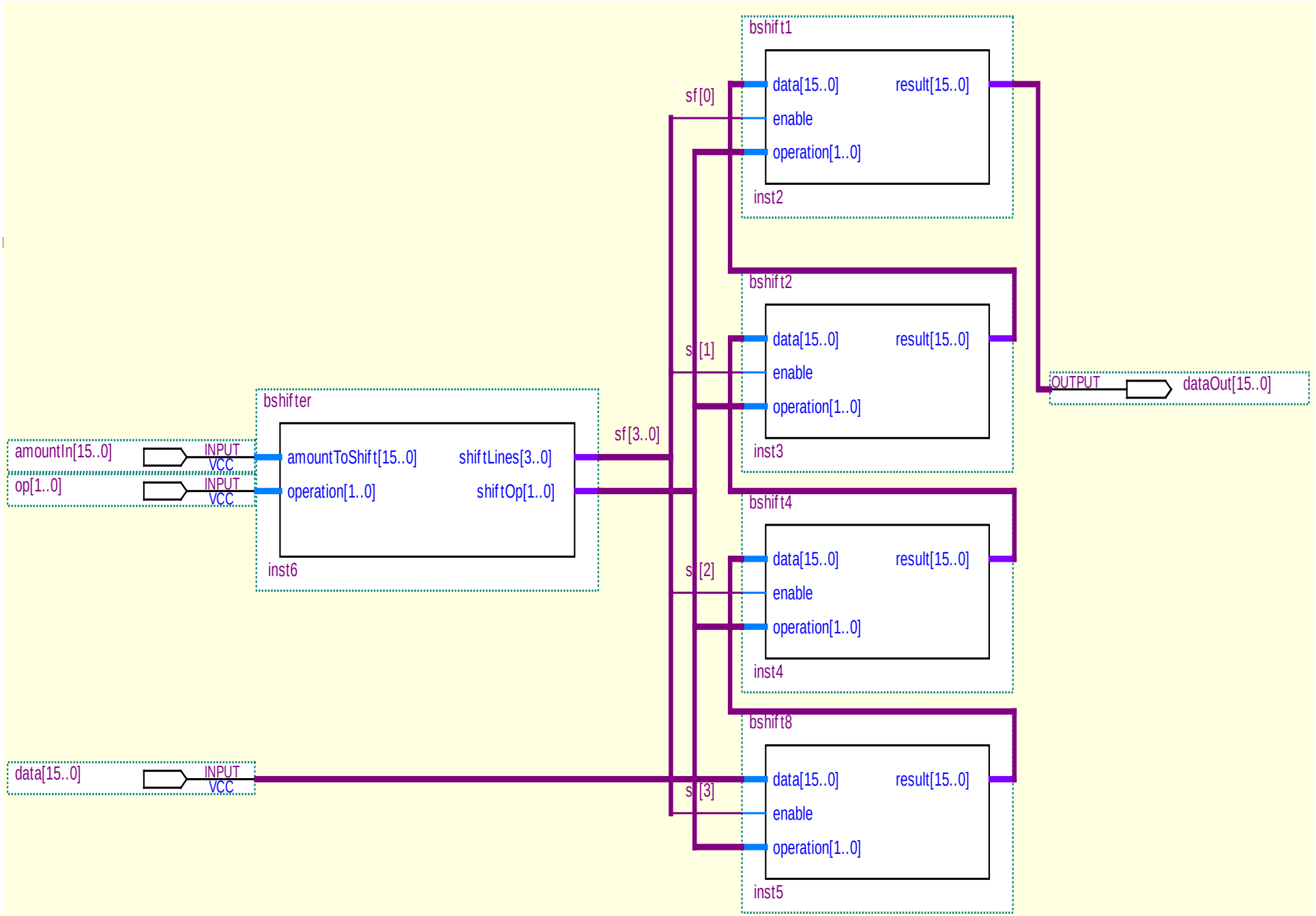


图8-34　SU_3移位运算模块内部结构图

8.4流水线各段设计和功能描述

8.4.3 Stage 3执行有效地址计算段(EXE)

2. 模块划分和实现

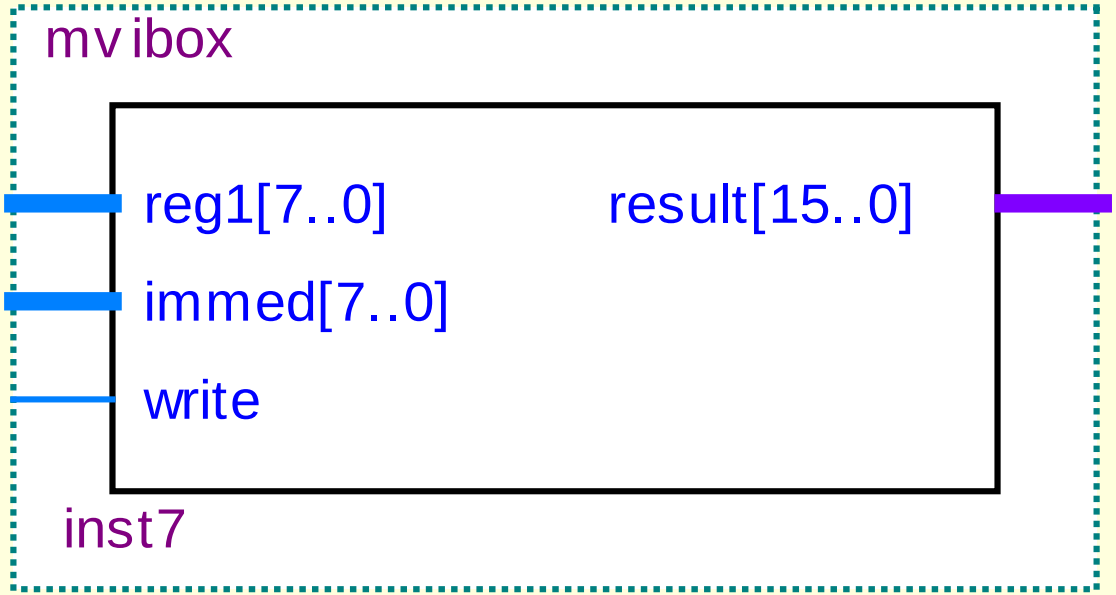


图 8-35 movibox数据输入模块结构

8.4流水线各段设计和功能描述

8.4.4 stage4访存段(MEM)

1. 功能描述

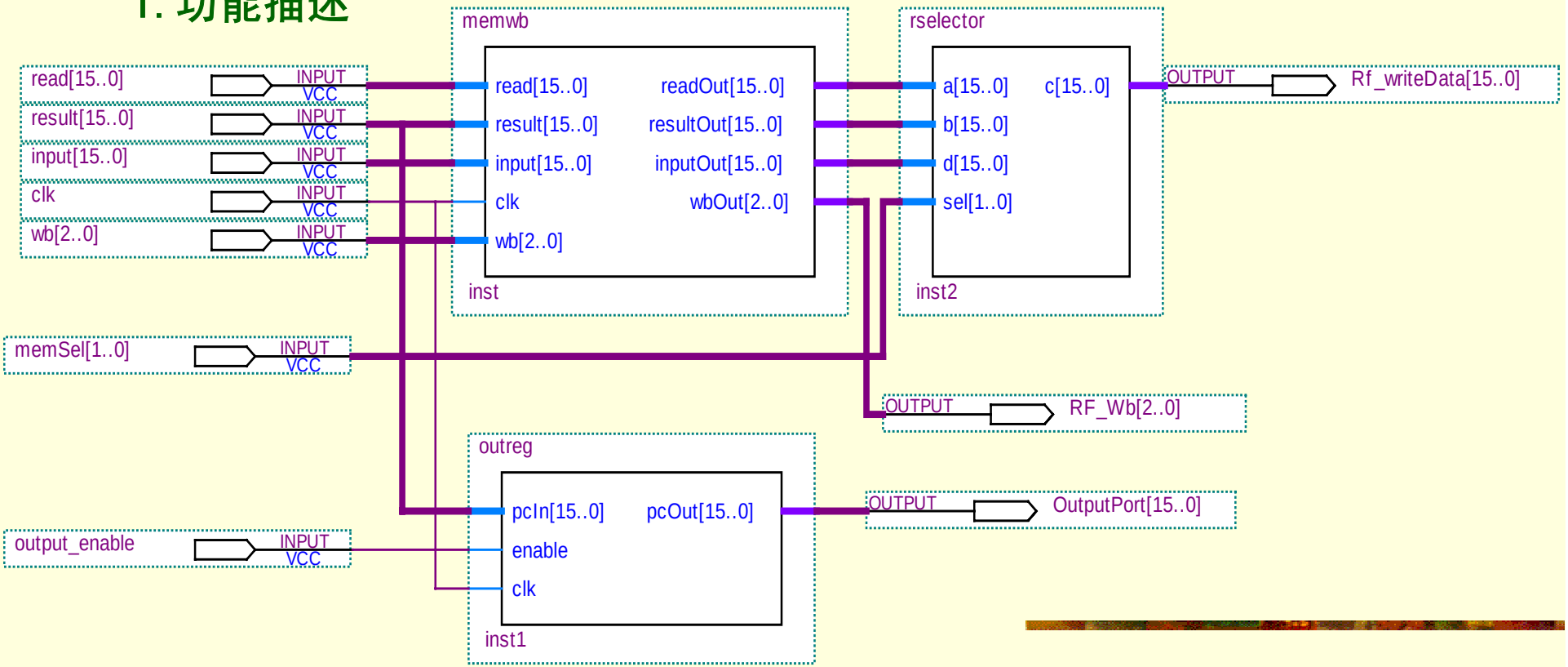


图8-36　访存段Stage 4的内部结构

8.4流水线各段设计和功能描述

8.4.4 stage4访存段(MEM)

2. 模块划分和实现

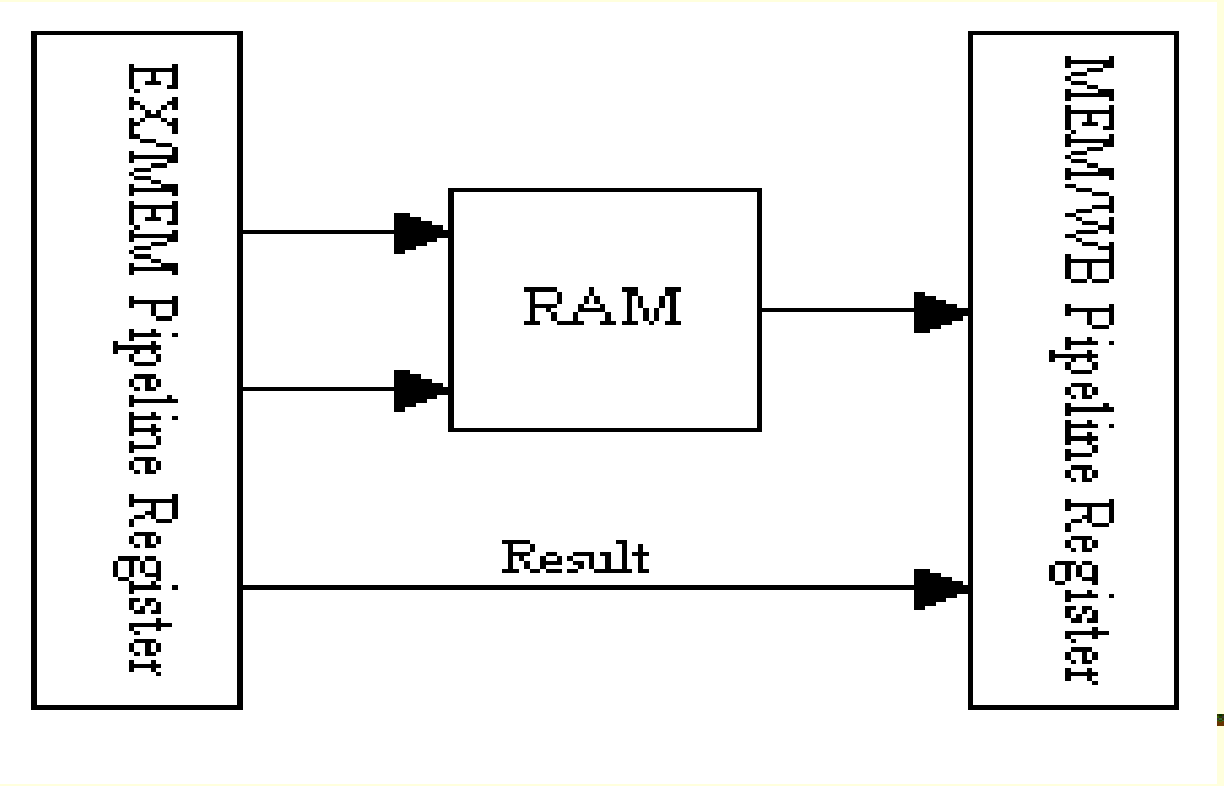


图8-37　Stage 4的基本结构

【例8-14】

```
library ieee;
use ieee.std_logic_1164.all;
ENTITY memwb IS
   PORT(read, result, input  : STD_LOGIC_VECTOR(15 DOWNTO 0);
        clk  : IN STD_LOGIC;
        wb   : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
        readOut, resultOut,inputOut : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
        wbOut  : OUT STD_LOGIC_VECTOR(2 DOWNTO 0));
END ENTITY memwb;
ARCHITECTURE memwb_behav OF memwb IS
BEGIN
   name : PROCESS(clk) IS
   VARIABLE regValue : STD_LOGIC_VECTOR(50 DOWNTO 0);
   BEGIN
      IF(clk='1') THEN
        regValue(50 DOWNTO 0) := result(15 DOWNTO 0) & read(15 DOWNTO 0) &
        input(15 DOWNTO 0) & wb(2 DOWNTO 0);  END IF;
     resultOut <= regValue(50 DOWNTO 35);  readOut <= regValue(34 DOWNTO 19);
     inputOut <= regValue(18 DOWNTO 3) ;    wbOut <= regValue(2 DOWNTO 0);
END ARCHITECTURE memwb_behav;
```

8.4流水线各段设计和功能描述

8.4.4 stage4访存段(MEM)

2. 模块划分和实现

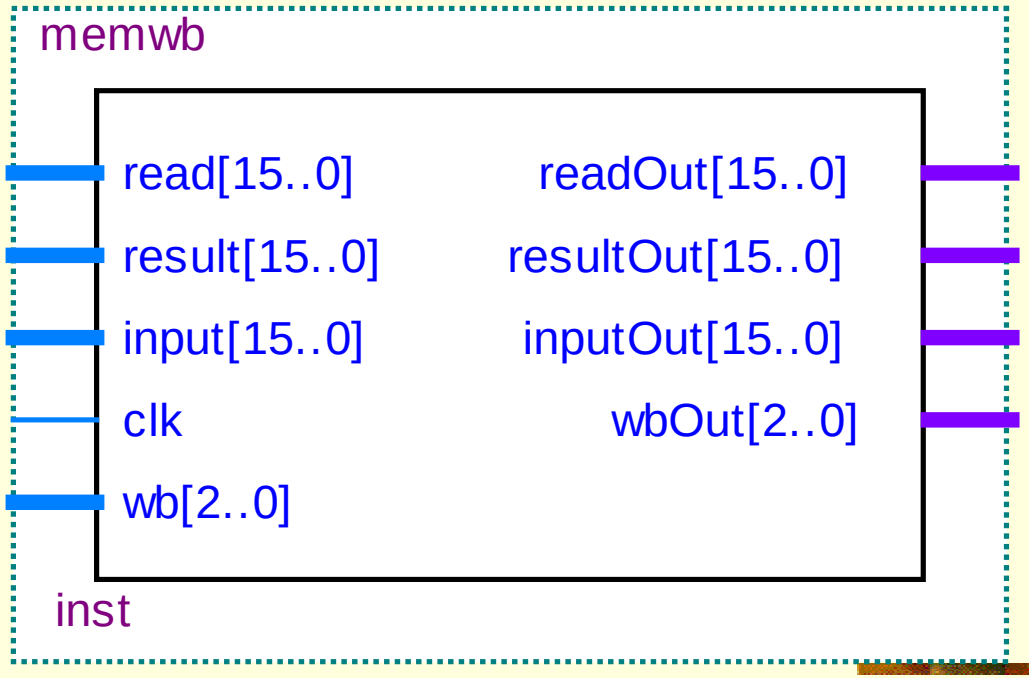


图8-38 Stage 4的MEM/WB流水线寄存器

例【 8-15】

```
library ieee;
use ieee.std_logic_1164.all;
ENTITY outreg IS
    PORT(pcIn : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
  enable,clk : IN STD_LOGIC;
      pcOut    : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
END ENTITY outreg;
ARCHITECTURE outreg_behav OF outreg IS
BEGIN
    name : PROCESS(clk) IS
    VARIABLE regValue : STD_LOGIC_VECTOR(15 DOWNTO 0);
     IF(clk='1' AND CLK'EVENT) THEN
        IF(enable='1') THEN   regValue := pcIn;
        END IF;
       pcOut <= regValue;
     END IF;
   END PROCESS name;
END ARCHITECTURE outreg_behav;
```

8.4流水线各段设计和功能描述

8.4.4 stage4访存段(MEM)

2. 模块划分和实现

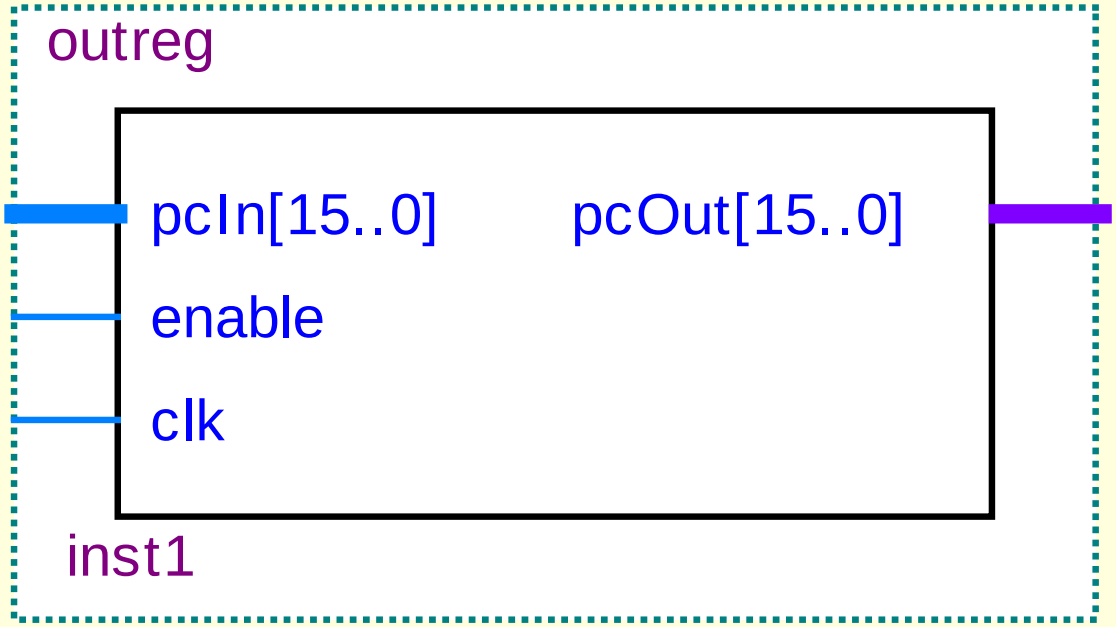


图8-39 Stage 4的输出寄存器

【例8-16】

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
ENTITY rselector IS
    PORT(a,b,d  : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
         sel    : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
         c      : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
END ENTITY rselector;
ARCHITECTURE rsel_behav OF rselector IS
BEGIN
    name : PROCESS(a,b,sel) IS
        CASE sel IS
            WHEN "00" =>    c<=a;
            WHEN "01" =>    c<=b;
            WHEN OTHERS =>  c<=d;
        END CASE;
    END PROCESS name;
END ARCHITECTURE rsel_behav;
```

8.4流水线各段设计和功能描述

8.4.4 stage4访存段(MEM)

2. 模块划分和实现

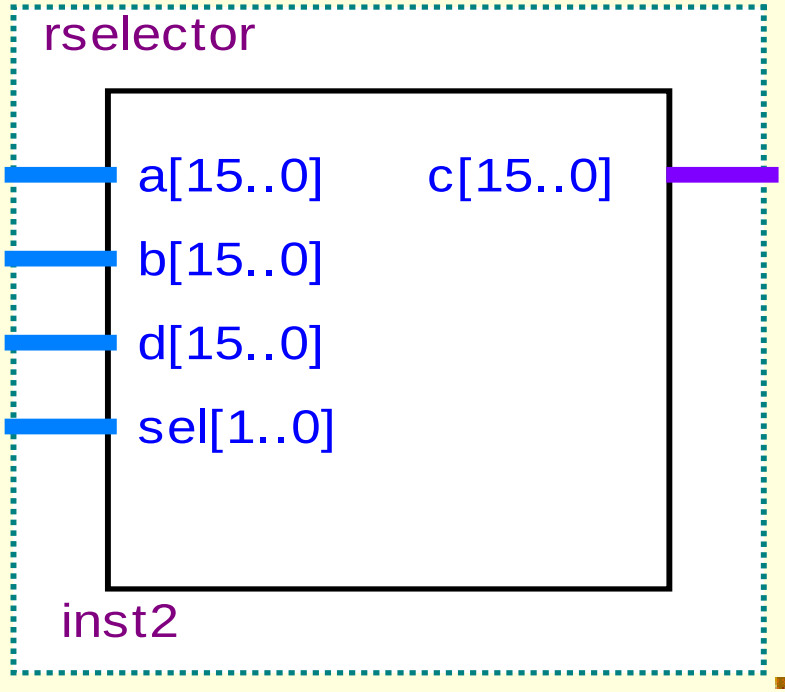


图8-40 Stage 4的3选1数据多路器

8.4流水线各段设计和功能描述

8.4.5 stage5回写段(WB)

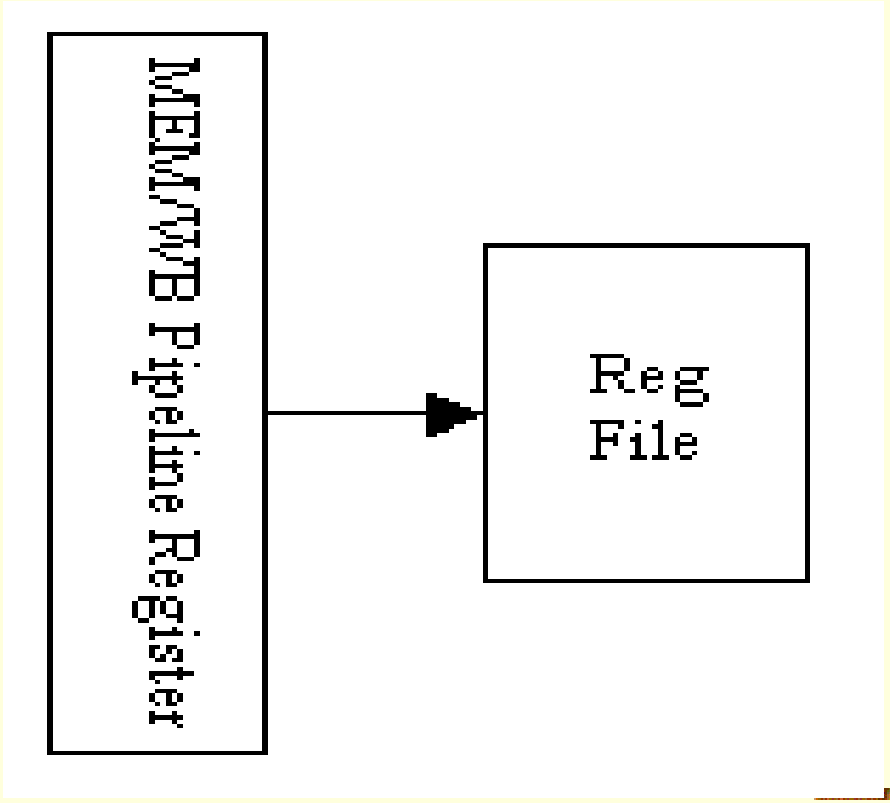


图8-41 Stage 5的基本结构

8.4流水线各段设计和功能描述

8.4.6 一些关键功能部件的设计

1. 数据相关的检测及处理

① 数据相关的检测

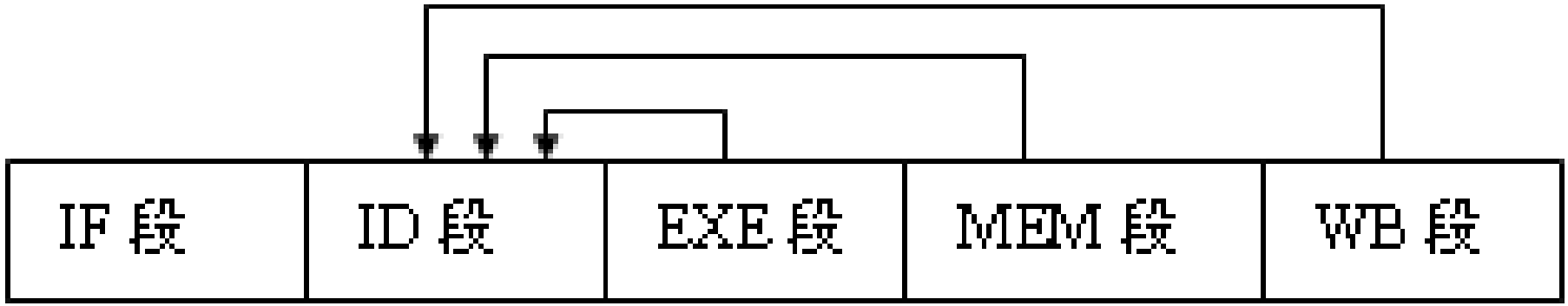


图8-42 数据相关性检测原理图

8.4流水线各段设计和功能描述

8.4.6 一些关键功能部件的设计

1. 数据相关的检测及处理

② 数据相关的发生

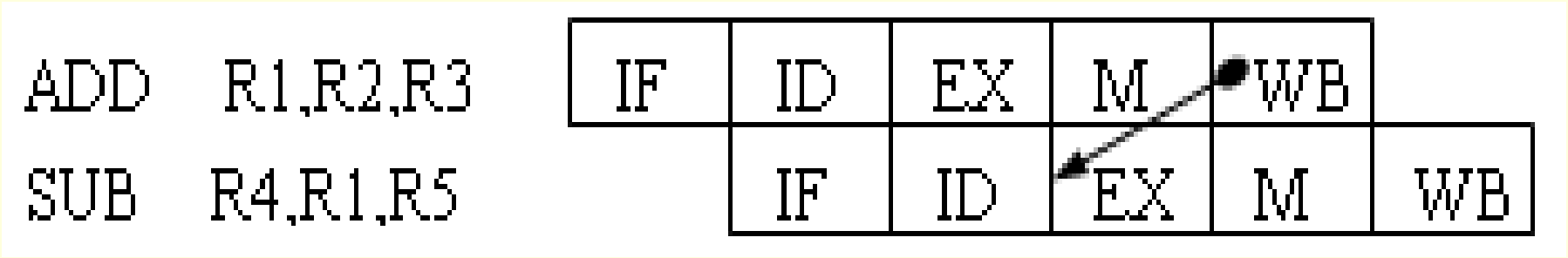


图8-43 数据相关实例

8.4流水线各段设计和功能描述

8.4.6 一些关键功能部件的设计

1. 数据相关的检测及处理

③ 数据相关的处理

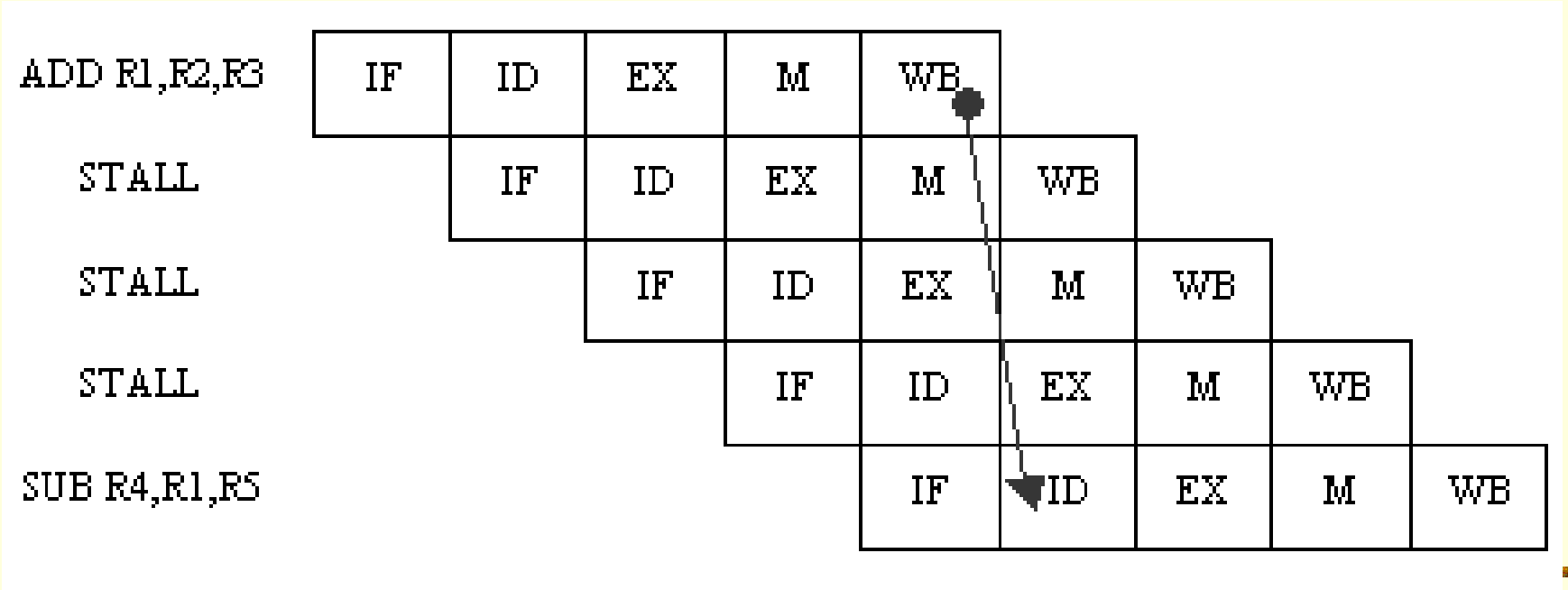


图8-44 阻塞流水线

8.4流水线各段设计和功能描述

8.4.6 一些关键功能部件的设计

1. 数据相关的检测及处理

③ 数据相关的处理

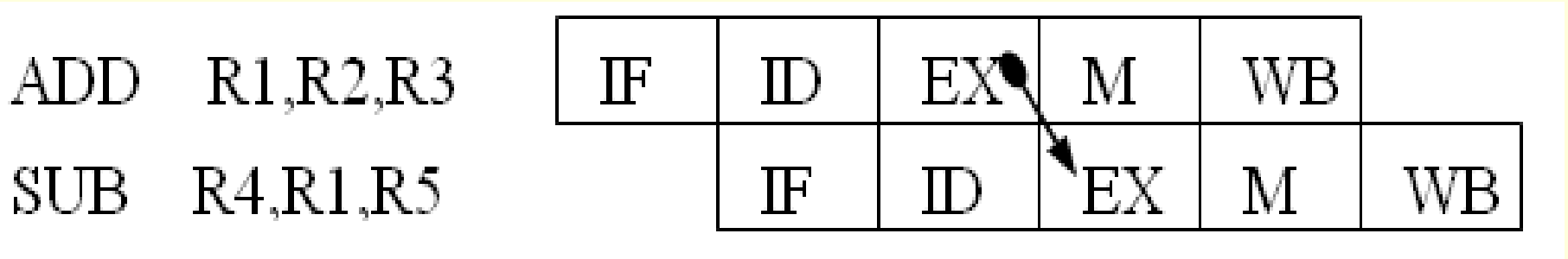


图8-45 数据前推控制

【例8-17】

```
library ieee;
use ieee.std_logic_1164.all;
entity hazard is
                    port(opcode :   in std_logic_vector(3 downto 0);
func, readone, readtwo, idexwrite :  in std_logic_vector(2 downto 0);
idexMemWE, idexwe :  in std_logic;
exmemwrite :        in std_logic_vector(2 downto 0);
memstage :          in std_logic;
pcenable, ifidenable,idexflush :     out std_logic);
end entity hazard;
architecture hazard_behav of hazard is
begin
name : process(opcode, func, readone, readtwo, idexwrite, idexMemWE, memstage,
exmemwrite) is
begin  --若写允许(we=1)有效，MEM/IO段的数据处理如下.
if(idexMemWE='1') then
if(idexwrite=readone or idexwrite=readtwo) then
pcenable <= '0'; ifidenable <= '0'; idexflush <= '1';
else  pcenable <= '1'; ifidenable <= '1'; idexflush <= '0';
end if;
--若写允许的同时遇到分支转移指令，并且要写入的寄存器就是转移指令要读出的寄存
器，
```

（接下页）

```
--则无论是MEM/IO段还是EX执行段都必须stall暂停执行
else if((opcode="0010" and func="010") or (opcode="0010" and
func="011") or opcode="0101") then --要写入的是EX执行段寄存器吗？
if(idexwe='1') then
if(idexwrite = readone or idexwrite=readtwo) then
pcenable <= '0'; ifidenable <= '0'; idexflush <= '1';
else pcenable <= '1'; ifidenable <= '1'; idexflush <= '0';
end if;
elsif (memstage='1') then  --要写入的是MEM存储段寄存器吗?
if(exmemwrite = readone or exmemwrite = readtwo) then
pcenable <= '0'; ifidenable <= '0'; idexflush <= '1';
else pcenable <= '1'; ifidenable <= '1'; idexflush <= '0';
end if;
--若两者都不是，则按正常操作
Else pcEnable <='1'; ifidenable <= '1'; idexflush <= '0'; end if;
else pcenable <= '1'; ifidenable <= '1'; idexflush <= '0'; end if;
end process name;
end architecture hazard_behav;
```

【例8-18】

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_unsigned.all;
ENTITY forward IS
   PORT(exmRegWrite,memwbRegWrite  : in std_logic;
        exmWriteReg, memwbWriteReg  : in std_logic_vector(2 downto 0);
        idexReadOne,idexReadTwo      : in std_logic_vector(2 downto 0);
        aluSelA, aluSelB             : out std_logic_vector(1 downto 0));
END ENTITY forward;
ARCHITECTURE forward_behav OF forward IS
BEGIN
   name : PROCESS(exmRegWrite, exmWriteReg, idexReadOne, idexReadTwo) IS
   BEGIN
      IF(exmRegWrite='1') THEN  --若来自EX/MEM段的寄存器要被写入
         IF(memwbRegWrite='0') THEN   --若写寄存器=读寄存器one,则写入
            IF(exmWriteReg=idexReadOne) THEN  aluSelA <= "01";
            ELSE   aluSelA <= "00";  END IF;
            IF(exmWriteReg=idexReadTwo) THEN  aluSelB <= "01";
            ELSE   aluSelB <= "00";  END IF;
```

（接下页）

```
ELSE
            IF(exmWriteReg=idexReadOne) THEN   aluSelA <= "01";
            ELSE IF(memwbWriteReg=idexReadOne) THEN aluSelA <= "10";
            ELSE  aluSelA <= "00"; END IF;
            END IF;
           IF(exmWriteReg=idexReadTwo) THEN  aluSelB <= "01";
--若EX段写寄存器=ID段读寄存器two,则写入
            ELSE IF(memwbWriteReg=idexReadTwo) THEN  aluSelB <= "10";
            ELSE  aluSelB <= "00";  END IF;
            END IF;
         END IF;
      ELSE IF(memwbRegWrite='1') THEN
            IF(memwbWriteReg=idexReadTwo) THEN  aluSelB <= "10";
            ELSE  aluSelB <= "00";  END IF;
            IF(memwbWriteReg=idexReadOne) THEN  aluSelA <= "10";
            ELSE aluSelA <= "00";  END IF;
         ELSE   aluSelA <= "00";  aluSelB <= "00"; END IF;
      END IF;
END PROCESS name;
END ARCHITECTURE forward_behav;
```

8.4流水线各段设计和功能描述

8.4.6 一些关键功能部件的设计

1. 数据相关的检测及处理

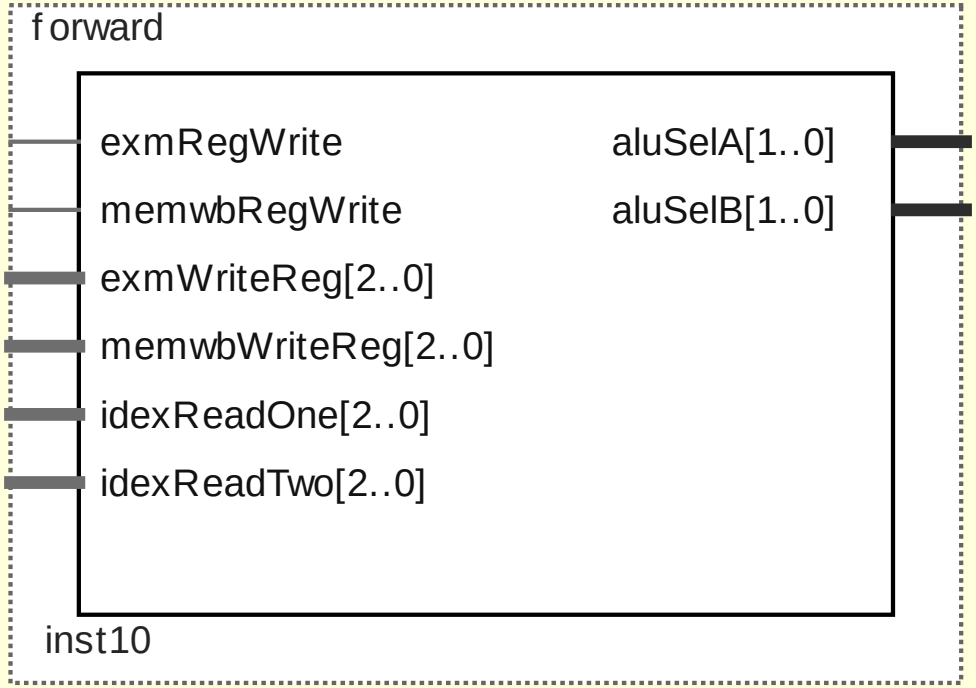


图8-47 forward模块电路结构

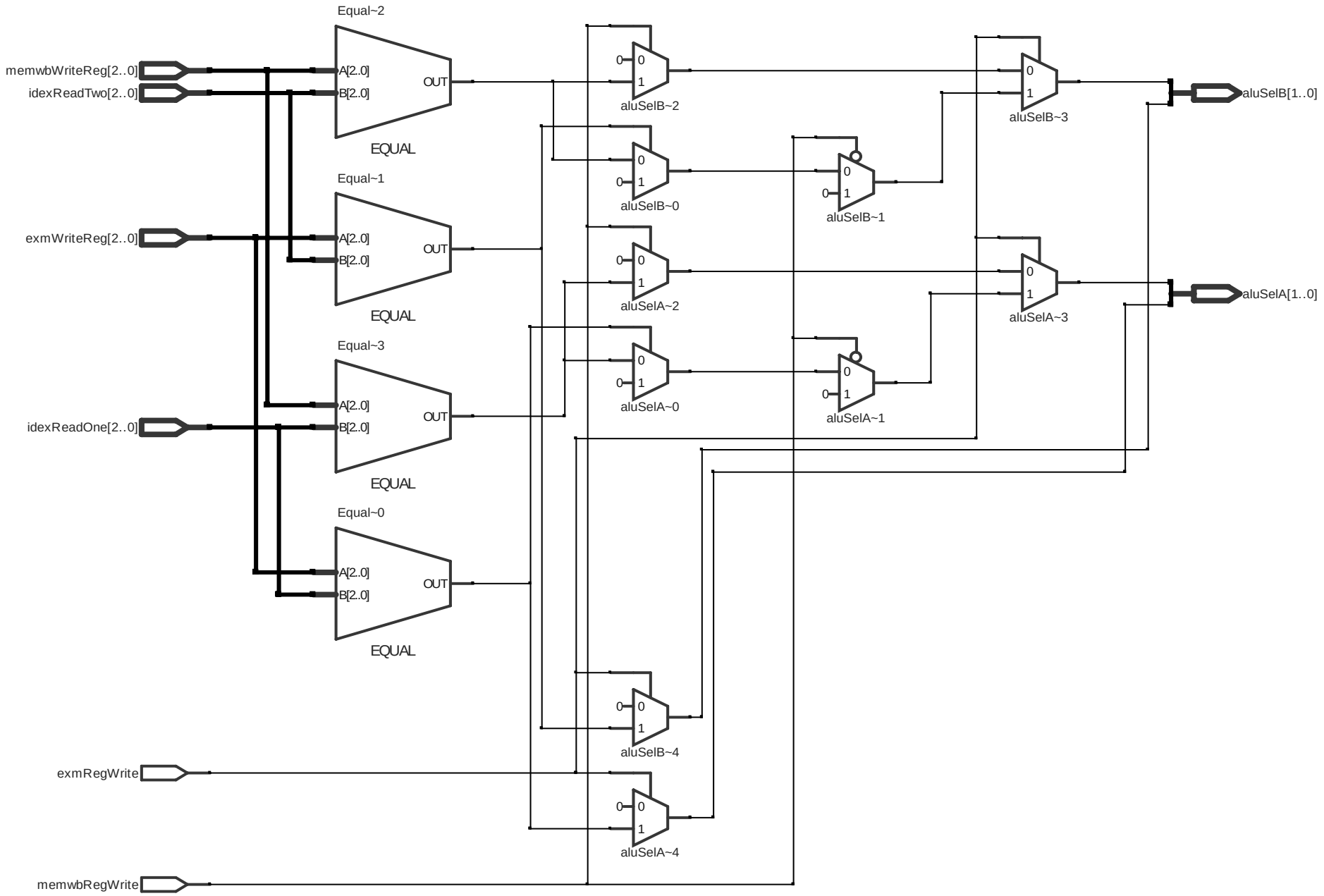


图8-48 forward模块RTL电路结构

8.4流水线各段设计和功能描述

8.4.6 一些关键功能部件的设计

1. 数据相关的检测及处理

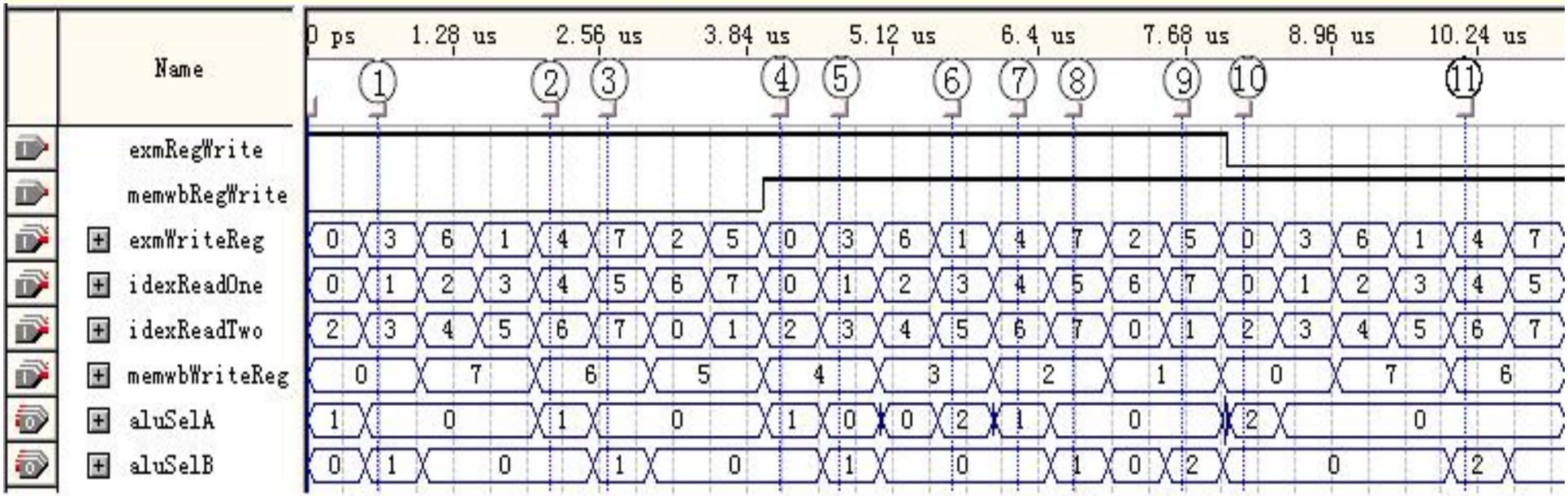


图8-49 forward模块的时序仿真图

8.4流水线各段设计和功能描述

8.4.6 一些关键功能部件的设计

2. 控制相关

3. 结构相关

（1）结构相关检测

（2）阻塞

（3）预取

（4）资源重复

8.4流水线各段设计和功能描述

8.4.7 控制单元

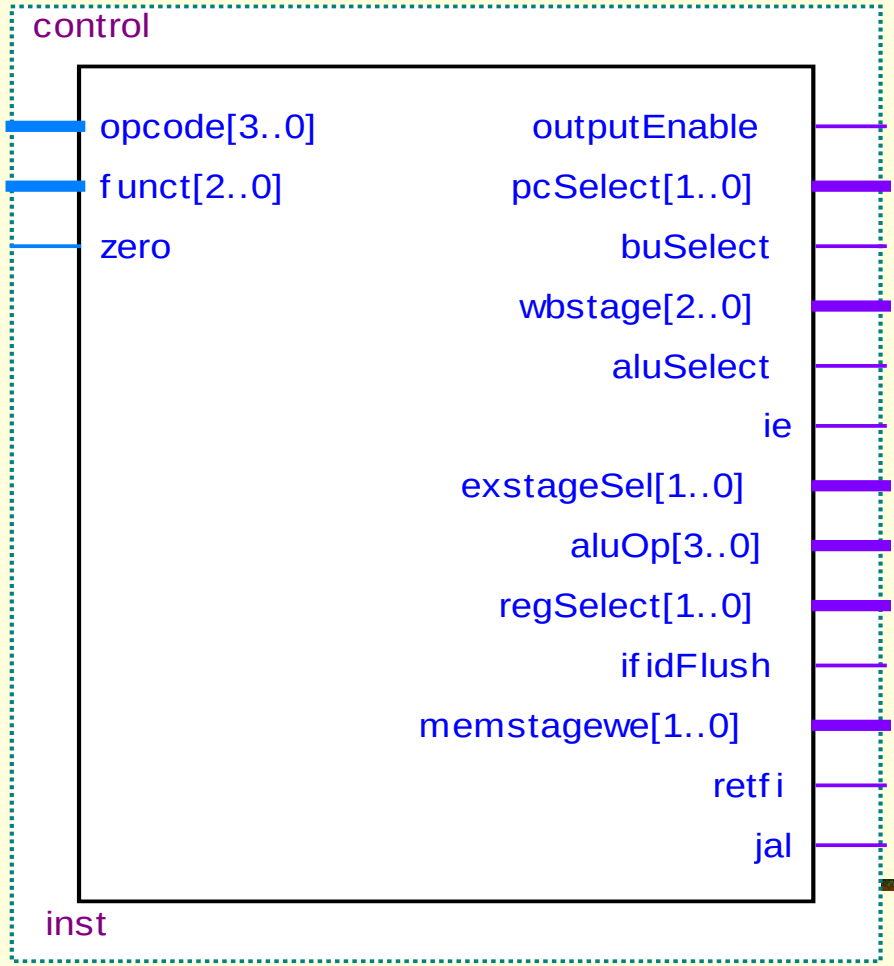


图8-50 控制单元control的结构图

8.4流水线各段设计和功能描述

8.4.8 中断与异常

1. 中断

中断是指外部事件向处理器发出的服务请求。

2. 异常

本处理器能够处理的两种异常是算术溢出和未定义指令。
算术溢出或未定义指令将导致异常信号线变为高电平，
这一信号将导致当前指令及其后续指令都从流水线排出。

3. 优先级

在本设计中，优先级的数值范围从**0**到**5**，其中，中断**0**具有最高优先级。优先级的顺序实际上是通过**IF**、**ELSIF**和**ELSE**语句实现的。

8.4流水线各段设计和功能描述

8.4.9 流水线CPU系统电路结构

这是一个具有五级流水线的RISC结构的CPU，基本操作有取指IF、译码ID、执行EXE、访存MEM、和回写WB。主要的组成部件有：取指、指令译码、执行、访存/回写、指令预取、前推控制、中断控制、分支转移、相关检测及核心控制器等功能模块。此外还有程序存储器和数据存储器。

8.4流水线各段设计和功能描述

8.4.10 CPU与LCD显示模块的接口

现代计算机体系结构实验

IF:PC	00	Instr	0000		
ID:PC	00	REG1	00	DT1	0000
		REG2	00	DT2	1100
		D6#	00	D16#	000C
EX:PC	86	FUNC	00	SEL	00
ALU	0C00	REGA	FFFF	REGB	0000
MEM:RD	0000	IN	0000	RSLT	0000
WB:REG	11	DATA	0000		
I/O:IN	000C	OUT	0600		
INTR	00	intPC	0C00		

图8-51 液晶LCD240×128显示流水线信息

8.4流水线各段设计和功能描述

8.4.10 CPU与LCD显示模块的接口

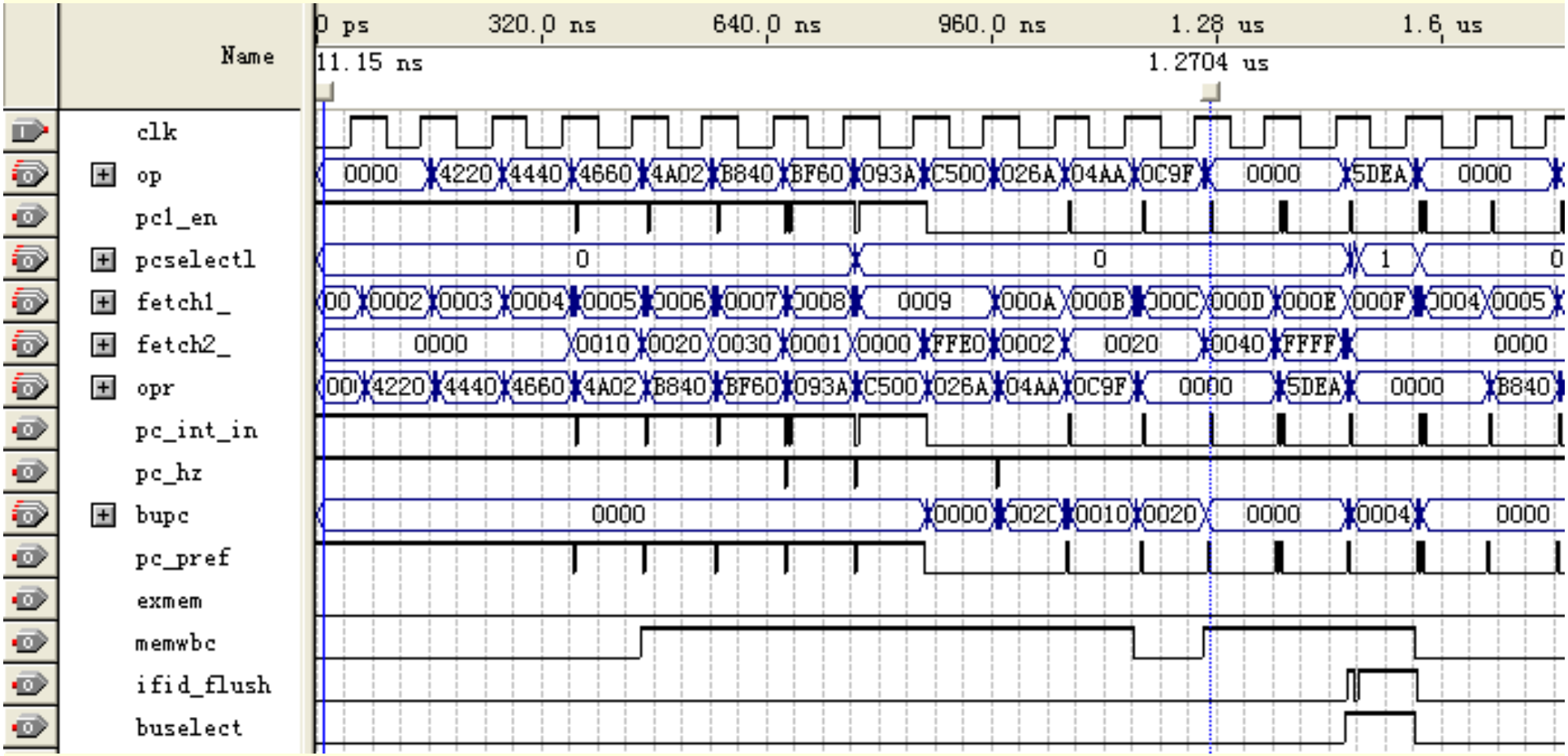


图8-52 流水线CPU执行程序仿真波形

【例8-19】应用程序设计

地址	机器码	指令	说　　明
00	4220	MVIL　R1,10H	R1 是源指针，立即数 10H 送 R1，
01	4440	MVIL　R2,20H	R2 是目的指针，立即数 20H 送 R2
02	4660	MVIL　R3,30H	R3 是结束地址，立即数 30H 送 R3
03	4A02	MVIL　R5,01`	立即数 01H 送 R5
04	B840	LOOP:LW　R4,R1,0	将 R1 的内容作为地址，从存储器中取数送 R4，
05	BF60	LW　R7,R1,10H	将 R1 的内容加 10H 作为地址，从存储器中取数送 R7
06	093A	ADD　R4,R4,R7	R4←（R4）+（R7）
07	C500	SW　R2,R4,0	将 R4 的内容存到以 R2 的内容为地址的 RAM 存储单元
08	026A	ADD　R1,R1,R5	修改源指针
09	04AA	ADD　R2,R2,R5	修改目的指针
0A	0C9F	SLT　R6,R2,R3	比较结束地址 R6←（R2）-（R3）
0B	5DEA	BZI　R6,LOOP	判断、程序转移
0C	0000	NOP	结束
0D-3F	0000		

习 题

8-1．基于本章介绍的流水线CPU，编写程序，在七段数码管上循环显示数字0~9。

8-2．基于本章介绍的流水线CPU，编写程序，先把字符‘A’~‘F’写到内存的040H~045H几个单元中，然后再读出来并送到输出端口。

8-3．基于本章介绍的流水线CPU，编写程序，完成从输入端口输入字符并送到输出端口。使用子程序调用指令和转移指令；在子程序中，试完成将大写英文字母变为小写字母，并送到输出端口的功能。

8-4．基于本章介绍的流水线CPU，设计一个程序，在数码管LED上依次显示输出‘0’~‘9’这10个数字。

8-5．基于本章介绍的流水线CPU，计算1到N的累加和，将结果回写存储器并送到输出端口显示。

8-6．基于本章介绍的流水线CPU，设计一个有读写内存和子程序调用指令的程序，功能是读出指定内存中的大写字母字符，并将其送到输出端口；转换为小写字母后再回写存储器原存储区域。

8-7．如果要为CPU设计一新的应用程序，如何将汇编程序代码转换为机器码，并将机器码配置到lpm_rom中？

实验与设计

实验8-1．Stage1取指令段实验

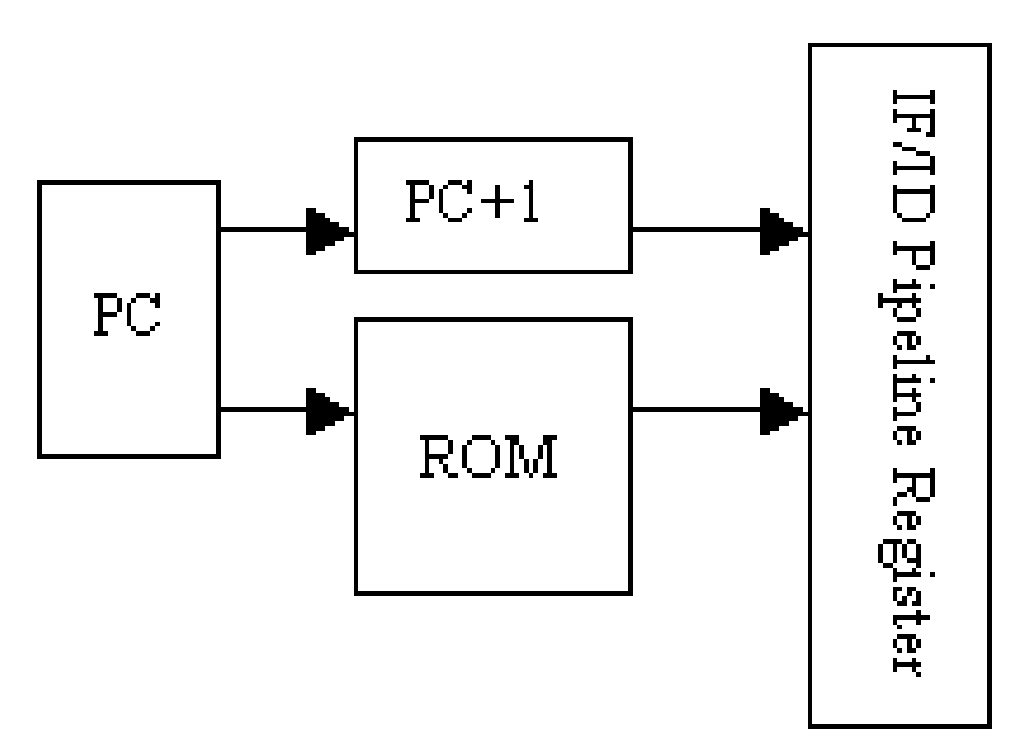


图8-53 Stage 1 取指段IF的结构

实验与设计

实验8-1．Stage1取指令段实验

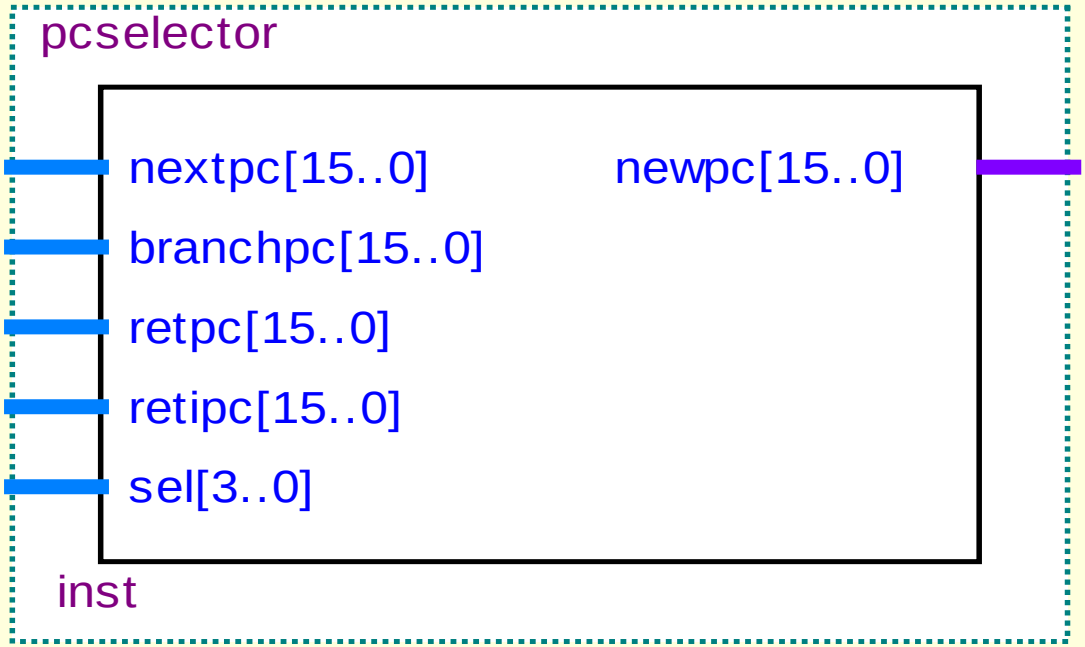


图8-54 选择模块selector的结构

实验与设计

实验8-1．Stage1取指令段实验

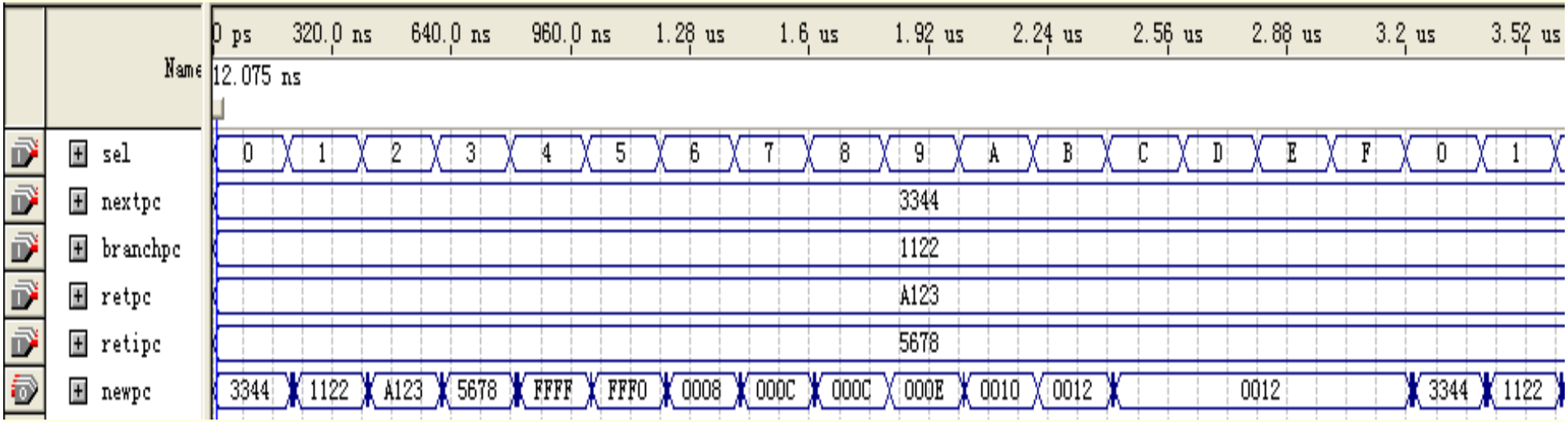


图8-55 选择模块selector的仿真波形

实验与设计

实验8-1．Stage1取指令段实验

Addr	+0	+1	+2	+3	+4	+5	+6	+7
00	4220	4440	4660	4A02	B840	BF60	093A	C880
08	026A	04AA	0C9F	0000	0000	5DEA	0000	0000
10	0000	0000	0000	0000	0000	0000	0000	0000
18	0000	0000	0000	0000	0000	0000	0000	0000
20	0000	0000	0000	0000	0000	0000	0000	0000
28	0000	0000	0000	0000	0000	0000	0000	0000
30	0000	0000	0000	0000	0000	0000	0000	0000
38	0000	0000	0000	0000	0000	0000	0000	0000

图8-56 lpm_rom文件中的数据

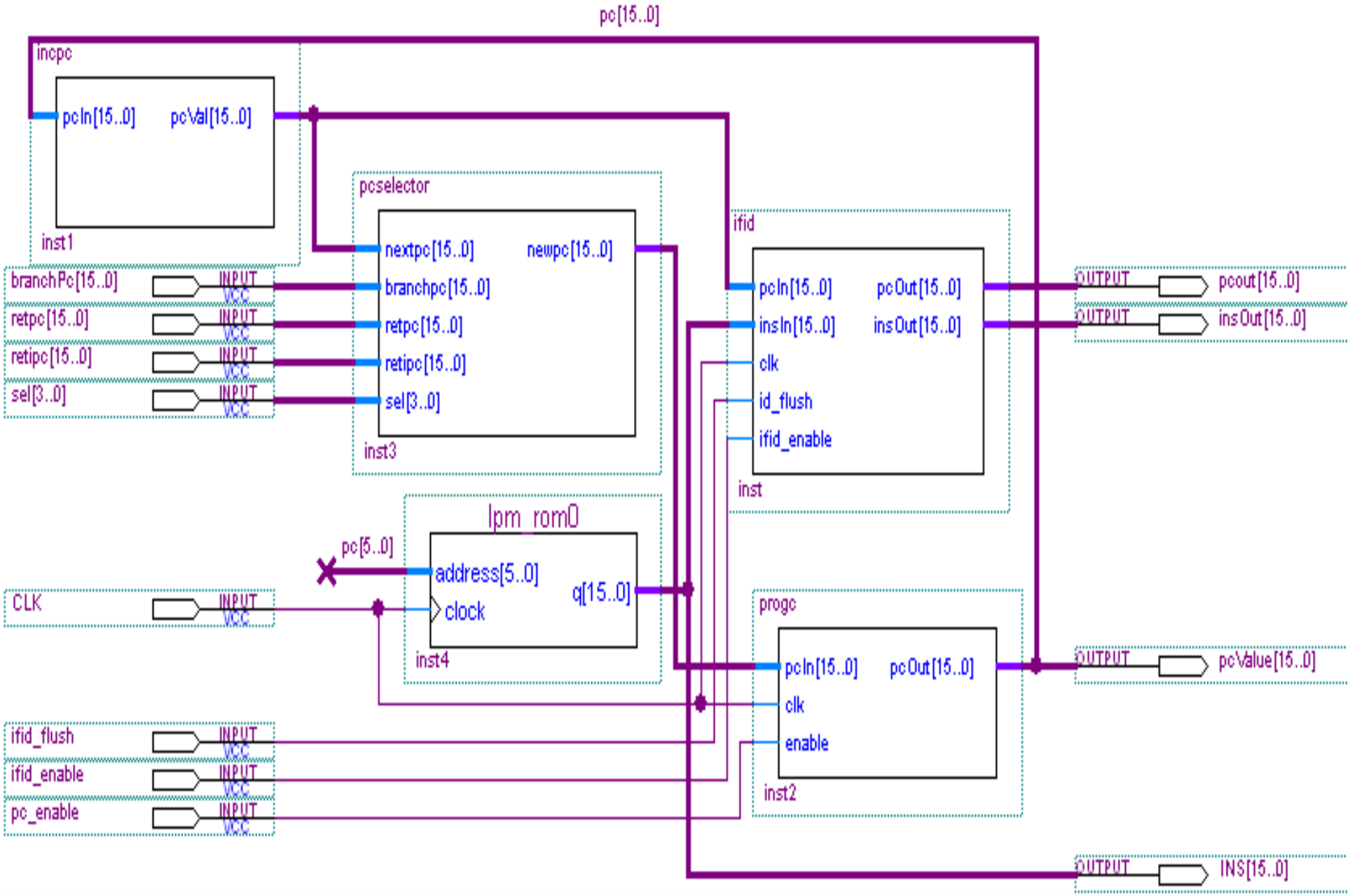


图8-57 Stage1 综合实验电路

实验与设计

实验8-1．Stage1取指令段实验

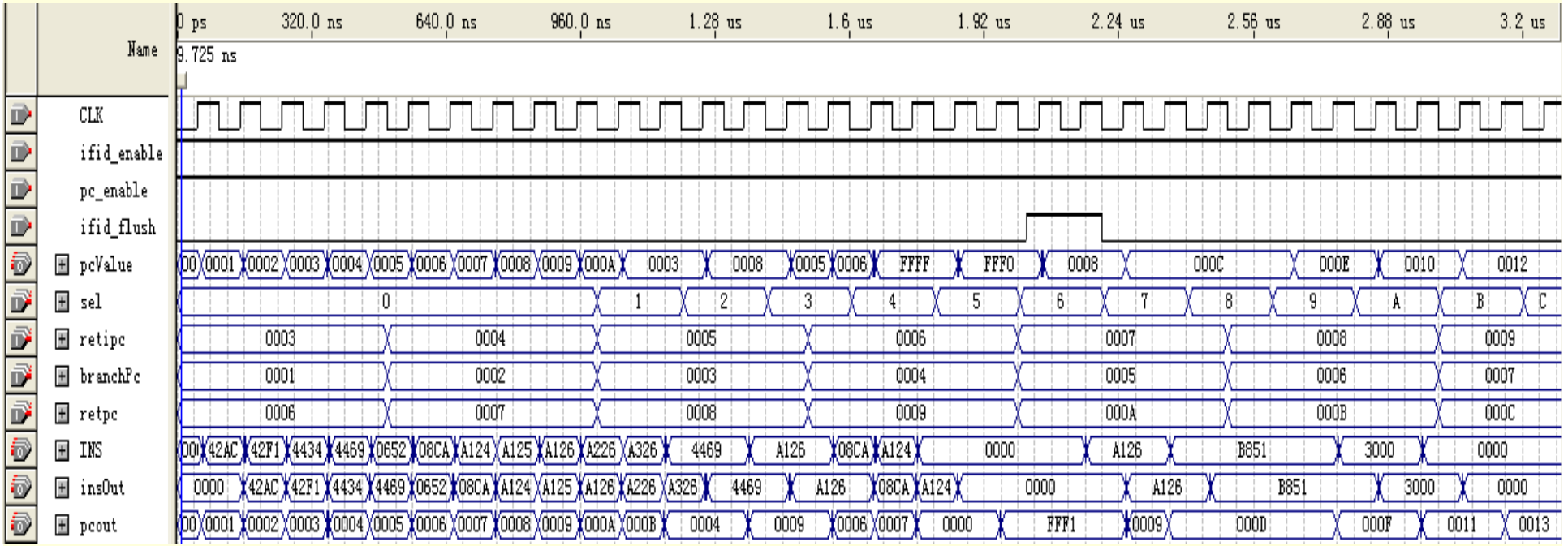


图8-58 Stage1 仿真波形

实验与设计

实验8-2. Stage2指令译码段实验

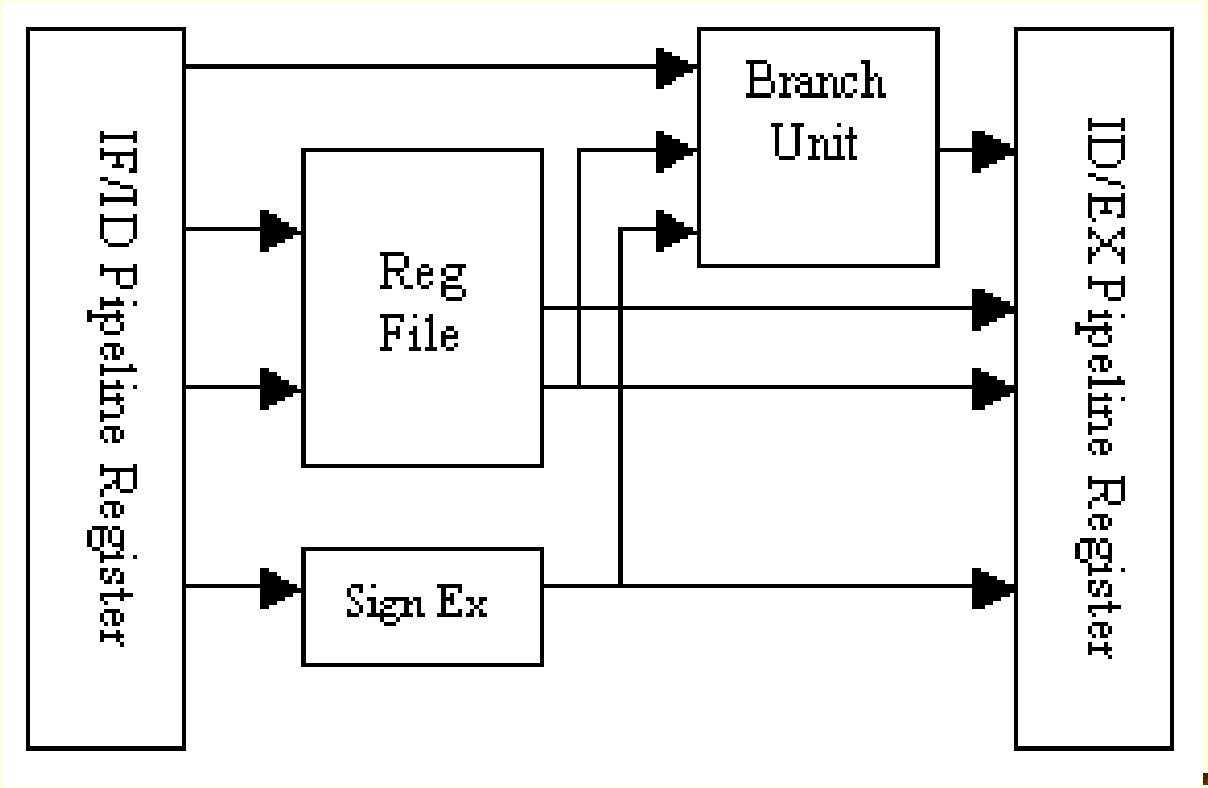


图8-59 Stage 2译码段结构

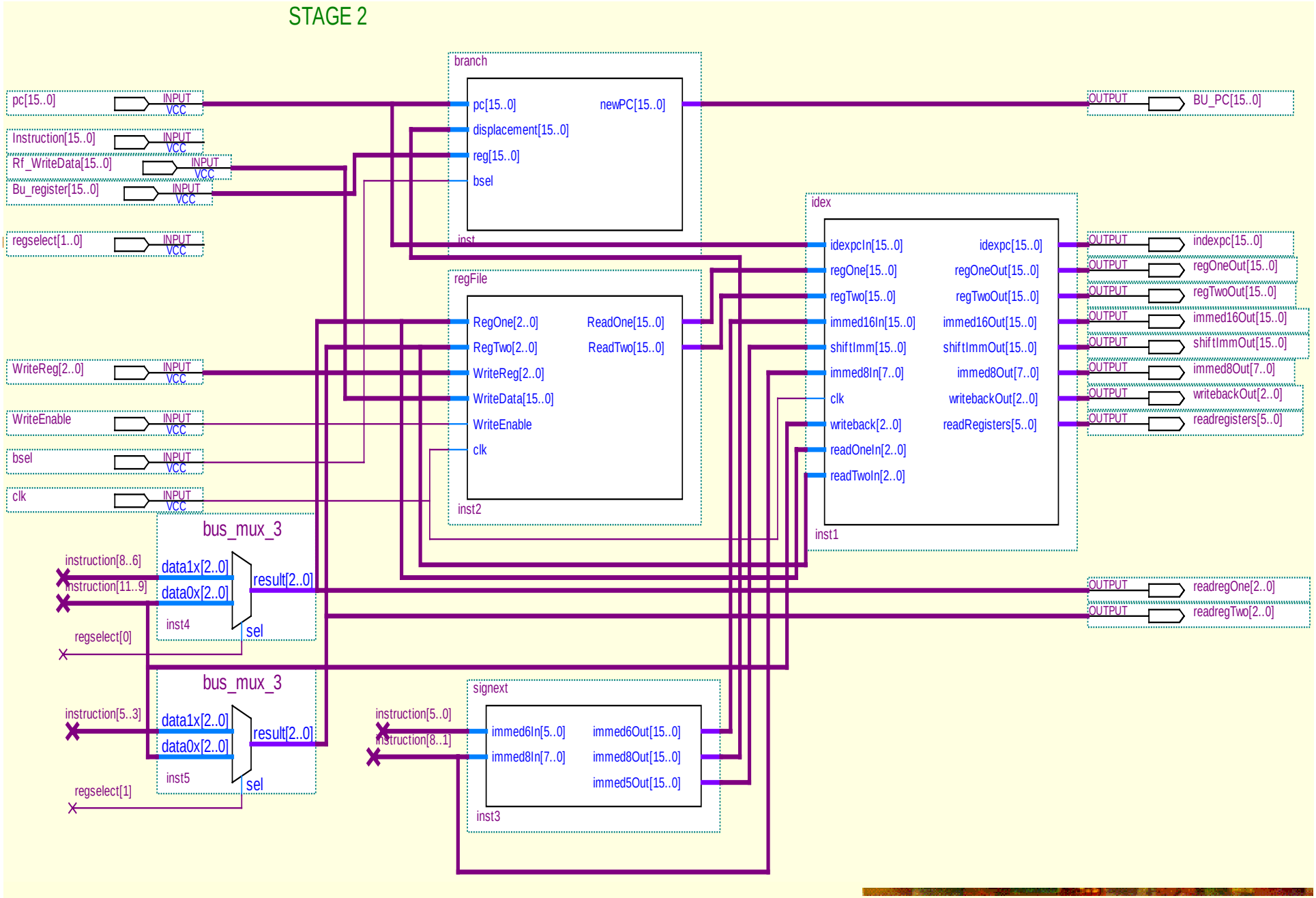


图8-60 Stage 2 译码段ID的各模块信号连接电路

实验与设计

实验8-2. Stage2指令译码段实验

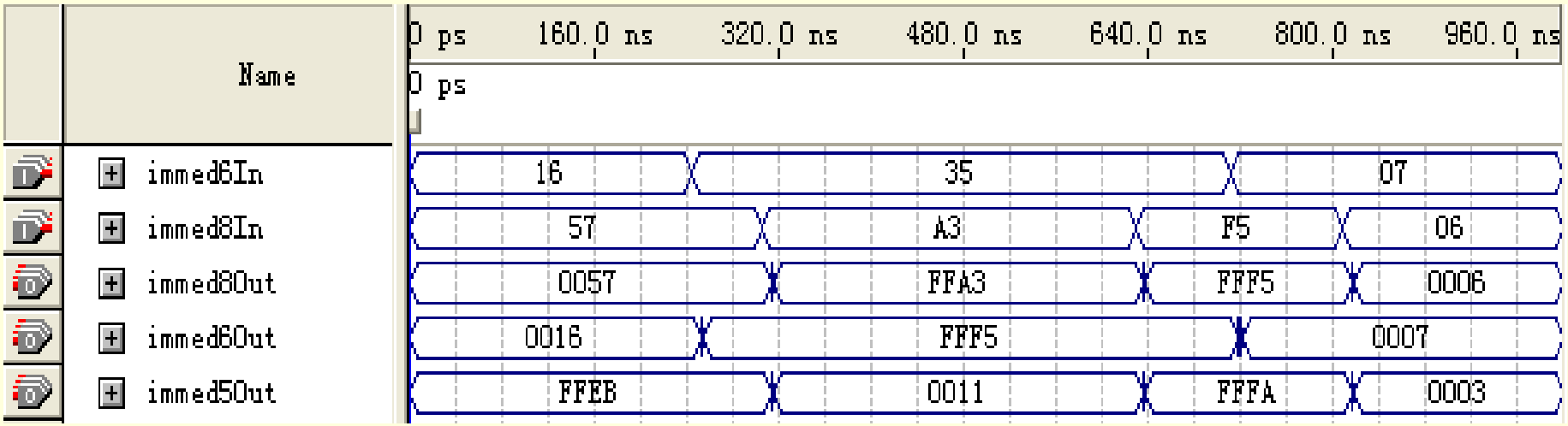


图8-61 signext符号扩展模块仿真波形

实验与设计

实验8-2. Stage2指令译码段实验

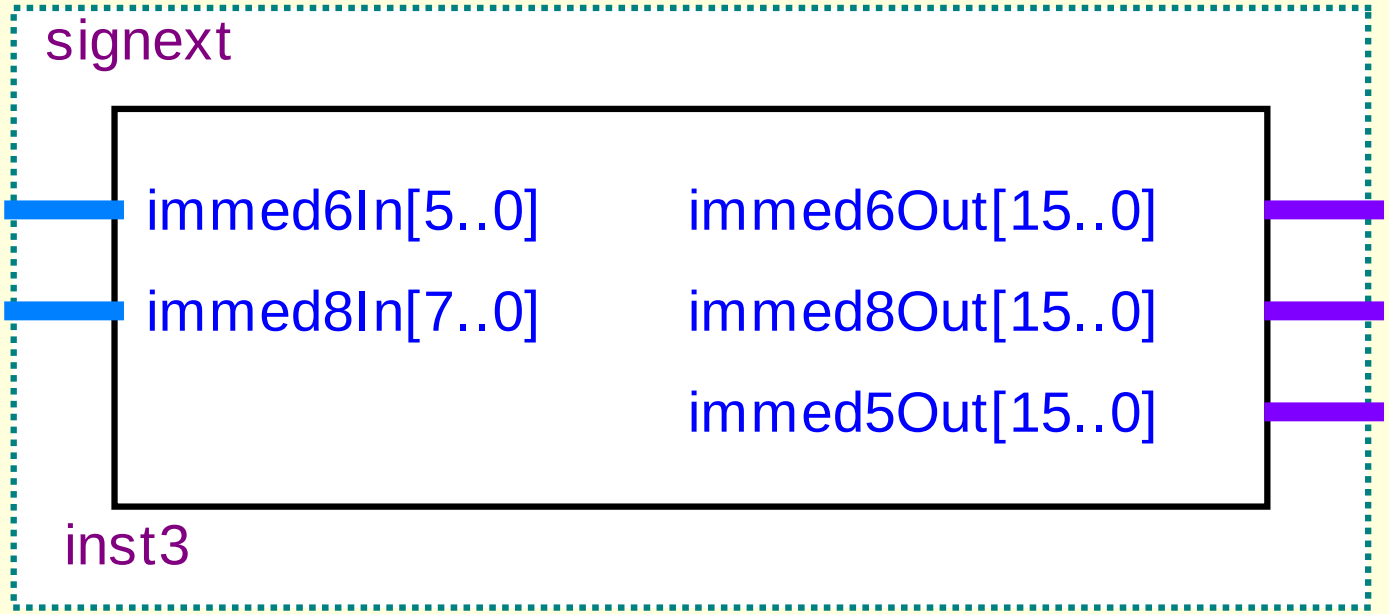


图8-62 符号扩展模块结构

实验与设计

实验8-2. Stage2指令译码段实验

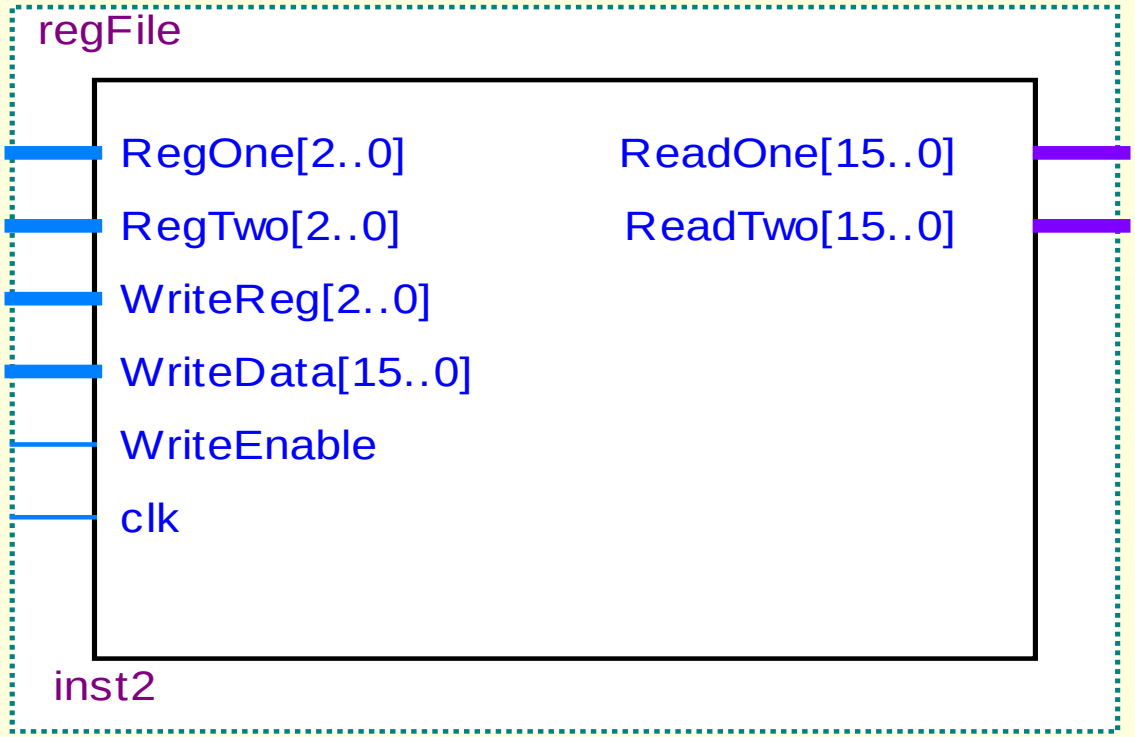


图8-63 寄存器文件模块结构

实验与设计

实验8-2. Stage2指令译码段实验

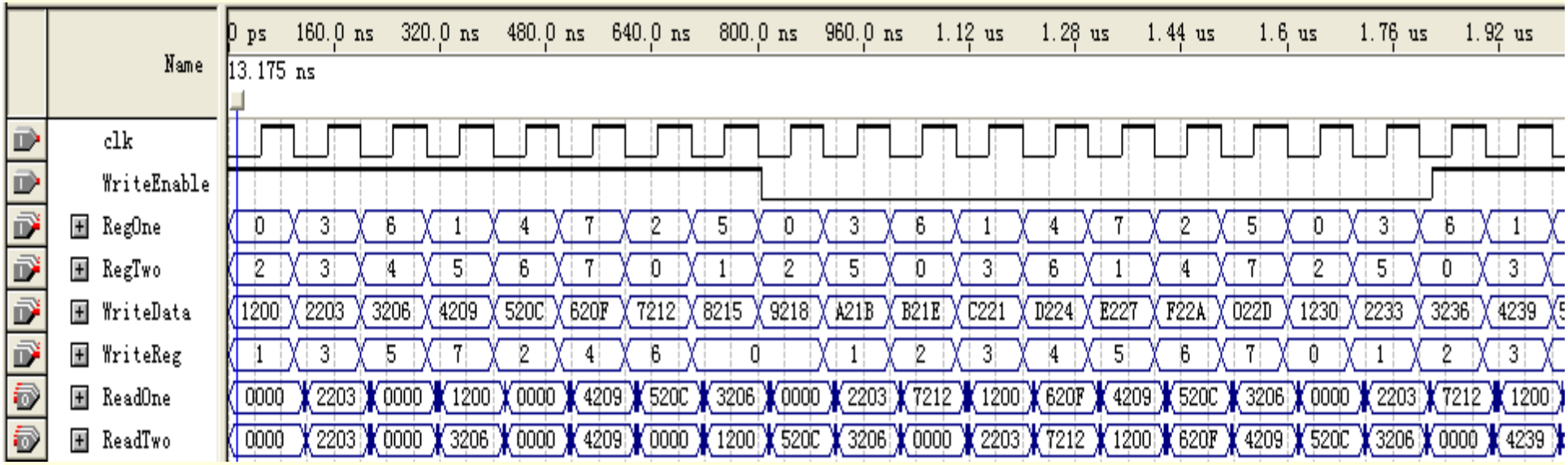


图8-64 regFile的时序仿真波形

实验与设计

实验8-2. Stage2指令译码段实验

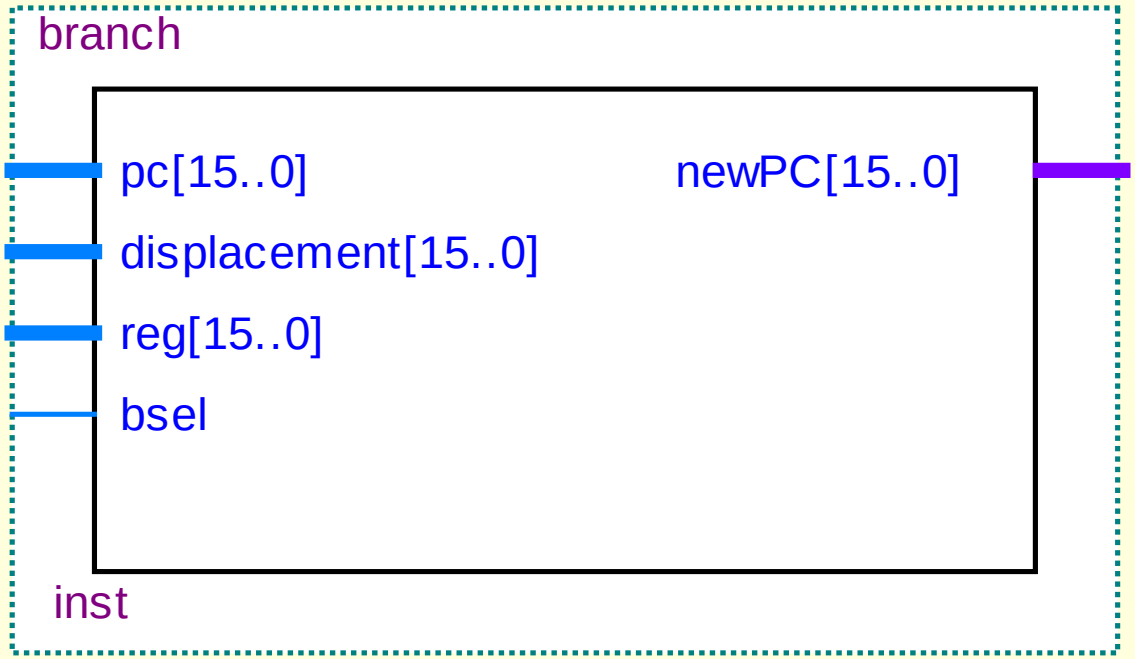


图8-65 分支控制模块结构

实验与设计

实验8-2. Stage2指令译码段实验

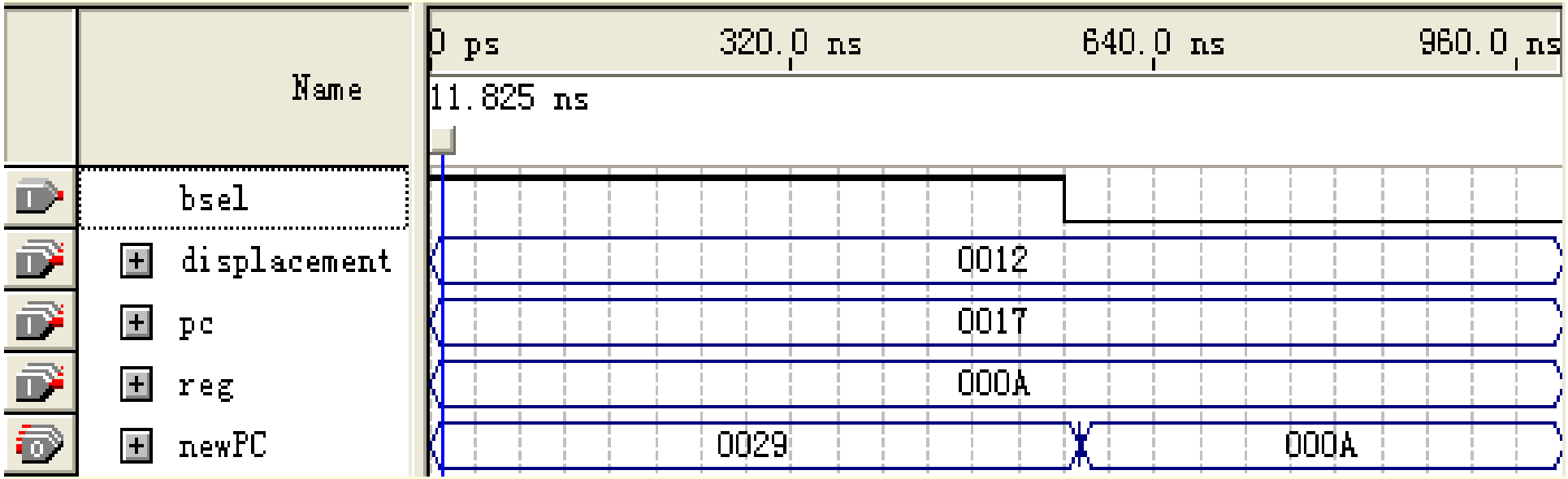


图8-66 branch.vhd的仿真波形

实验与设计

实验8-2. Stage2指令译码段实验

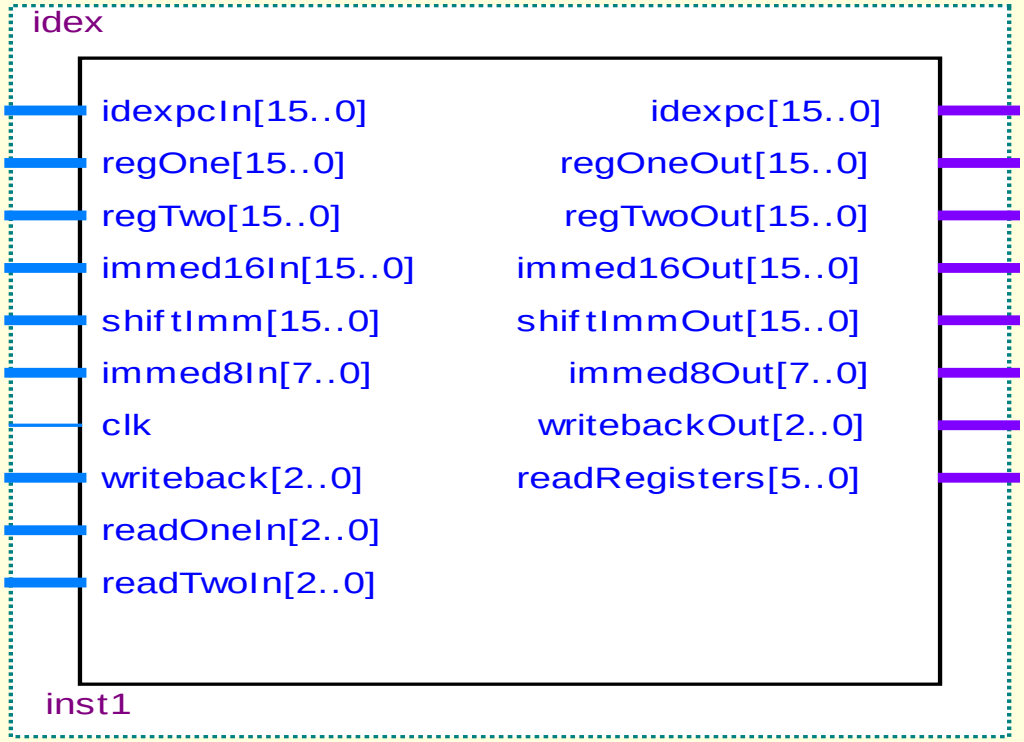


图8-67 ID/EX段流水线寄存器的结构

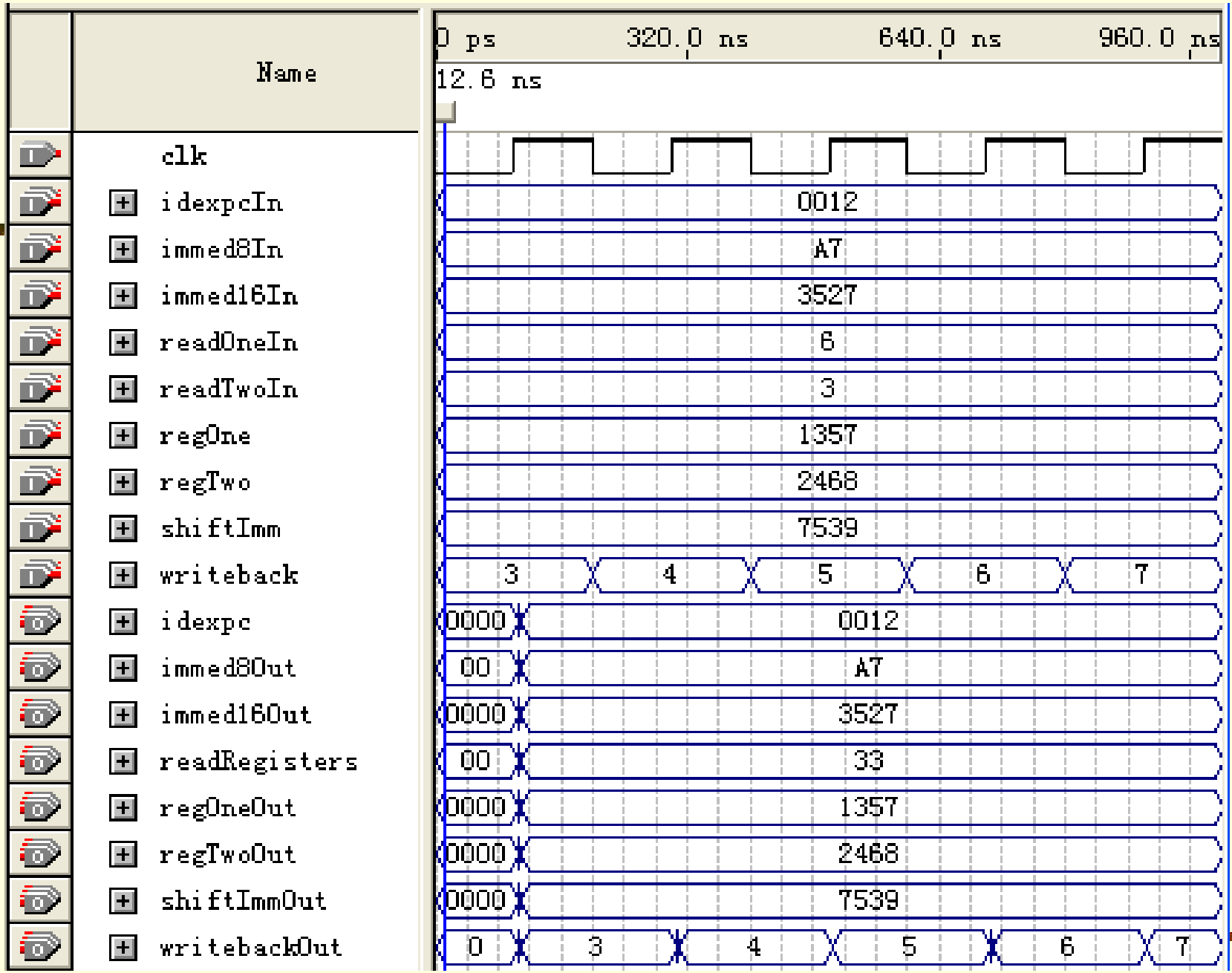


图8-67 ID/EX段流水线寄存器的结构

实验与设计

实验8-2. Stage2指令译码段实验

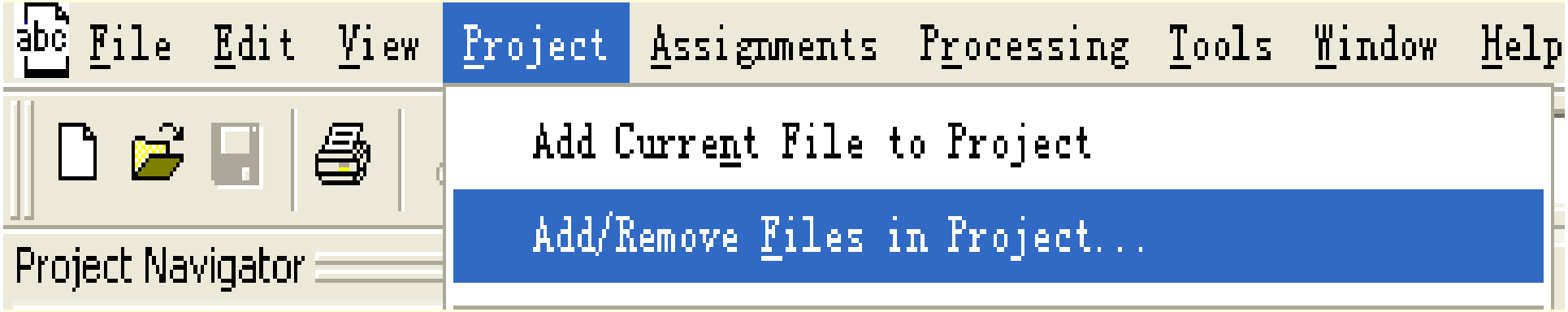


图8-69 在Project中添加/删除程序

bus_mux_3.vhd的程序如下：

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY lpm;
USE lpm.lpm_components.all;
ENTITY bus_mux_3 IS
        PORT( data0x，data1x      : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
                sel  :  IN  STD_LOGIC  ;  result  :  OUT  STD_LOGIC_VECTOR  (2
DOWNTO 0) );
END bus_mux_3;
ARCHITECTURE SYN OF bus_mux_3 IS
        SIGNAL sub_wire0          : STD_LOGIC_VECTOR (2 DOWNTO 0);
        SIGNAL sub_wire1          : STD_LOGIC ;
        SIGNAL sub_wire2          : STD_LOGIC_VECTOR (0 DOWNTO 0);
        SIGNAL sub_wire3          : STD_LOGIC_VECTOR (2 DOWNTO 0);
        SIGNAL sub_wire4          : STD_LOGIC_2D (1 DOWNTO 0, 2 DOWNTO 0);
        SIGNAL sub_wire5          : STD_LOGIC_VECTOR (2 DOWNTO 0);
        COMPONENT lpm_mux
        GENERIC (lpm_size : NATURAL;
                lpm_type                  : STRING;
                lpm_width                 : NATURAL;
                lpm_widths                : NATURAL       );
        PORT (          sel       : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
```

（接下页）

实验与设计

```
                data        : IN STD_LOGIC_2D (1 DOWNTO 0, 2 DOWNTO 0);
                result      : OUT STD_LOGIC_VECTOR (2 DOWNTO 0)           );
        END COMPONENT;
BEGIN
        sub_wire5 <= data0x(2 DOWNTO 0);      result <= sub_wire0(2 DOWNTO 0);
        sub_wire1 <= sel;                sub_wire2(0) <= sub_wire1;
        sub_wire3 <= data1x(2 DOWNTO 0);              sub_wire4(1, 0) <= sub_wire3(0);
        sub_wire4(1, 1) <= sub_wire3(1);      sub_wire4(1, 2) <= sub_wire3(2);
        sub_wire4(0, 0) <= sub_wire5(0);      sub_wire4(0, 1) <= sub_wire5(1);
        sub_wire4(0, 2) <= sub_wire5(2);
lpm_mux_component : lpm_mux
        GENERIC MAP ( lpm_size => 2, lpm_type => "LPM_MUX", lpm_width => 3,
lpm_widths => 1 )
        PORT MAP (      sel => sub_wire2, data => sub_wire4,result=>sub_wire0 );
END SYN;
```

实验与设计

实验8-2. Stage2指令译码段实验

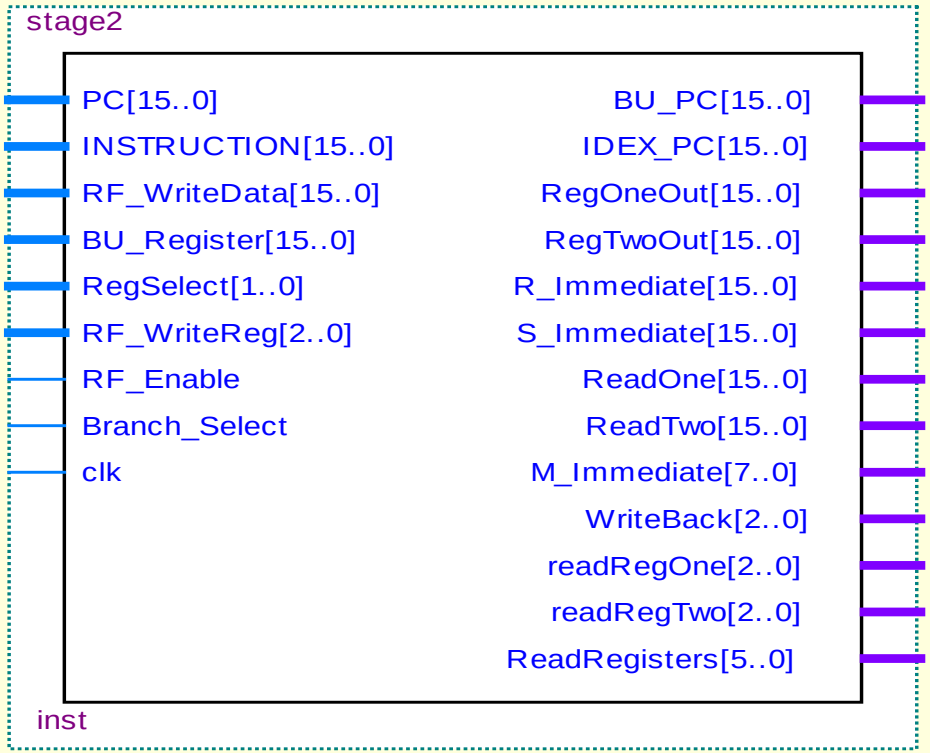


图8-70 编译后stage2的模块结构

图8-70 编译后stage2的模块结构

Func	功　能	说　明
0	c=a∧b	逻辑与
1	c=a∨b	逻辑或
2	c=a⊕b	逻辑异或
3	c=not(a∨b)	逻辑或非
4		
5	c=a+b	有符号数加法
6	c=a-b	有符号数减法
7	c=a+b	无符号数加法
8	c=a-b	无符号数减法
9	a-b	有符号数比较，小于时 c 置位
A	a-b	无符号数比较，小于时 c 置位
B	c =a	无操作
C	c=a	
D	c=a	
E	c=a	
F	c=a	

实验与设计

实验8-3. Stage3指令译码段实验

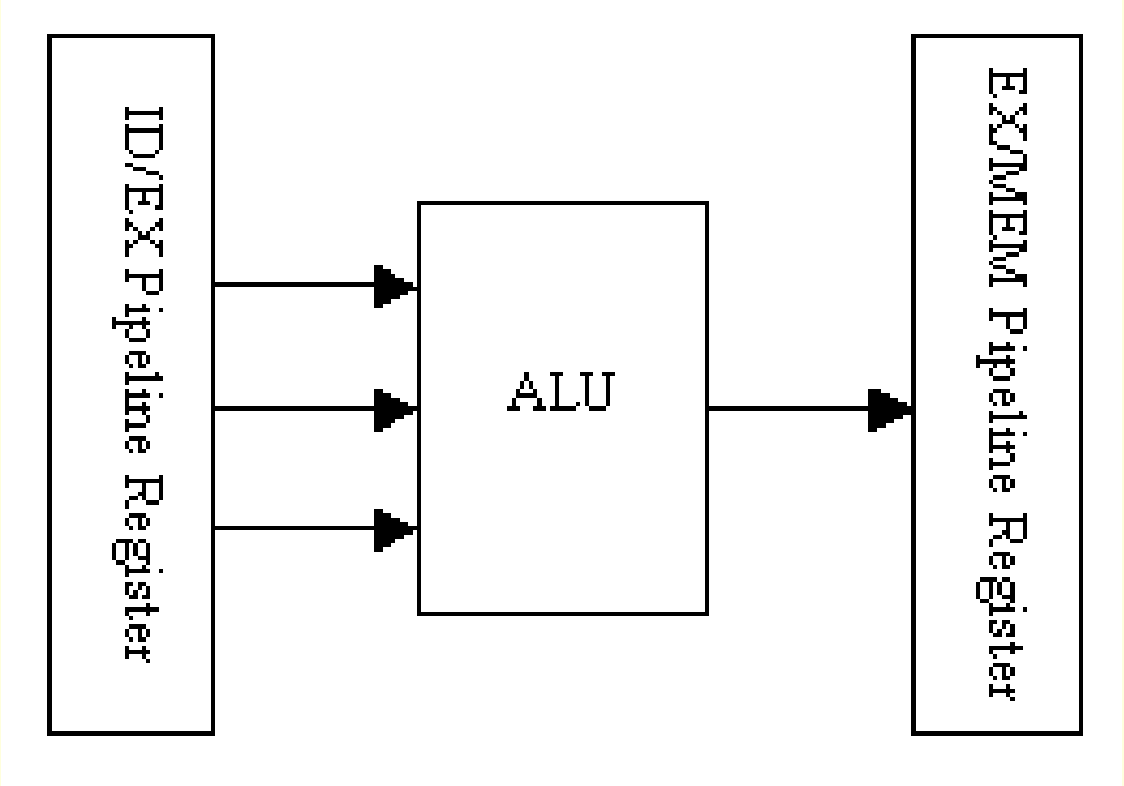


图8-71 stage 3执行EXE段的基本结构

实验与设计

实验8-3. Stage3指令译码段实验

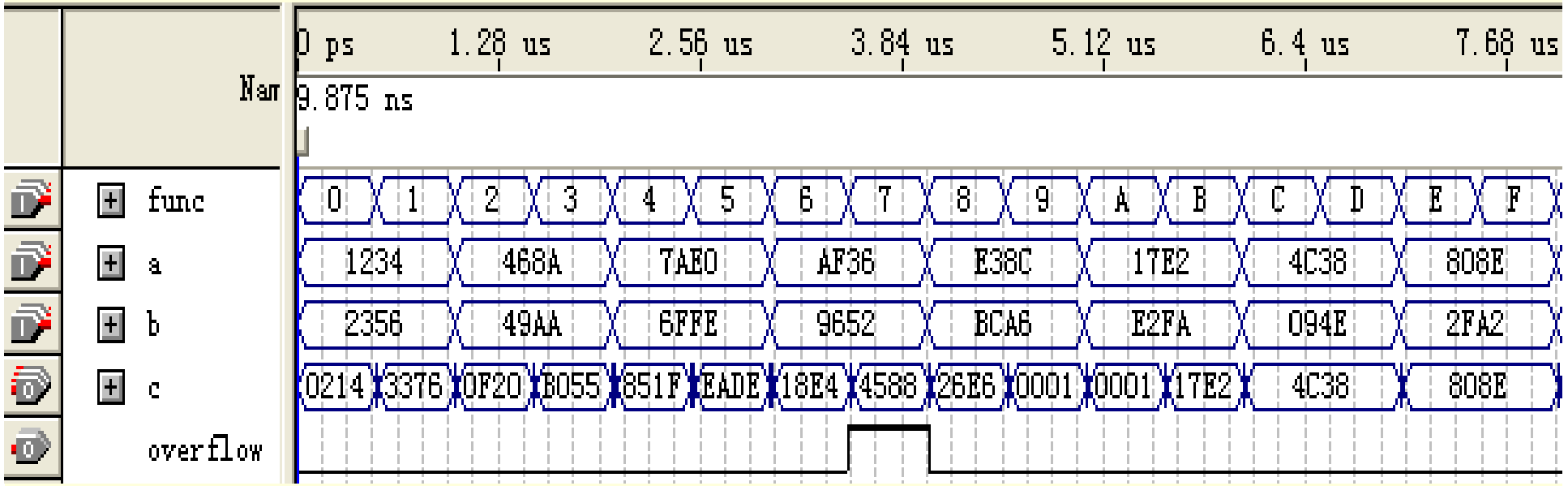


图8-72 alu_16的仿真波形

实验与设计

实验8-3. Stage3指令译码段实验

表8-4 移位运算器的功能

op	功　能	说　明
0	输出根据位移量进行数据输入或移位运算	有符号数逻辑左移
1		有符号数逻辑右移
2		有符号数算术右移
3		有符号数循环右移

实验与设计

实验8-3. Stage3指令译码段实验

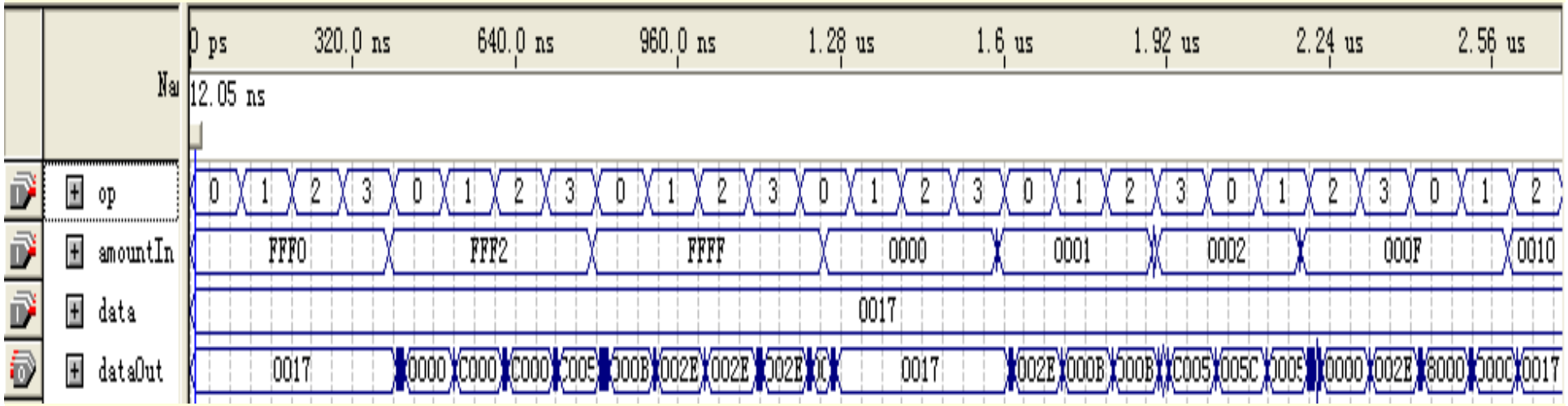


图8-73 移位运算器SU_3的仿真波形

实验与设计

实验8-3. Stage3指令译码段实验

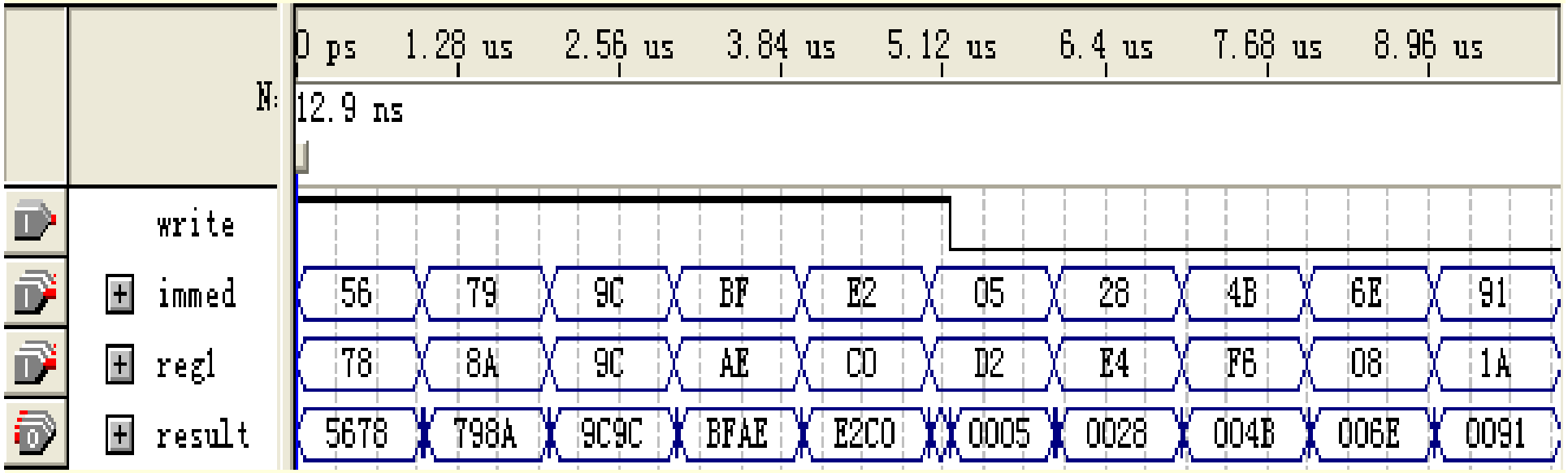


图8-74 数据输入模块movibox的仿真波形

实验与设计

实验8-3. Stage3指令译码段实验

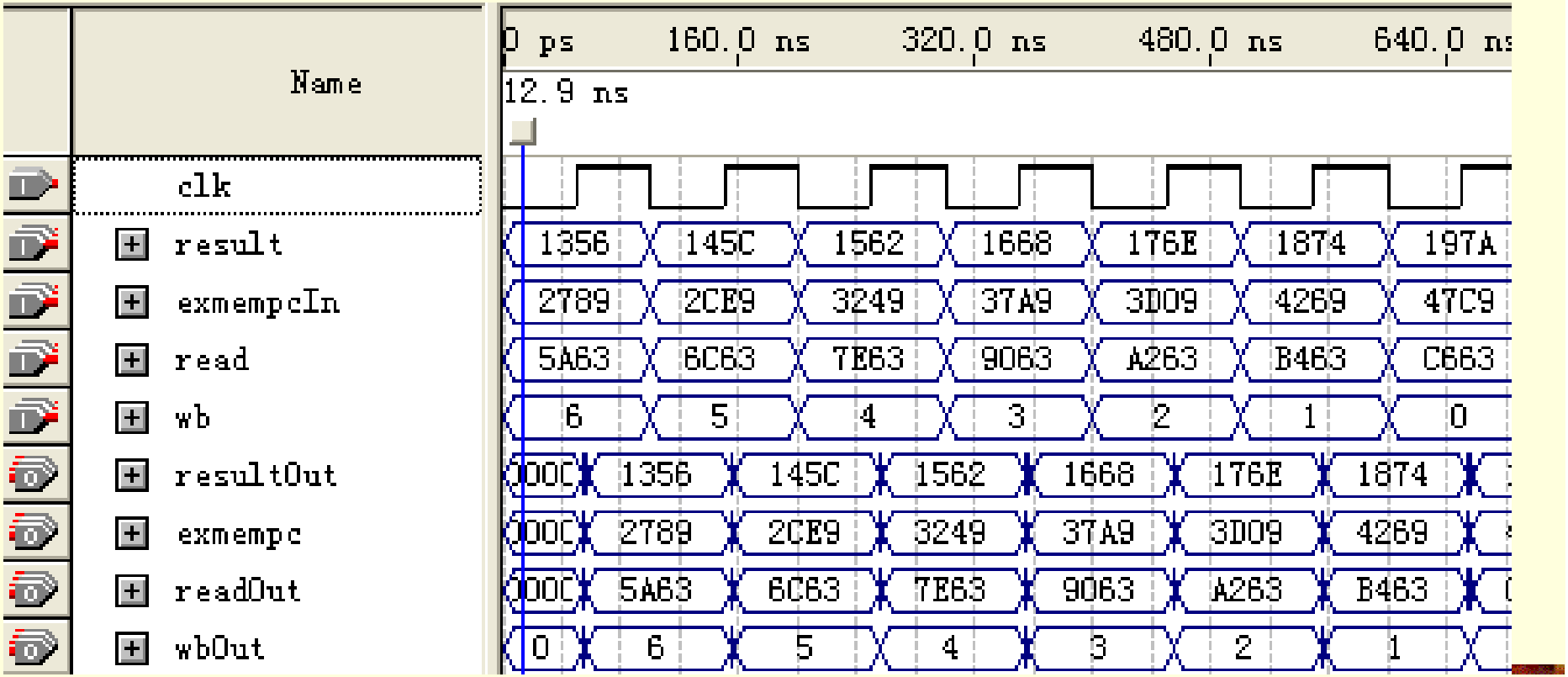


图8-75 EX/MEM段流水线寄存器的仿真波形

实验与设计

实验8-4. Stage4/5存储与写回段实验

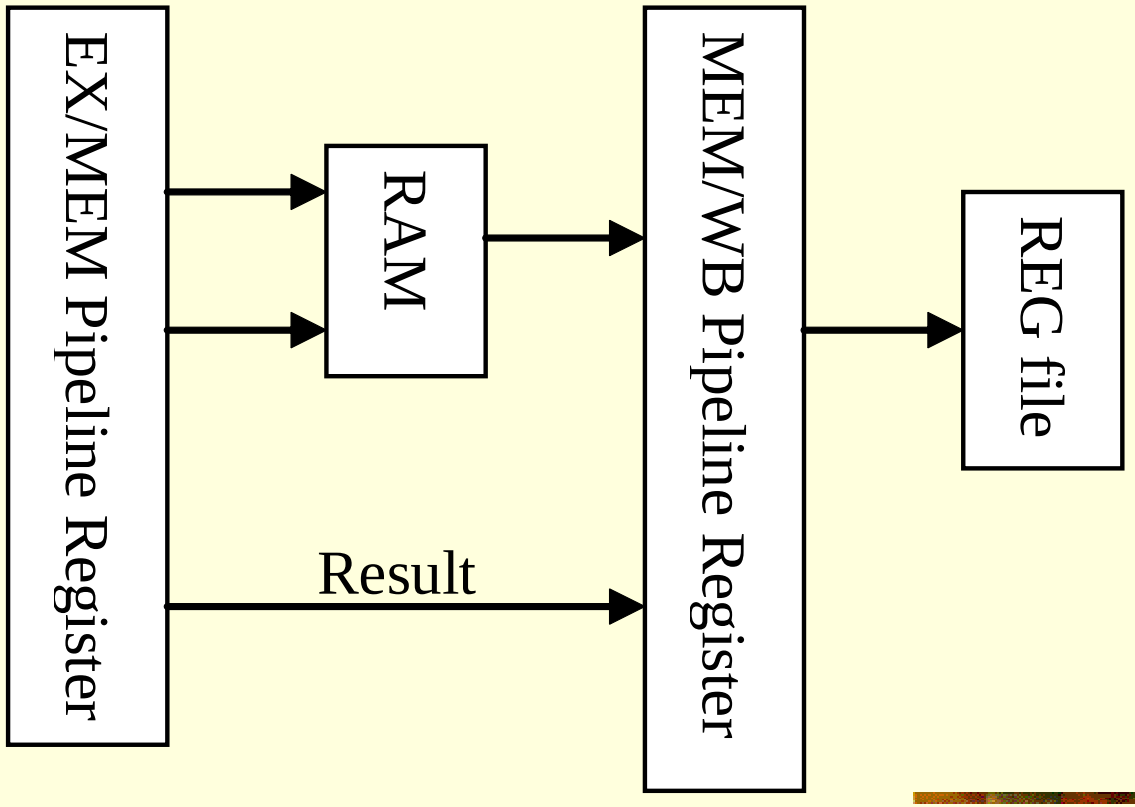


图8-76 Stage 4/5的主要部件

实验与设计

实验8-5. 数据相关性控制实验

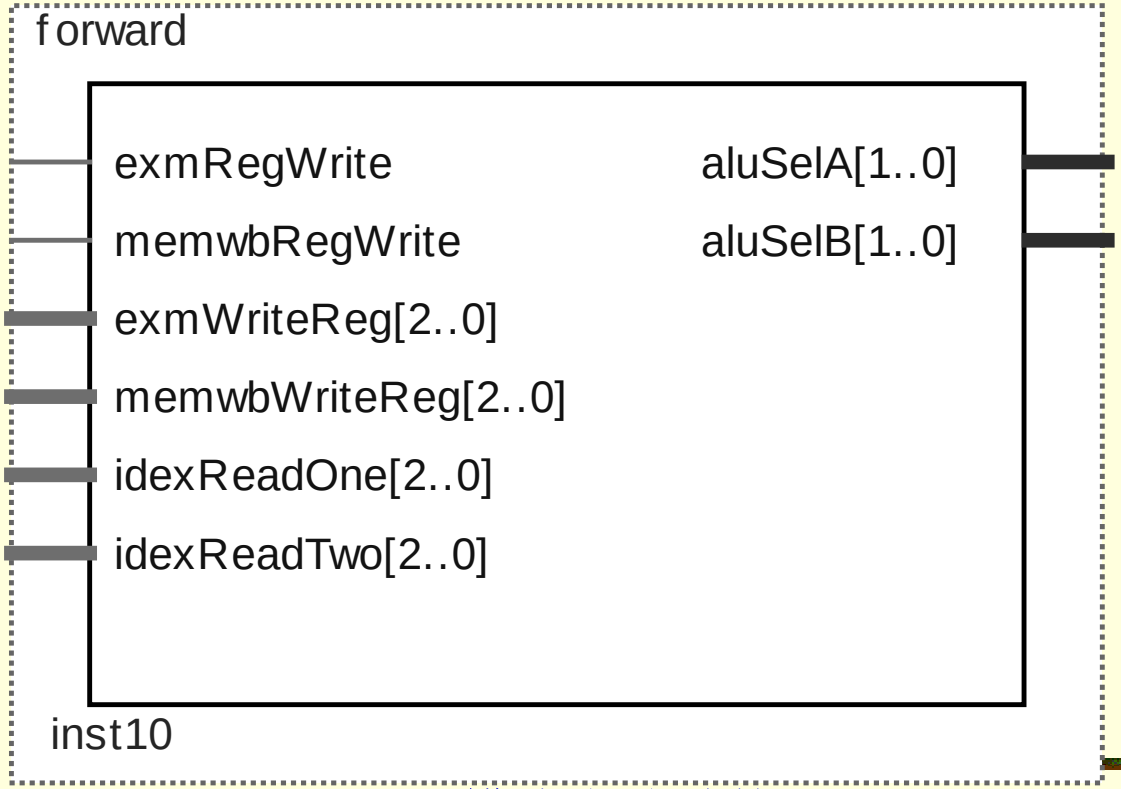


图8-77 forward模块电路结构

实验与设计

实验8-6. 数据通路实验

（1） 存储器访问指令(load/store ） **R1 ← (R2 + imm)**

（2） R型ALU操作 **R1 ← R2 op R3**

（3）R-I型ALU操作 **R1← R2 op imm**

（4）分支操作 **Branch condition ← R1 op 0**

(5) 跳转操作 JAL指令 **Branch condition← 0**

实验与设计

实验8-7. 流水线CPU综合实验

Instance Manager: Ready to acquire

Index	Instance ID	Status	Width	Depth	Type	Mode
0	rom3	Not running	16	64	RAM/ROM	Read/Write
1	ram1	Not running	16	64	RAM/ROM	Read/Write

AG ready

PT1]

20820DD)

MT\CH11\EX7_cpu16\cpu_16.sof

0 rom3:

```
000000  42 20  44 40  46 60  4A 02  B8 40  BF 60  09 3A  C8 80  02 6A  B D@F`J..@.`.:...j
000009  04 AA  0C 9F  00 00  00 00  5D EA  00 00  00 00  00 00  00 00  ........].........
000012  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00  ..................
00001B  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00  ..................
000024  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00  ..................
00002D  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00  ..................
000036  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00  ..................
00003F  00 00                                                          ..
```

1 ram1:

```
000000  00 11  00 12  00 13  00 14  00 15  00 16  00 17  00 18  00 19  ..................
000009  00 20  00 21  00 22  00 23  00 24  00 25  00 26  00 27  00 28  . .!."#.$.%.&.'.(
000012  00 29  00 30  00 31  00 32  00 33  00 34  00 35  00 36  00 37  .).0.1.2.3.4.5.6.7
00001B  00 38  00 39  00 40  00 41  00 42  00 43  00 44  00 45  00 46  .8.9.@.A.B.C.D.E.F
000024  00 47  00 48  00 49  00 4A  00 4B  00 4C  00 4D  00 4E  00 4F  .G.H.I.J.K.L.M.N.O
00002D  00 50  00 51  00 52  00 30  00 54  00 55  00 56  00 57  00 58  .P.Q.R.0.T.U.V.W.X
000036  00 59  00 5A  00 5B  00 5C  00 5D  00 5E  00 5F  00 60  00 61  .Y.Z.[.\.].^._.`.a
00003F  00 62                                                          .b
```

图8-78 程序运行前ROM和RAM中的数据

实验8-7. 流水线CPU综合实验

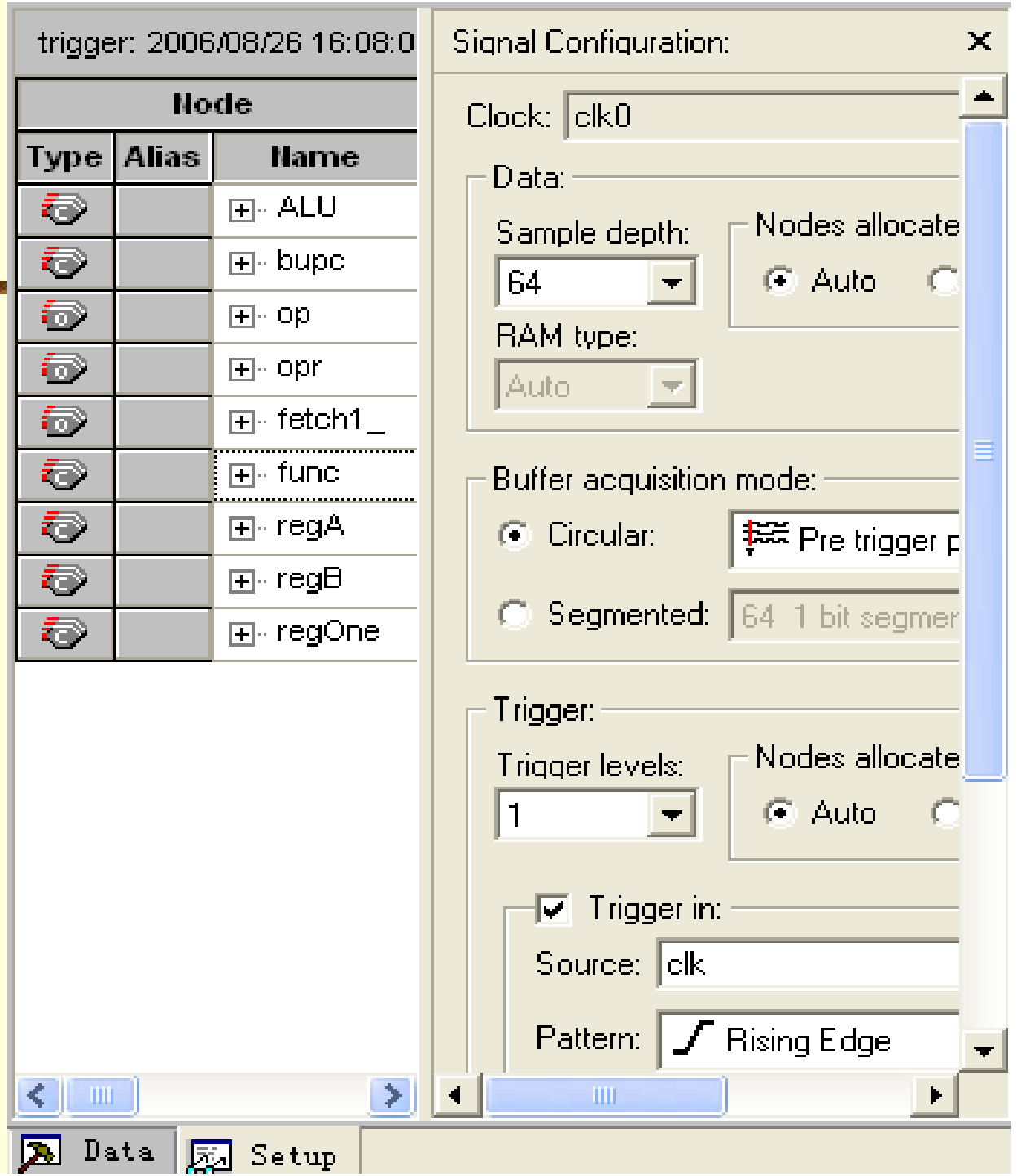


图8-79 嵌入式逻辑分析仪设置情况图

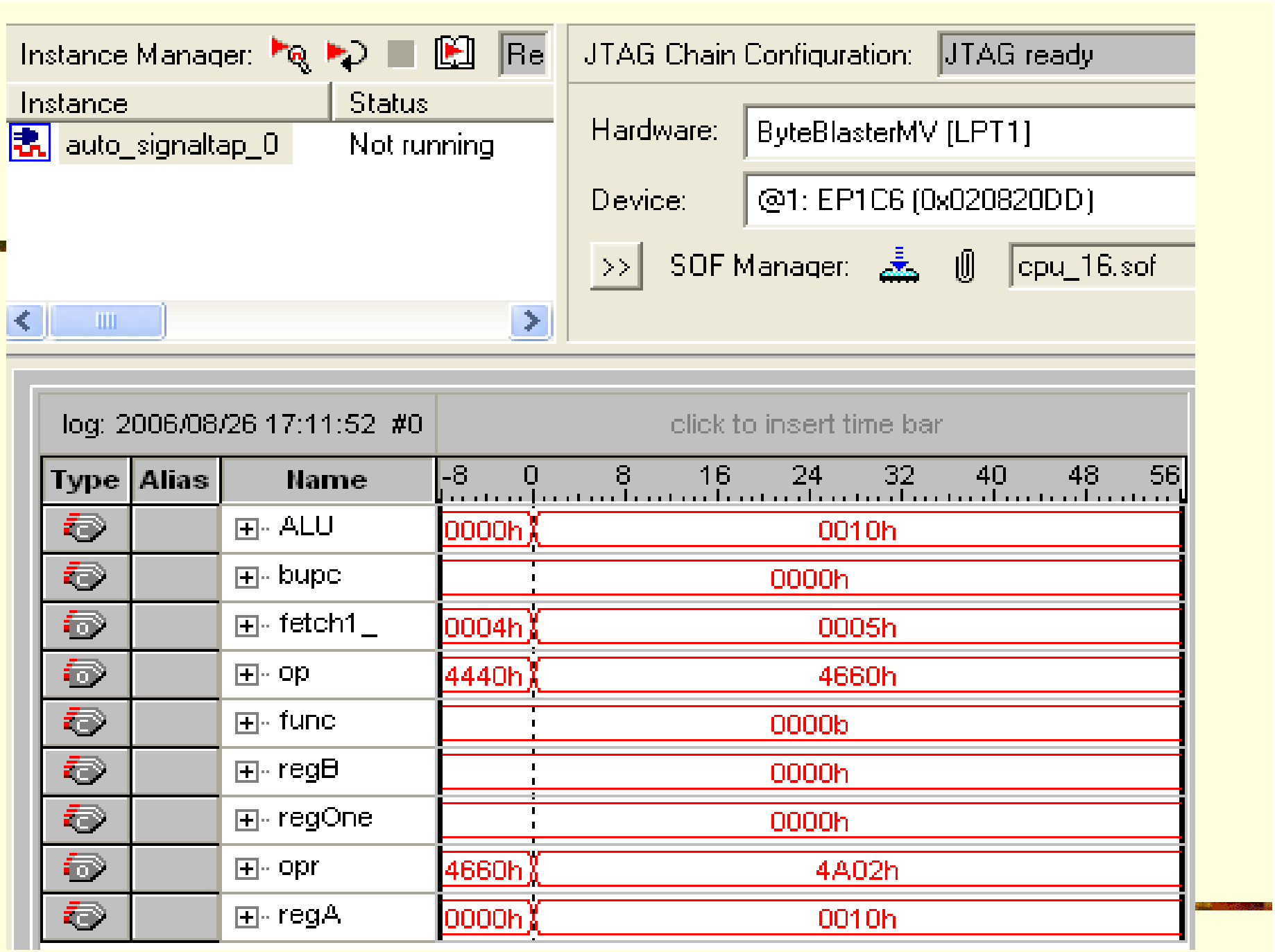


图8-80　CPU运行时逻辑分析仪显示波形

实验与设计

实验8-7. 流水线CPU综合实验

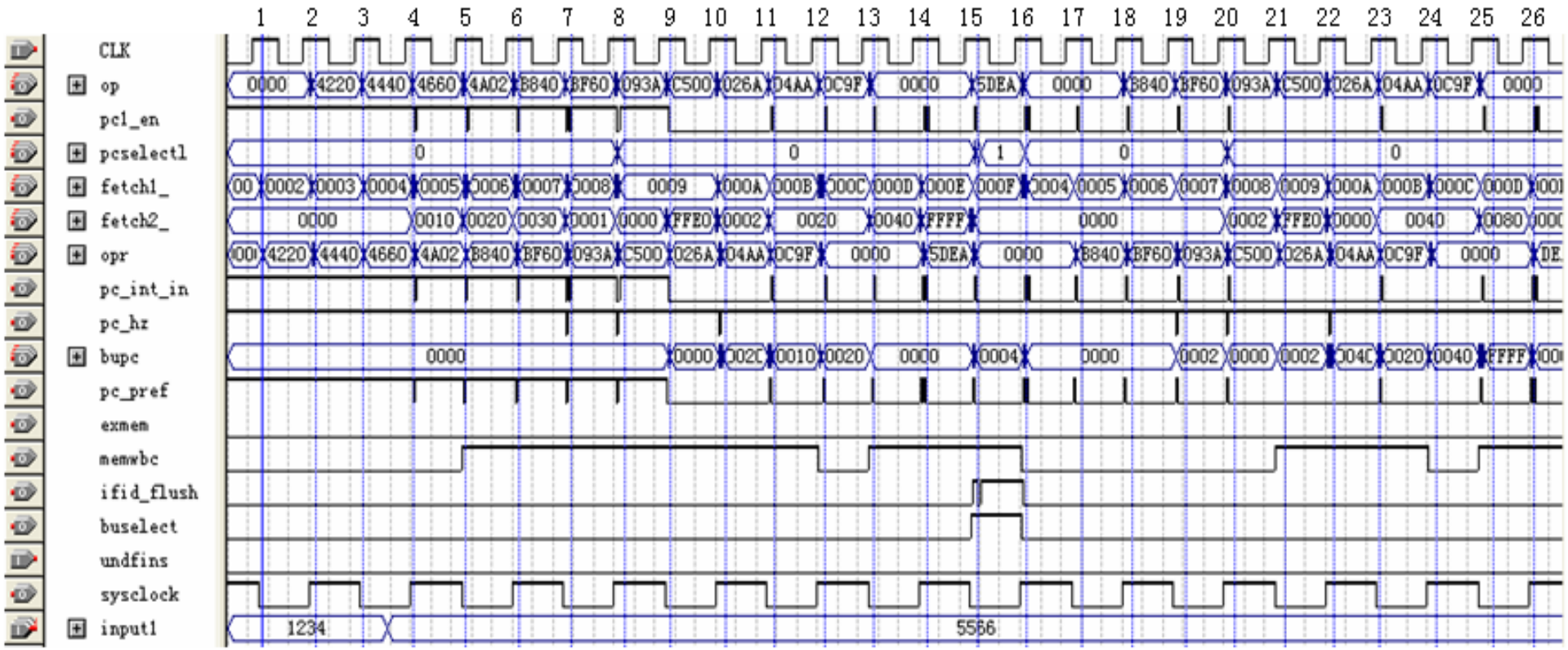


图8-81 流水线CPU的时序仿真波形

参考程序如下：

地址	机器码	指令	说　　明
00	4220	MVIL　R1,10H	R1是源指针，立即数10H送R1，
01	4440	MVIL　R2,20H	R2是目的指针，立即数20H送R2
02	4660	MVIL　R3,25H	R3是结束地址，立即数25H送R3
03	4A02	MVIL　R5,01`	常数01H送R5
04	4E0A	MVIL　R7,05H	常数05H送R7
05	B840	LOOP:LW　R4,R1,0	将R1的内容作为地址，从存储器中取数送R4，
06	093A	ADD　R4,R4,R7	R4←（R4）+（R7）
07	C500	SW　R2,R4,0	将R4的内容存到以R2的内容为地址的RAM存储单元
08	026A	ADD　R1,R1,R5	修改源指针
09	04AA	ADD　R2,R2,R5	修改目的指针
0A	0C9F	SLT　R6,R2,R3	比较结束地址R6←（R2）-（R3）
0B	5DEA	BZI　R6,LOOP	判断、程序转移，若R6< 0，则转到LOOP
0C	0000	NOP	结束
0D-3F	0000		