第8章

流水线CPU设计

8.2 指令系统设计

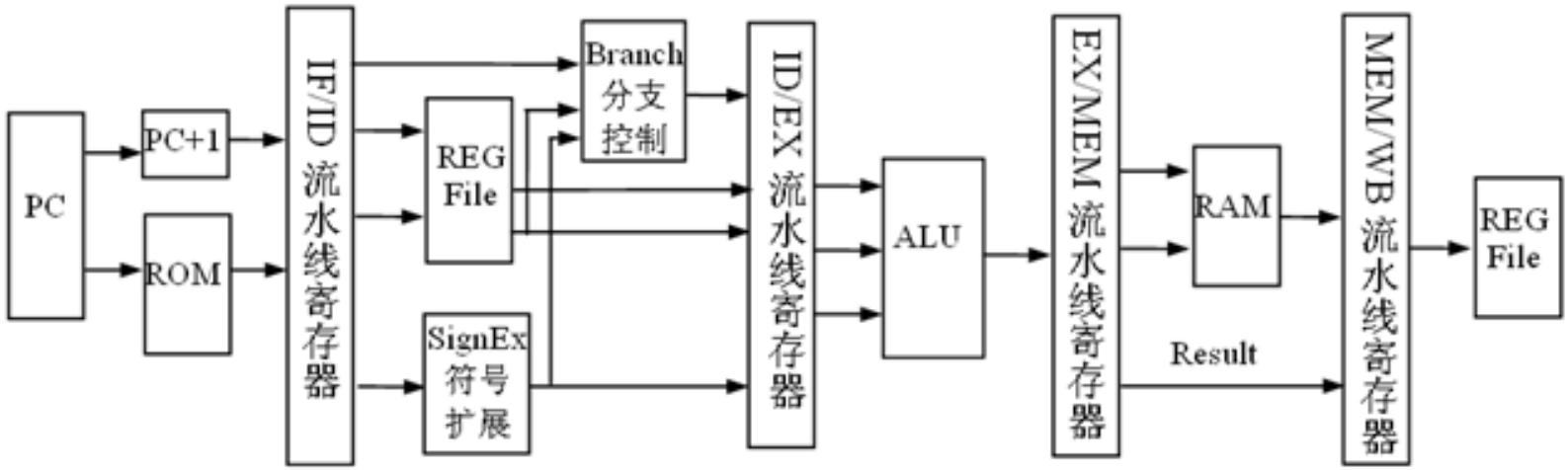


图 8-1 流水线 CPU 的结构

8.2 指令系统设计

表 8-1　指令的形式及功能

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

8.2 指令系统设计

0010	IN R1	R	从端口输入	000
	OUT R1	R	输出到端口	001
	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

1、寄存器型 (R-型)

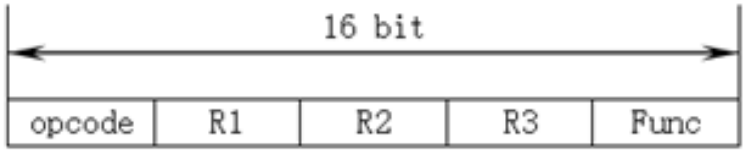


图 8-2　R-型指令 ADD　R1,R2,R3

2、寄存器立即数型(RI-型)

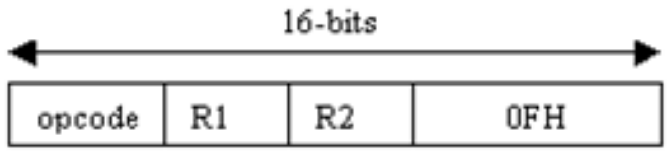


图 8-3　RI-型指令 ADDI　R1,R2,0FH

8.2 指令系统设计

3、立即数型 (I-型)

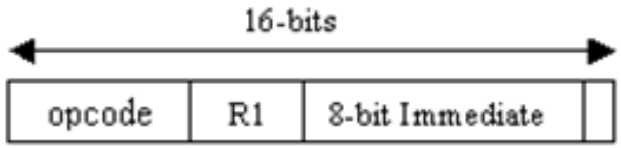


图 8-4　I-型指令　MVIL　R1,FFH

4、立即移位型 (SI-型)

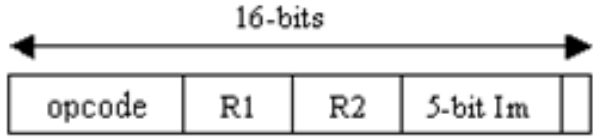


图 8-5　SI-型指令　RORI　R1,R2,2H

8.3 数据通路设计

1、R-型数据通路

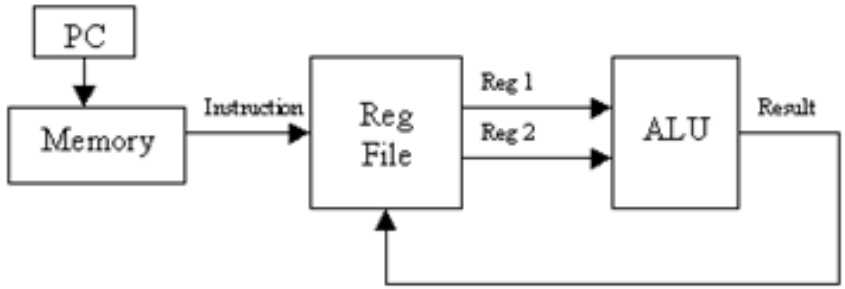


图 8-6　R-型 ALU 指令的数据通路

2、RI-型数据通路

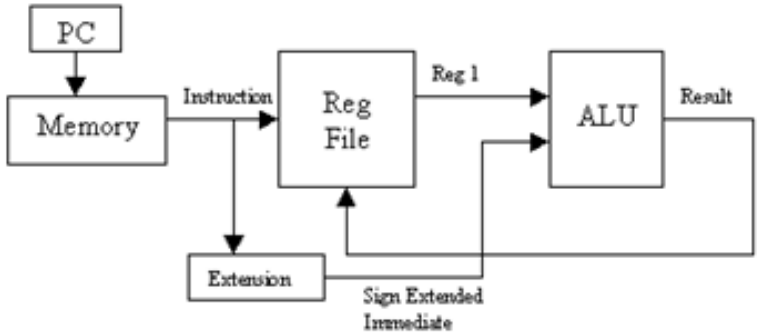


图 8-7　RI-型 ALU 指令的数据通路

8.3 数据通路设计

3、装入字数据通路

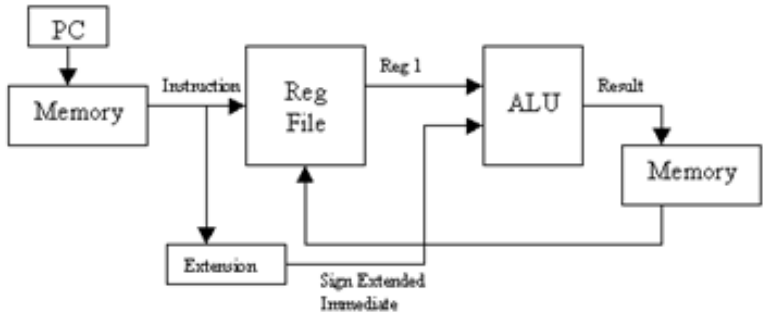


图 8-8　装入字数据通路

4、存储字数据通路

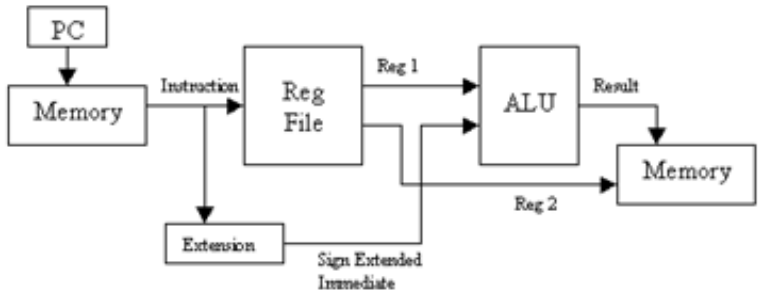


图 8-9 存储指令的数据通路

8.3 数据通路设计

5、寄存器转移数据通路

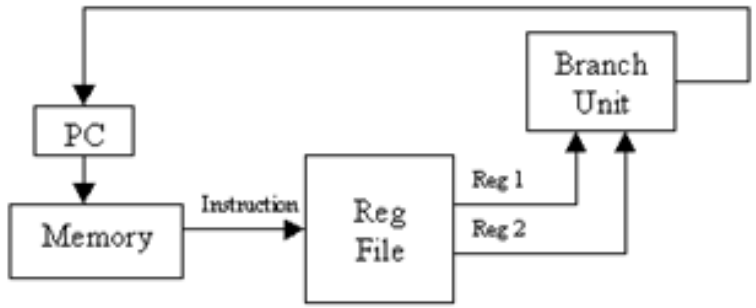


图 8-10　转移指令数据通路

8.4 流水线各段的功能描述与设计

8.4.1 Stage 1取指令段

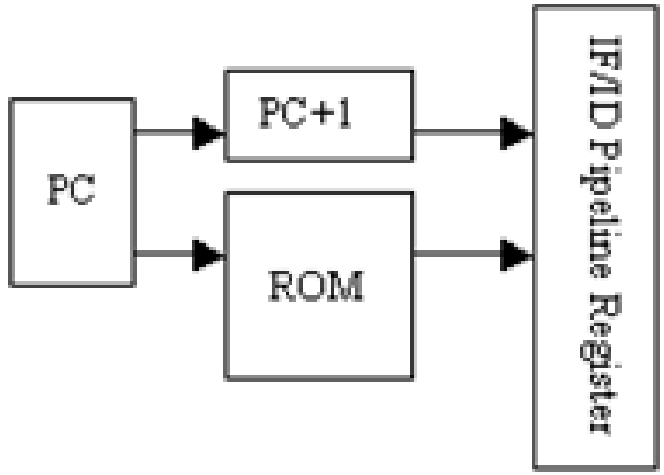


图 8-11　IF Stage 1 的结构

8.4 流水线各段的功能描述与设计

8.4.1 Stage 1取指令段

1. 功能描述

取指令及锁存

地址计算

检验指令的合法性

同步控制

8.4 流水线各段的功能描述与设计

8.4.1 Stage 1取指令段

2. 模块划分和实现

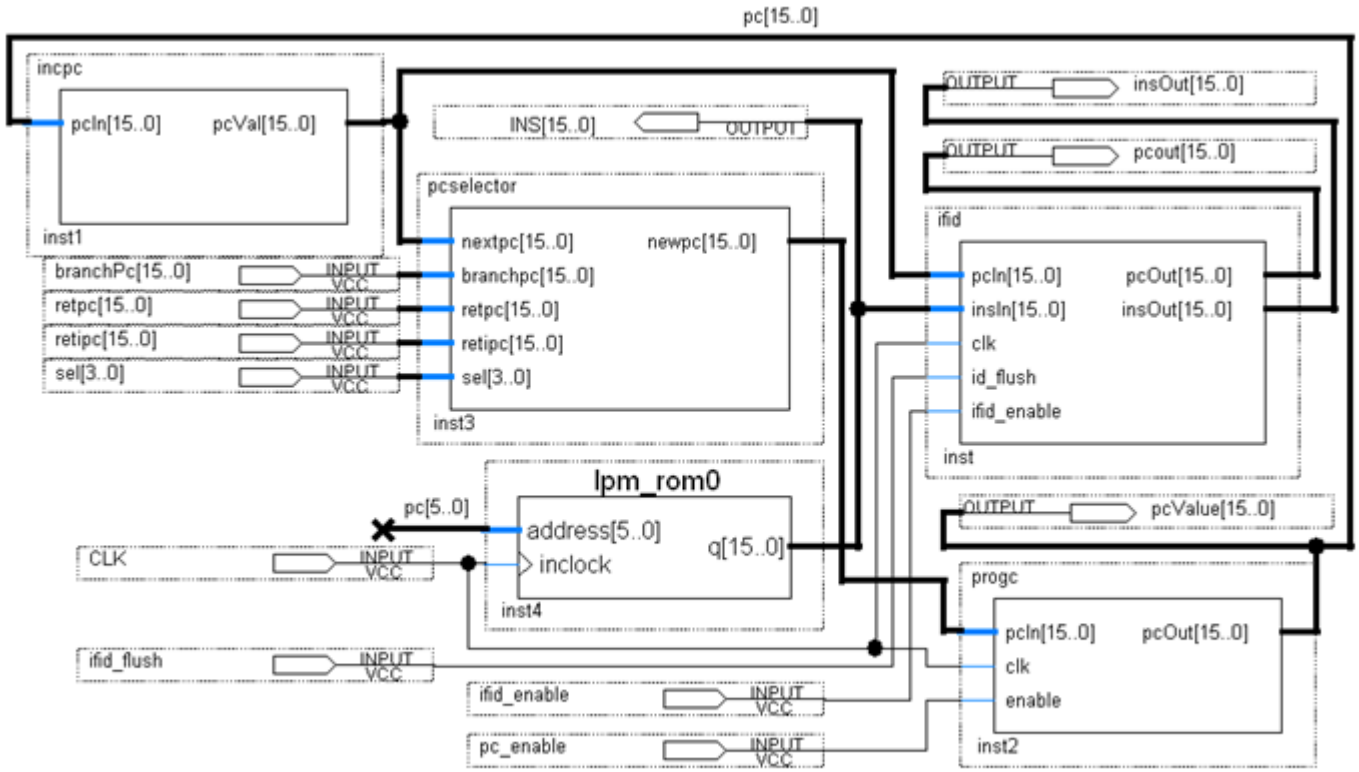


图 8-12　Stage 1 取指令段的各模块信号连接电路

8.4 流水线各段的功能描述与设计

8.4.1 Stage 1取指令段

2. 模块划分和实现

【例 8-1】

```
module t pcselector (nextpc, branchpc, retpc, retipc, sel, newpc);
  input[15:0] nextpc, branchpc, retpc, retipc;  input[3:0] sel;
  output[15:0] newpc;   reg[15:0] newpc;
  always @(sel)
    case (sel)
      4'b0000 :  newpc <= nextpc ;       4'b0001 :  newpc <= branchpc;
      4'b0010 :  newpc <= retpc ;        4'b0011 :  newpc <= retipc;
      4'b0100 :  newpc <= 16'HFFFF;      4'b0101 :  newpc <= 16'HFFF0;
      4'b0110 :  newpc <= 16'H0008;      4'b0111 :  newpc <= 16'H000C;
      4'b1000 :  newpc <= 16'H000C;      4'b1001 :  newpc <= 16'H000E;
      4'b1010 :  newpc <= 16'H0010;     default :  newpc <= 16'H0012;
     endcase
endmodule
```

8.4 流水线各段的功能描述与设计

8.4.1 Stage 1取指令段

2. 模块划分和实现

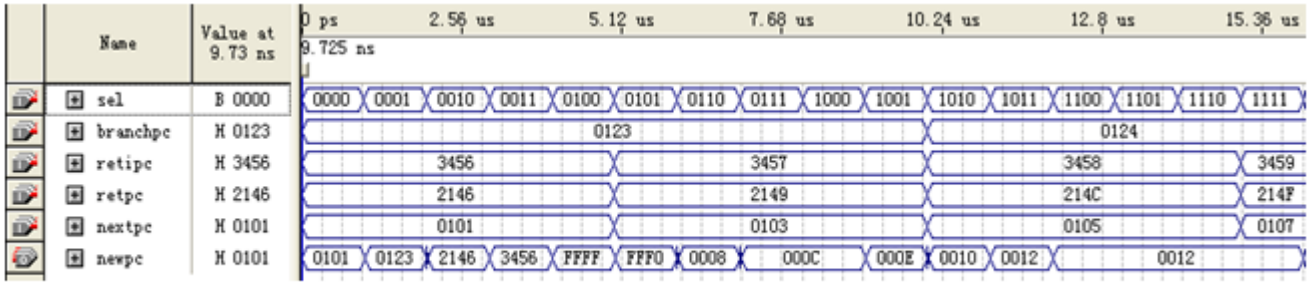


图 8-13　pcselector 的时序仿真波形图

8.4 流水线各段的功能描述与设计

8.4.1 Stage 1取指令段

2. 模块划分和实现

【例 8-2】

```
module progc (input[15:0] pcIn,input clk, enable, output reg[15:0] pcOut);
  reg[15:0] regValue;
   always @(posedge clk)
       if (enable==1'b1) begin  regValue=pcIn;  pcOut=regValue;  end
endmodule
```

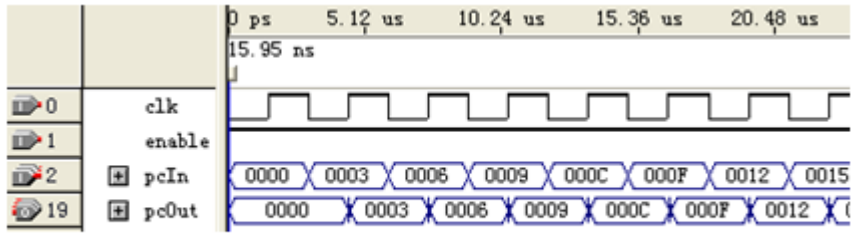


图 8-14　progc 的时序仿真图

8.4 流水线各段的功能描述与设计

8.4.1 Stage 1取指令段

2. 模块划分和实现

【例 8-3】

```
module t incpc (input[15:0] pcIn, output reg[15:0] pcVal);
  always @(pcIn) pcVal <= pcIn + 1 ;
endmodule
```

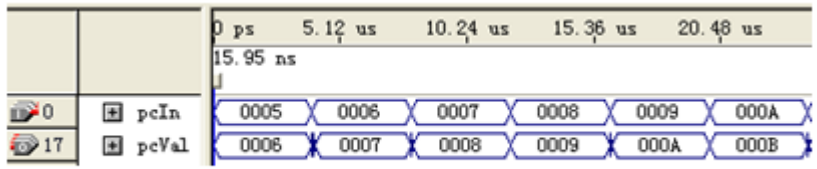


图 8-15　程序计数器加 1 模块的仿真波形

8.4 流水线各段的功能描述与设计

8.4.1 Stage 1取指令段

2. 模块划分和实现

【例 8-4】

```
module ifid(pcIn, insIn, clk, id_flush, ifid_enable, pcOut, insOut);
  input[15:0] pcIn, insIn; input clk, id_flush, ifid_enable;
  output[15:0] pcOut,insOut;  reg[15:0] pcOut,insOut; reg[31:0] regValue;
 always @(posedge clk)  begin
       if (ifid_enable==1'b1)   begin
          if (id_flush==1'b1)  regValue[31:0]={regValue[31:16],16'H0000};
             else   regValue[31:0] = {pcIn[15:0], insIn[15:0]};    end
              pcOut<=regValue[31:16]; insOut<=regValue[15:0] ;    end
  endmodule
```

8.4 流水线各段的功能描述与设计

8.4.1 Stage 1取指令段

2. 模块划分和实现

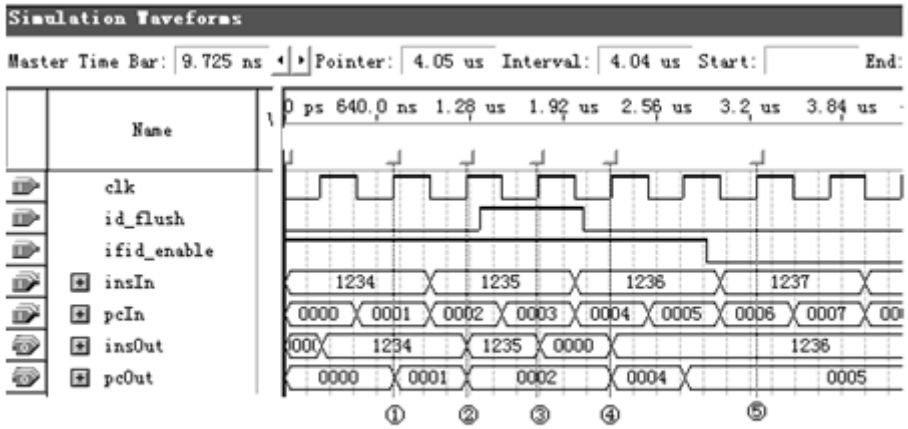


图 8-16　ifid 流水线寄存器的时序仿真波形图

8.4 流水线各段的功能描述与设计

8.4.2 Stage 2译码段ID

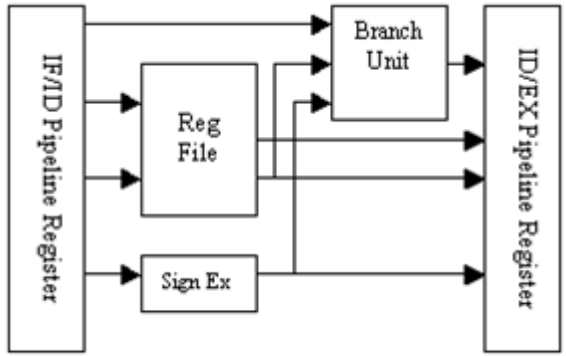


图 8-17 stage 2 指令译码段 ID 的基本结构

1. stage2的功能描述

8.4 流水线各段的功能描述与设计

8.4.2 Stage 2译码段ID

2．模块划分和实现

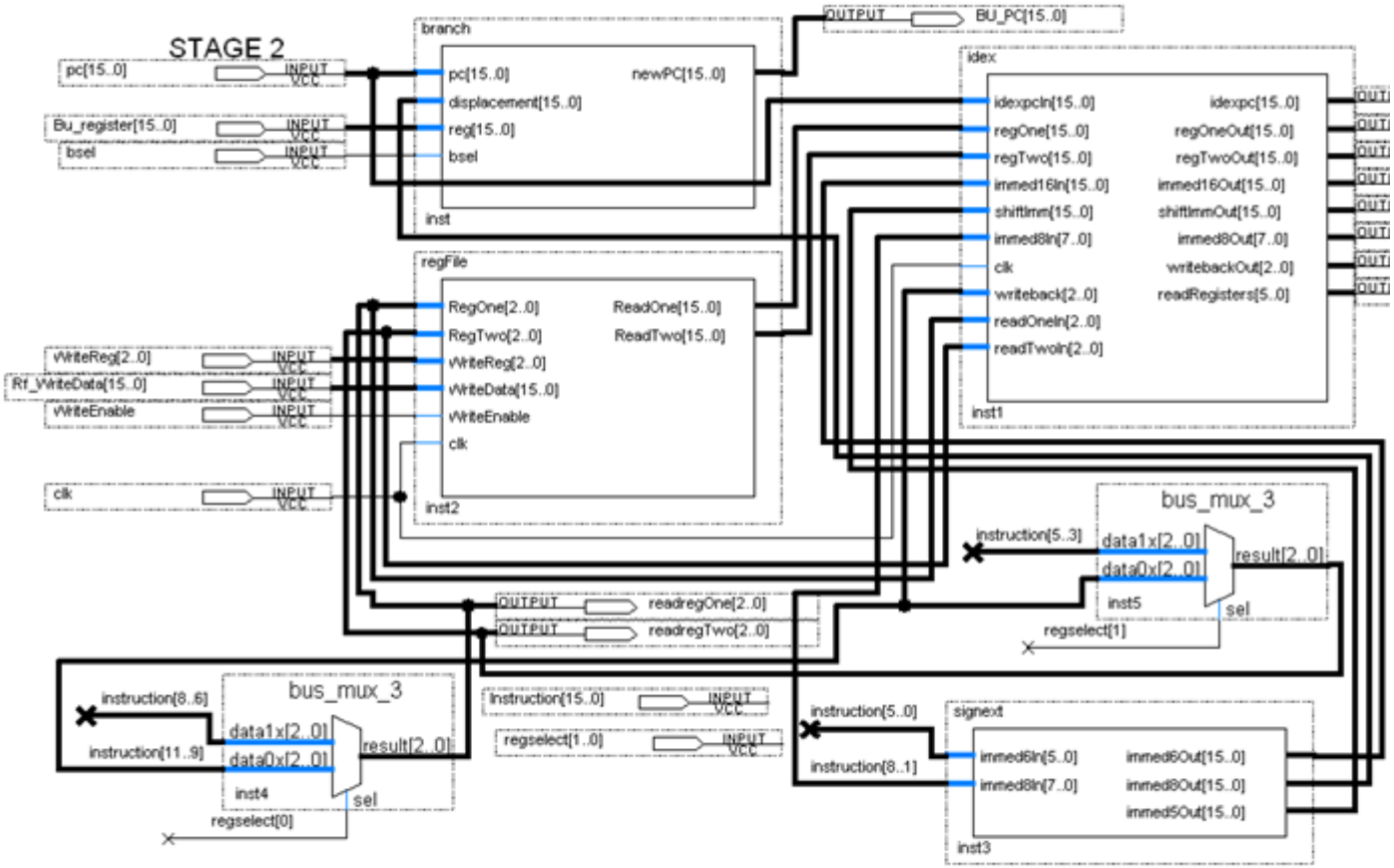


图 8-18　Stage 2 译码段 ID 的各模块信号连接电路

8.4 流水线各段的功能描述与设计

8.4.2 Stage 2译码段ID

2．模块划分和实现

（1）符号扩展模块。

【例 8-5】

```
module signext (immed6In, immed8In, immed6Out, immed8Out, immed5Out);
 input[5:0] immed6In; input[7:0] immed8In; output[15:0] immed6Out;
   output[15:0] immed8Out, immed5Out;
   reg[15:0] immed8Out, immed6Out, immed5Out;
  always @(immed6In or immed8In)   begin
      if ((immed6In[5])==1'b0)   immed6Out<={10'H000, immed6In[5:0]};
      else  immed6Out<={10'H3FF, immed6In[5:0]} ;
      if ((immed8In[7])==1'b0)   immed8Out<={8'H00, immed8In[7:0]} ;
      else immed8Out<={8'H11, immed8In[7:0]} ;
      if ((immed8In[4])==1'b0) immed5Out<={11'H000, immed8In[5:1]} ;
      else  immed5Out<={11'H7FF, immed8In[5:1]};   end
endmodule
```

（2）寄存器文件模块。

【例 8-6】

```
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;
       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;
```

（2）寄存器文件模块。

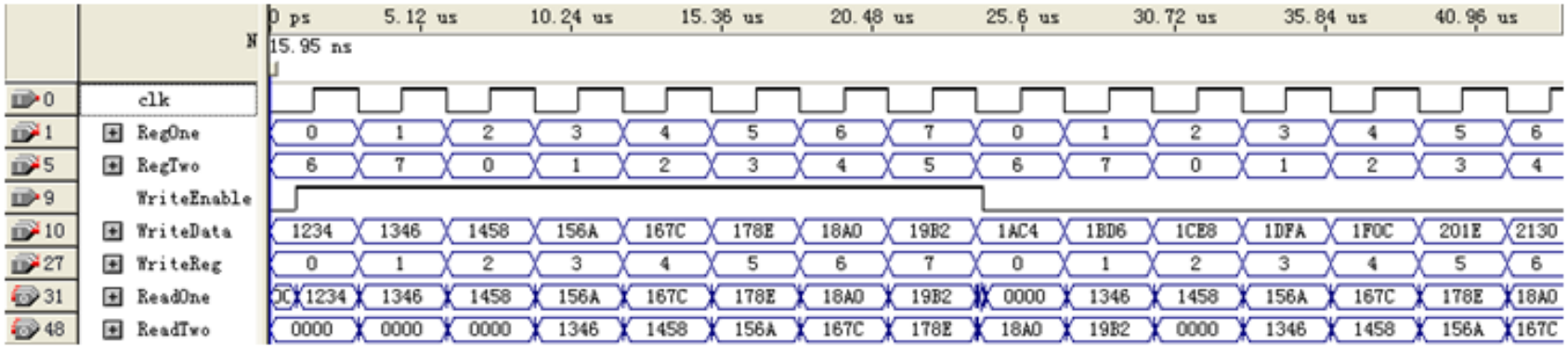


图 8-9　regFile 的时序仿真图

1. 读寄存器。从寄存器读出数据，分为两种情况

① 当**WriteEnable**为高电平时，即写允许有效。

② 当**WriteEnable**为低电平时，即写允许无效。

2. 写寄存器。向寄存器写入数据，也分为两种情况

① 写入信号有效。

② 当写信号无效。

8.4 流水线各段的功能描述与设计

8.4.2 Stage 2译码段ID

2．模块划分和实现

（3）分支控制模块。

【例 8-7】

```
module branch (input[15:0] pc, displacement, reg_xhdl0, input bsel,
               output reg[15:0] newPC);
   always @(pc or displacement or reg_xhdl0 or bsel)
      if (bsel==1'b1) newPC<=pc+displacement; else newPC<=reg_xhdl0 ;
  endmodule;
```

8.4 流水线各段的功能描述与设计

（4）ID/EX段的流水线寄存器。

【例 8-8】

```
module idex  (idexpcIn, regOne, regTwo, immed16In, shiftImm, immed8In, clk,
    writeback, readOneIn, readTwoIn, idexpc, regOneOut, regTwoOut,
    immed16Out, shiftImmOut, immed8Out, writebackOut, readRegisters);
     input[15:0] idexpcIn,regOne, regTwo, immed16In, shiftImm;
     input[7:0] immed8In; input clk; input[2:0] writeback, readOneIn, readTwoIn;
      output[15:0] idexpc,regOneOut, regTwoOut,immed16Out, shiftImmOut;
      output[7:0] immed8Out;  output[2:0] writebackOut;
      output[5:0] readRegisters;
      reg[15:0] idexpc, regOneOut, regTwoOut,immed16Out, shiftImmOut;
      reg[7:0] immed8Out; reg[2:0] writebackOut; reg[5:0] readRegisters;
      reg[96:0] regValue;
       always @(clk)  begin
         if (clk==1'b1) regValue[96:0]={idexpcIn[15:0], readOneIn[2:0],
            readTwoIn[2:0], shiftImm[15:0], regOne[15:0],regTwo[15:0],
            immed16In[15:0], immed8In[7:0], writeback[2:0]};
          idexpc <= regValue[96:81]; readRegisters <= regValue[80:75] ;
          shiftImmOut <= regValue[74:59]; regOneOut <= regValue[58:43] ;
          regTwoOut <= regValue[42:27]; immed16Out <= regValue[26:11] ;
          immed8Out<=regValue[10:3]; writebackOut<=regValue[2:0];  end
    endmodule
```

8.4 流水线各段的功能描述与设计

8.4.3 Stage 3执行有效地址计算段(EXE)

1．功能描述

（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：Branch condition ← 0

8.4 流水线各段的功能描述与设计

8.4.3 Stage 3执行有效地址计算段(EXE)

2. 模块划分和实现

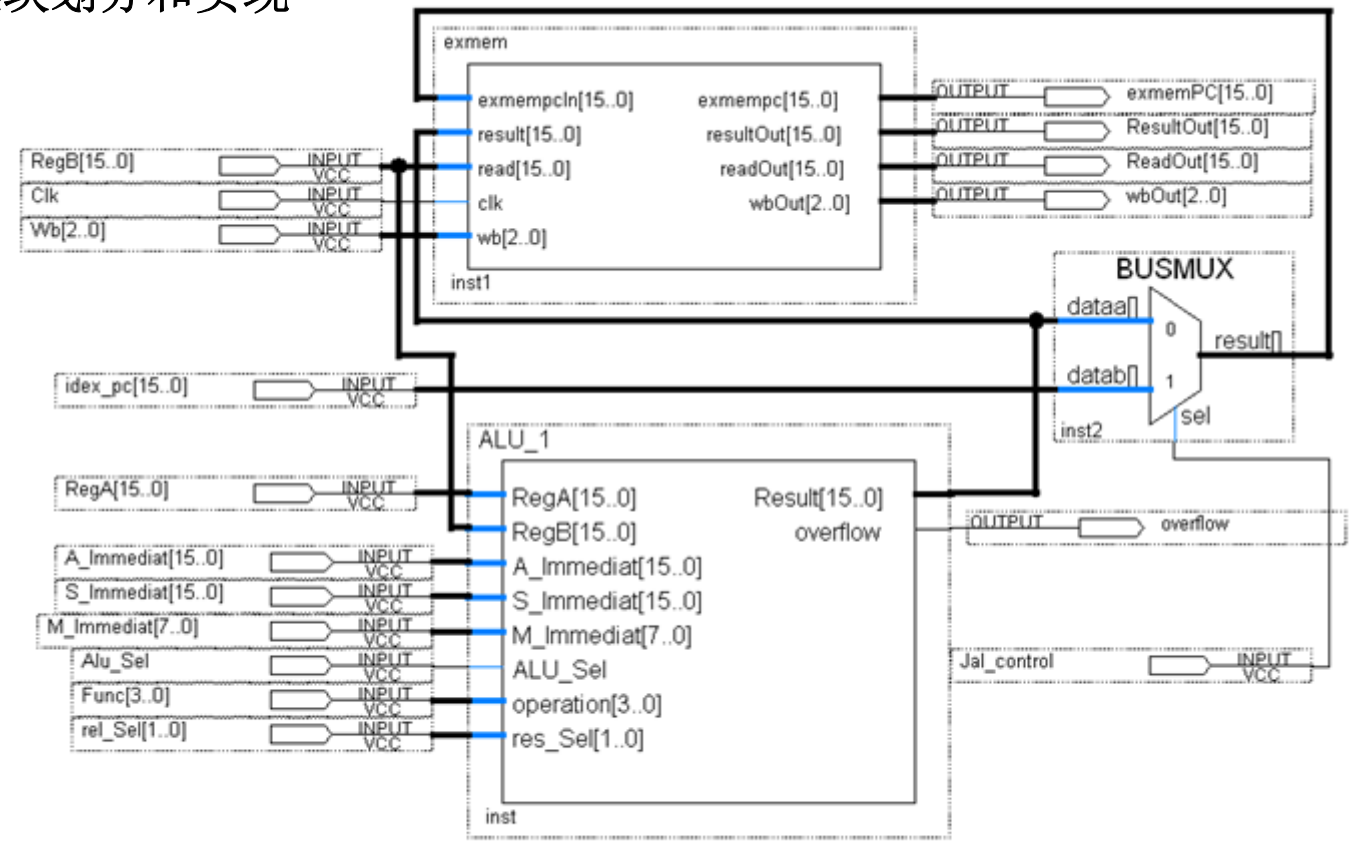


图 8-11　Stage 3 EXE 执行段的各模块信号连接电路

8.4 流水线各段的功能描述与设计

2. 模块划分和实现

（1）**EX/MEM**段流水线寄存器。

【例 8-9】

```
module  exmem(exmempcIn, result, read, clk, wb, exmempc, resultOut,
                  readOut, wbOut);
 input[15:0] exmempcIn, result, read;  input clk; input[2:0] wb;
 output[15:0] exmempc, resultOut, readOut; output[2:0] wbOut;
 reg[15:0] exmempc,resultOut,readOut;reg[2:0] wbOut;reg[50:0] regValue;
   always @(clk)   begin
     if (clk == 1'b1)
     regValue[50:0]={exmempcIn[15:0], result[15:0], read[15:0], wb[2:0]};
      exmempc <= regValue[50:35] ; // EX 段的 PC 值
      resultOut <= regValue[34:19] ; //来自 ALU 的运算结果
      readOut <= regValue[18:3] ; //从 ALU 读出的数据
      wbOut <= regValue[2:0] ; //回写的寄存器
   end
endmodule
```

8.4 流水线各段的功能描述与设计

2. 模块划分和实现

（2）运算模块**ALU**：

（3）算术**/**逻辑运算模块：

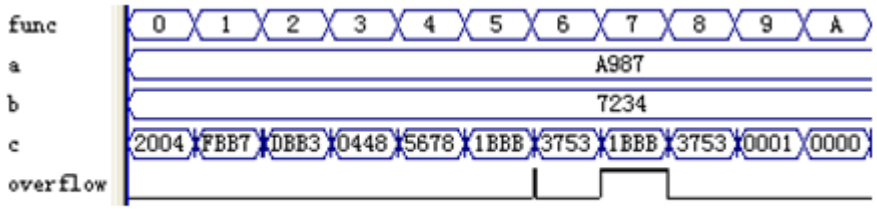


图 8-12　alu_16 模块的仿真波形

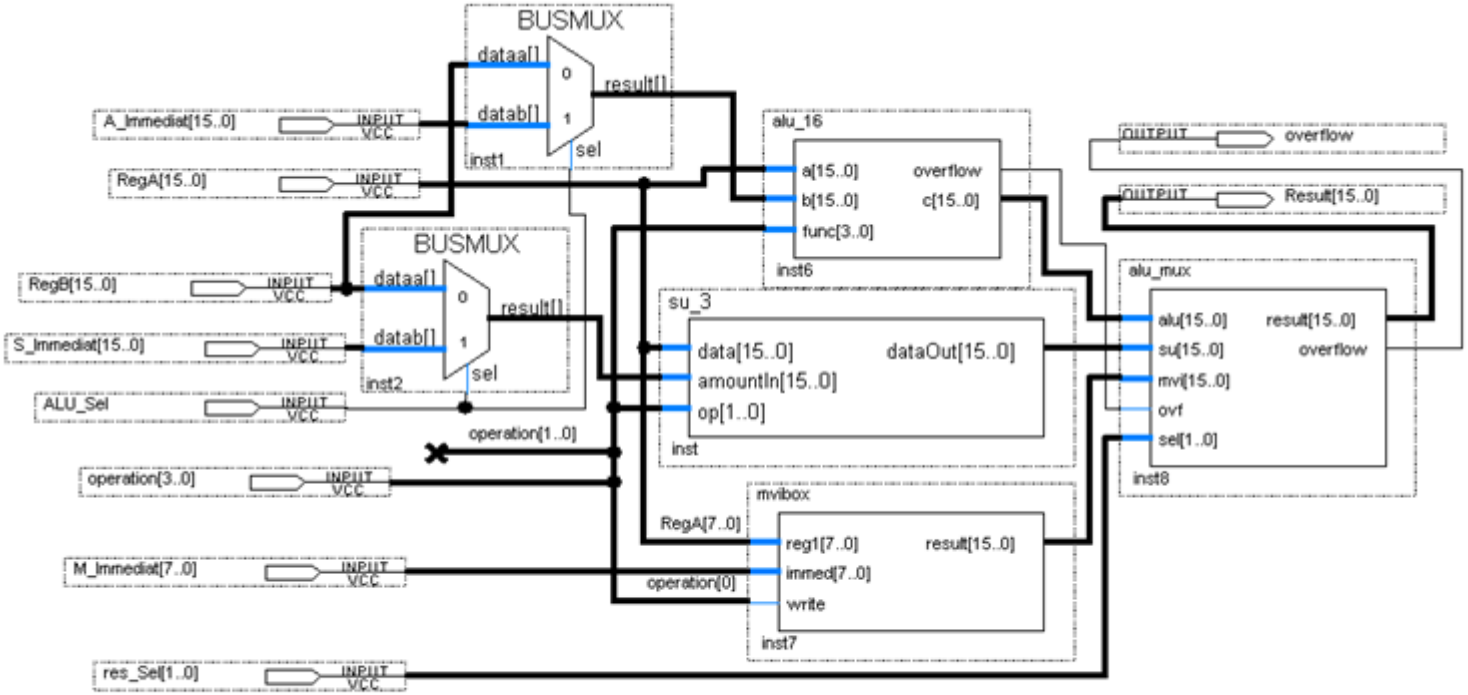


图 8-13　运算模块 ALU 的内部结构图

8.4 流水线各段的功能描述与设计

2. 模块划分和实现

（4）移位运算模块：

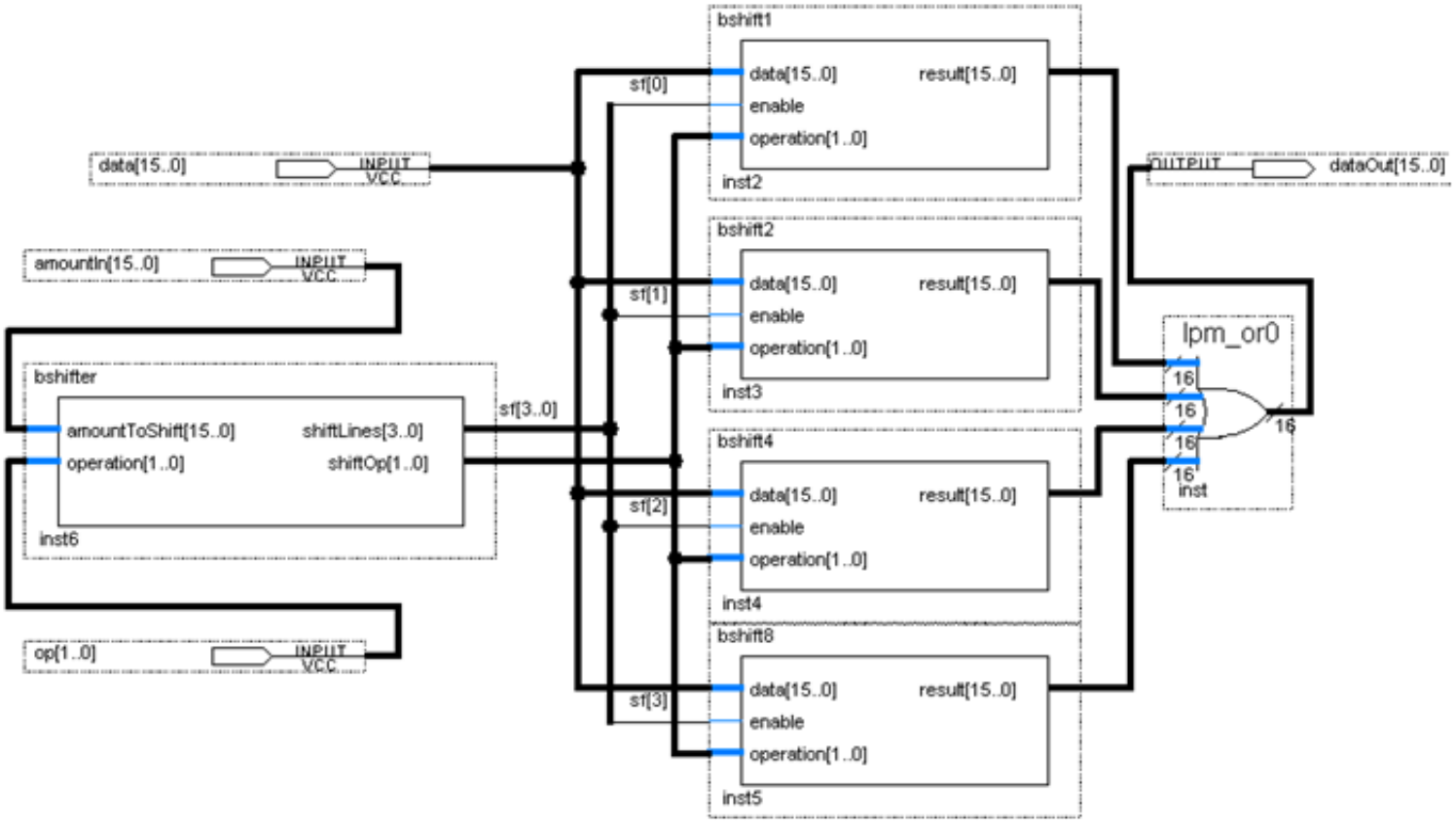


图 8-14　su_3 移位运算模块内部结构图

【例 8-10】

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.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
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';
WHEN "0001" => c <= a or b;  overflow <= '0';
WHEN "0010" => c <= a xor b; overflow <= '0';
WHEN "0011" => c <= a nor b; overflow <= '0';
WHEN "0100" => c <= not a;   overflow <= '0';
```

接下页

```
WHEN "0101" => signedResult := conv_signed(conv_integer(signed(a))
       +conv_integer(signed(b)),16);
      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);
        c<=conv_std_logic_vector(signedResult,16); overflow <= '0';
WHEN "0111" => 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;
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 ;
END ARCHITECTURE alu_behav;
```

8.4 流水线各段的功能描述与设计

2. 模块划分和实现

（5）寄存器数据输入模块：

【例 8-11】

```
module movibox (input[7:0] reg1,immed,input write,output reg[15:0] result);
  always @(reg1 or immed or write)
    if (write==1'b0) result<={8'H00,immed}; else result<={immed,reg1};
endmodule
```

8.4 流水线各段的功能描述与设计

8.4.4 stage4访存段(MEM)

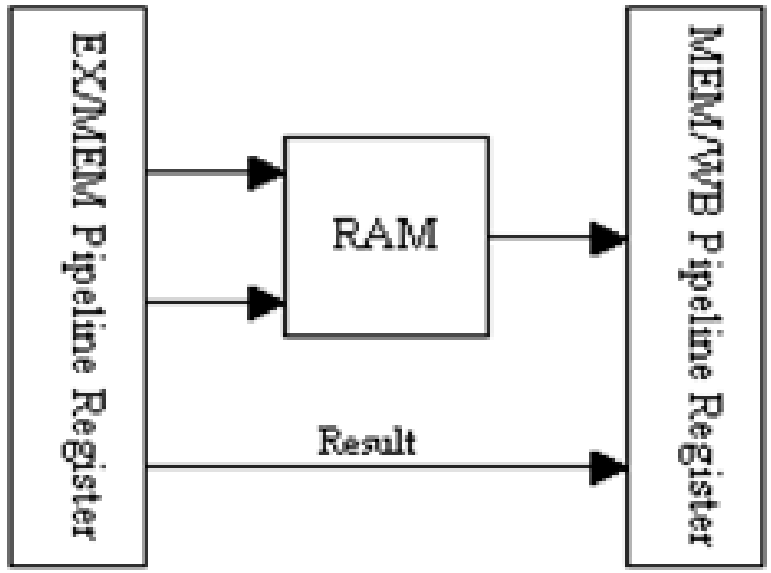


图 8-15 Stage 4 的基本结构

8.4 流水线各段的功能描述与设计

8.4.4 stage4访存段(MEM)

1.功能描述

LW R1, R2, data6　或　SW R1, R2, data6。

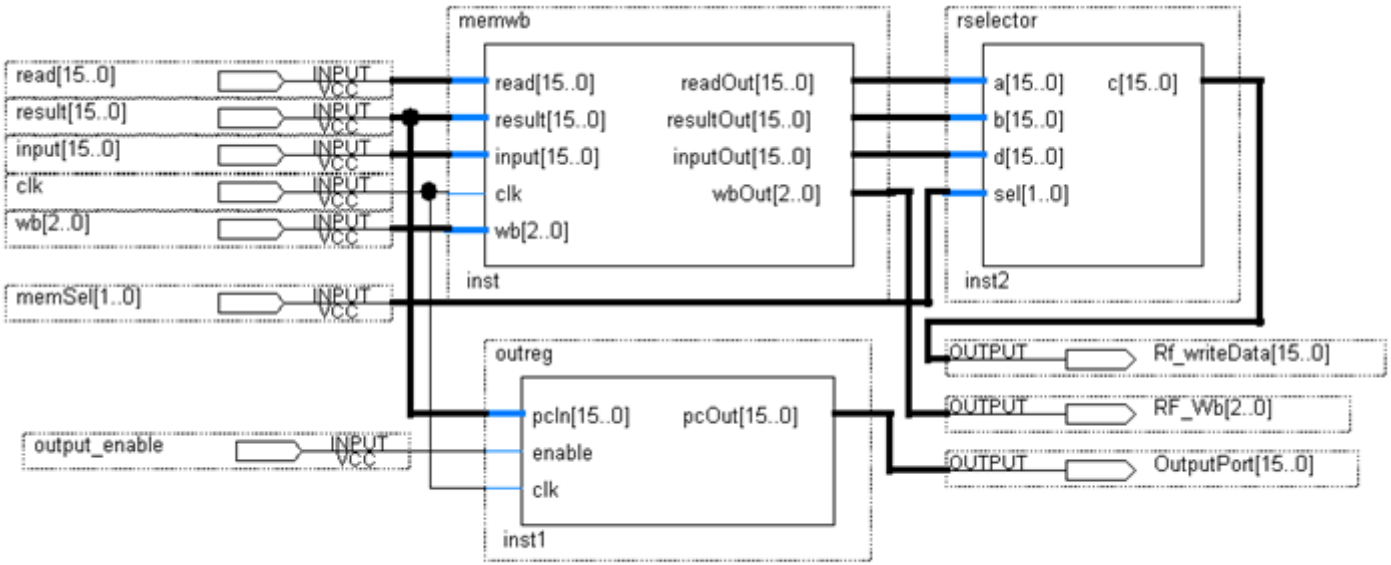


图 8-16　访存段 Stage 4 的内部结构

8.4 流水线各段的功能描述与设计

8.4.4 stage4访存段(MEM)

2. 模块划分和实现

（1）MEM/WB流水线寄存器。

【例 8-12】

```
module memwb (input[15:0] read, result, input_xhdl0, input clk,
input[2:0] wb, output[15:0] readOut, resultOut, inputOut,
output[2:0] wbOut);   reg[50:0] regValue;
   always @(clk)
     if (clk == 1'b1)  regValue[50:0] <= {result[15:0], read[15:0],
                                          input_xhdl0[15:0], wb[2:0]};
    assign resultOut =regValue[50:35]; assign  readOut=regValue[34:19];
    assign inputOut=regValue[18:3];   assign  wbOut=regValue[2:0];
endmodule
```

8.4 流水线各段的功能描述与设计

8.4.4 stage4访存段(MEM)

2. 模块划分和实现

（3）多路数据选择器：

【例 8-14】

```
module rselector (input[15:0] a,b,d,input[1:0] sel,output reg[15:0] c);
  always @(a or b or sel)
      case (sel)
          2'b00 : c<=a;    2'b01 : c<=b;
        default : c<=d;    endcase
endmodule
```

8.4 流水线各段的功能描述与设计

8.4.4 stage4访存段(MEM)

2. 模块划分和实现

（2）输出端口寄存器。

【例 8-13】

```
module  outreg(input[15:0] pcIn, input enable, clk, output reg[15:0] pcOut);
  reg[15:0] regValue;
   always @(posedge clk) begin
      if (enable==1'b1)  regValue=pcIn;  pcOut=regValue;  end
endmodule
```

8.4 流水线各段的功能描述与设计

8.4.5 stage5回写段(WB)

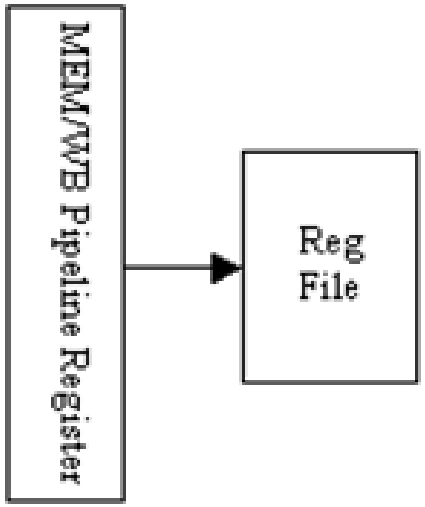


图 8-17 Stage 5 的基本结构

8.4 流水线各段的功能描述与设计

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。

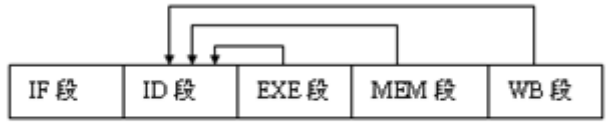


图 8-18 数据相关性检测原理图

② 数据相关的发生。

图 8-19 数据相关实例

8.4 流水线各段的功能描述与设计

1. 数据相关的检测及处理

③ 数据相关的处理。

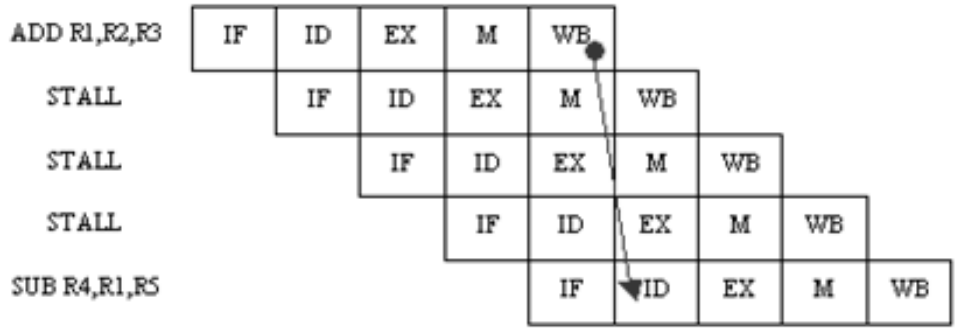


图 8-20 阻塞流水线

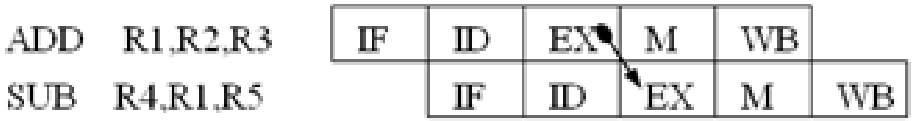


图 8-21 数据前推控制

③ 数据相关的处理。

（1）数据相关检测控制模块。

【例 8-15】

```
module  hazard (opcode, func, readone, readtwo, idexwrite, idexMemWE, idexwe,
   exmemwrite, memstage, pcenable, ifidenable, idexflush);
 input[3:0] opcode;input[2:0] func,readone,readtwo,idexwrite,exmemwrite;
 input idexMemWE, idexwe, memstage; output pcenable, ifidenable, idexflush;
  reg pcenable, ifidenable, idexflush;
     always @(opcode,func,readone,readtwo,idexwrite,idexMemWE,
      memstage,exmemwrite)   begin
       if (idexMemWE == 1) begin
          if (idexwrite==readone | idexwrite==readtwo)
          begin pcenable<=0; ifidenable<=0; idexflush<=1; end
          else  begin  pcenable<=1; ifidenable<=1; idexflush<=0; end
            end  else     begin
          if ((opcode==2 & func==2)|(opcode==2 & func==3)|opcode==5)
             begin  if (idexwe == 1)      begin
                if (idexwrite==readone | idexwrite==readtwo)
                begin pcenable<=0; ifidenable<=0; idexflush<=1; end
                else  begin pcenable<=1; ifidenable<=1; idexflush<=0; end
             end  else if (memstage==1) begin
                if (exmemwrite == readone | exmemwrite == readtwo)
                begin  pcenable<=0; ifidenable<=0; idexflush<=1; end
                else  begin  pcenable<=1; ifidenable<=1; idexflush<=0; end
             end   else
             begin  pcenable<=1; ifidenable<=1; idexflush<=0; end
          end     else
          begin pcenable<=1; ifidenable<=1; idexflush<=0;  end
       end
     end
  endmodule
```

③ 数据相关的处理。

（2）数据前推控制模块。

【例 8-16】

```
module forward(exmRegWrite, memwbRegWrite, exmWriteReg, memwbWriteReg,
        idexReadOne, idexReadTwo, aluSelA, aluSelB);
  input exmRegWrite, memwbRegWrite;
  input[2:0] exmWriteReg, memwbWriteReg, idexReadOne, idexReadTwo;
  output[1:0] aluSelA, aluSelB;     reg[1:0] aluSelA, aluSelB;
  always @(exmRegWrite, exmWriteReg, idexReadOne, idexReadTwo)  begin
         if (exmRegWrite == 1'b1)   begin
            if (memwbRegWrite == 1'b0)  begin//若写寄存器=读寄存器one,则写入
       if (exmWriteReg==idexReadOne) aluSelA<=2'b01; else aluSelA<=2'b00;
       if (exmWriteReg==idexReadTwo) aluSelB<=2'b01; else aluSelB<=2'b00;
              end        else
        begin if (exmWriteReg == idexReadOne) aluSelA <= 2'b01 ; else
        begin if (memwbWriteReg==idexReadOne) aluSelA<=2'b10;
                  else aluSelA<=2'b00; end
             if (exmWriteReg == idexReadTwo) aluSelB <= 2'b01 ;
     else  begin if (memwbWriteReg==idexReadTwo) aluSelB<=2'b10;
                 else aluSelB<=2'b00; end
           end     end
   else begin if (memwbRegWrite == 1'b1)  begin
       if (memwbWriteReg==idexReadTwo) aluSelB<=2'b10; else aluSelB<=2'b00;
       if (memwbWriteReg==idexReadOne) aluSelA<=2'b10; else aluSelA<=2'b00;
        end  else  begin aluSelA<=2'b00; aluSelB<=2'b00;  end
            end    end
    endmodule
```

8.4 流水线各段的功能描述与设计

1. 数据相关的检测及处理

③ 数据相关的处理。

（2）数据前推控制模块。

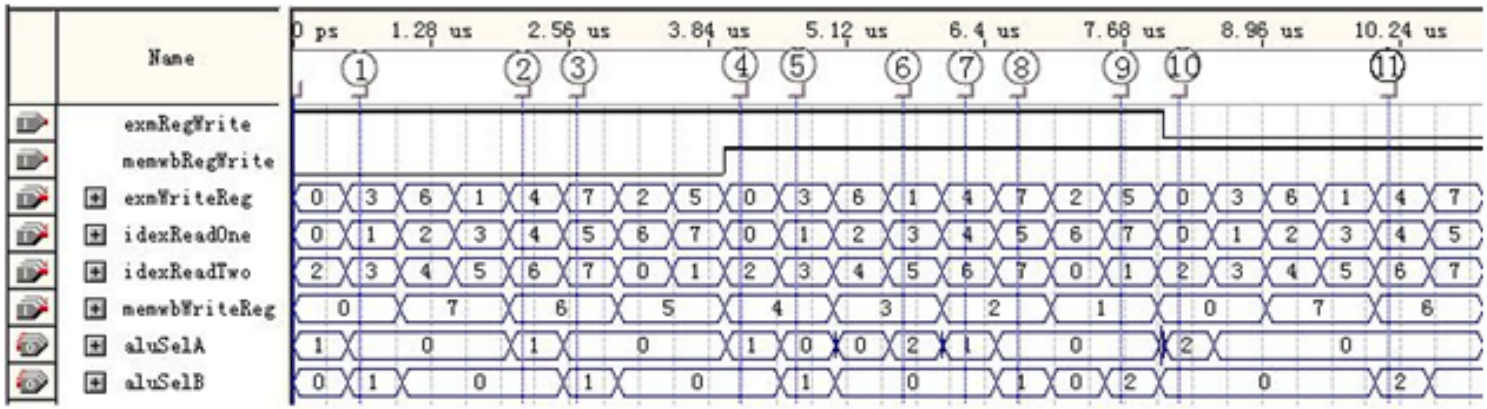


图 8-22 forward 模块的时序仿真图

8.4 流水线各段的功能描述与设计

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 流水线各段的功能描述与设计

8.4.7 控制单元

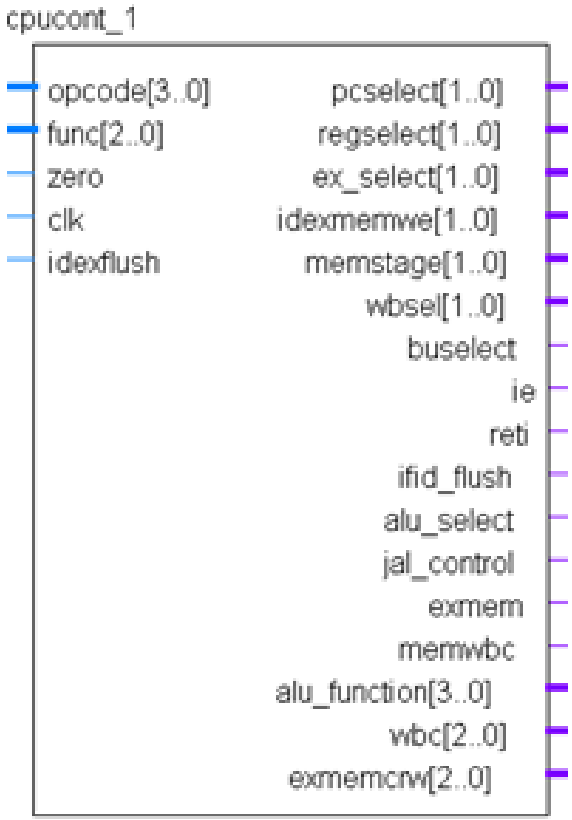
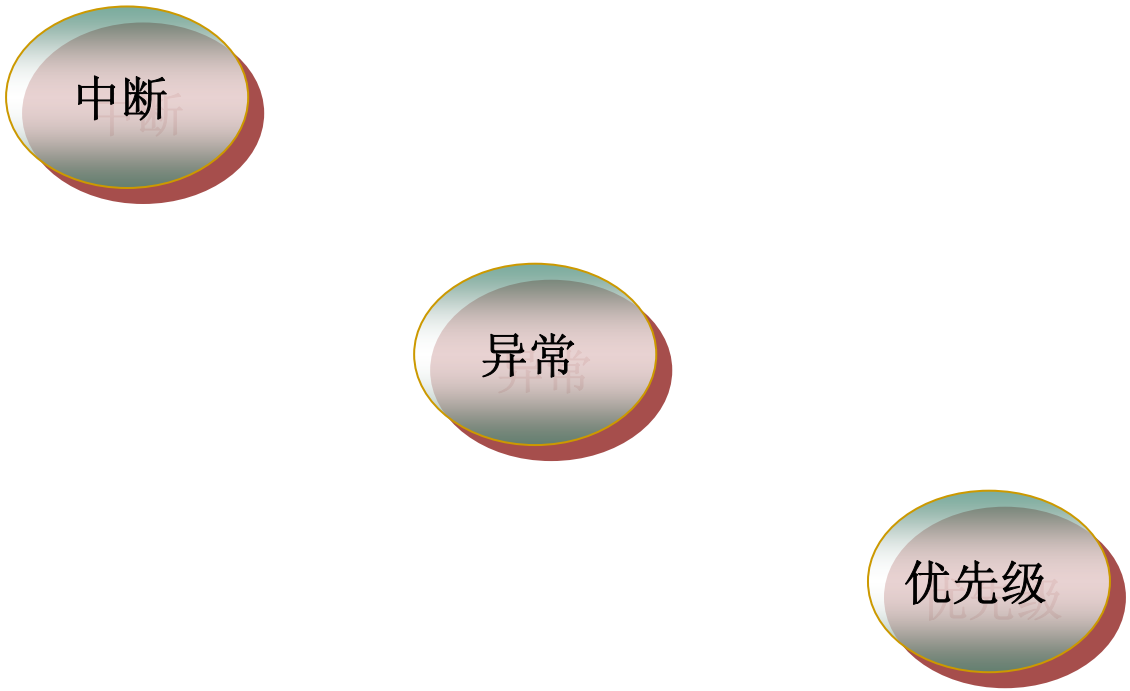


图 8-23 控制单元的结构图

8.4 流水线各段的功能描述与设计

8.4.8 中断与异常



8.4 流水线各段的功能描述与设计

8.4.9 流水线CPU系统构建与测试

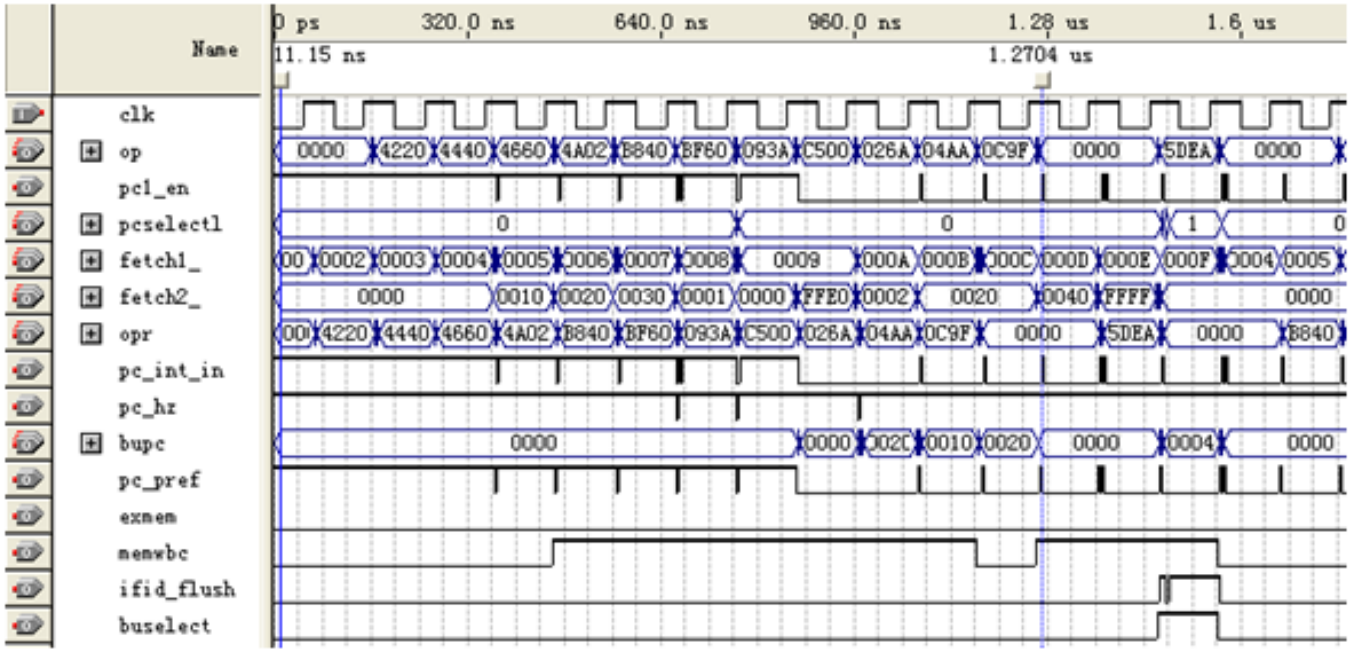


图 8-24 流水线 CPU 执行程序仿真波形

8.4 流水线各段的功能描述与设计

8.4.9 流水线CPU系统构建与测试

表 8-2 流水线 CPU 测试程序示例

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		

实验与设计

8-1. Stage1取指令段设计实验

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 8-25　lpm_rom 文件中的数据

实验与设计

8-2. Stage2指令译码段设计实验

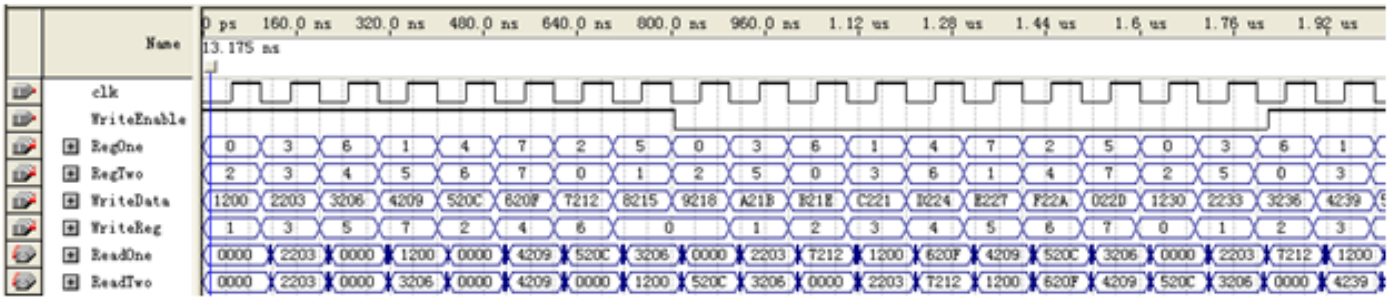


图 8-26　regFile 的时序仿真波形

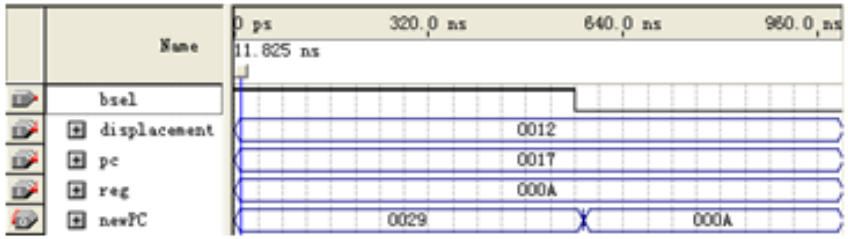


图 8-27　branch.vhd 的仿真波形

```
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);
        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

8-2. Stage2指令译码段设计实验

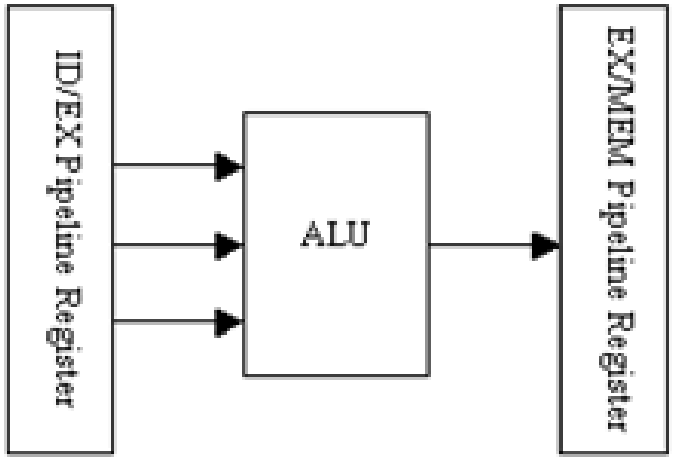


图 8-28 stage 3 执行 EXE 段的基本结构

实验与设计

8-3. Stage3指令译码段设计实验

表 8-3　alu_16 的运算/逻辑功能

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

8-3. Stage3指令译码段设计实验

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

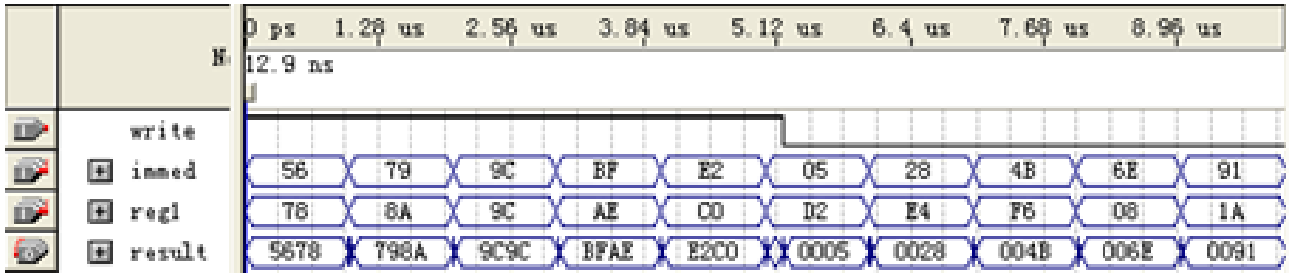


图 8-29 数据输入模块 movibox 的仿真波形

实验与设计

8-4. Stage4/5存储与写回段设计实验

8-5. 数据相关性控制实验

8-6. 数据通路设计实验

实验与设计

8-7. 流水线CPU综合设计

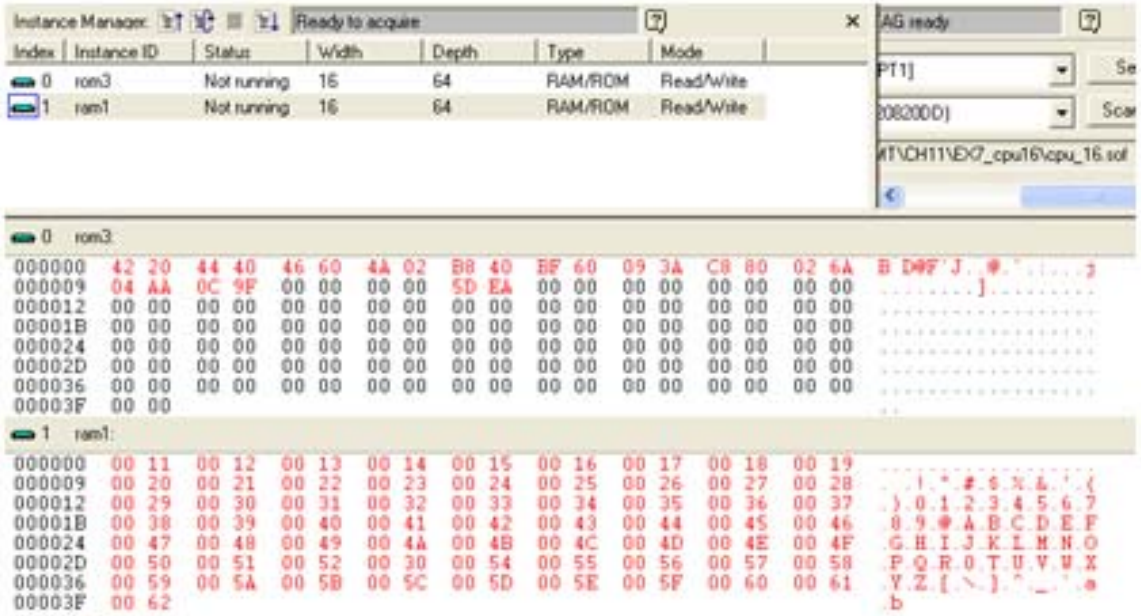


图 8-30 程序运行前 ROM 和 RAM 中的数据

实验与设计

8-7. 流水线CPU综合设计

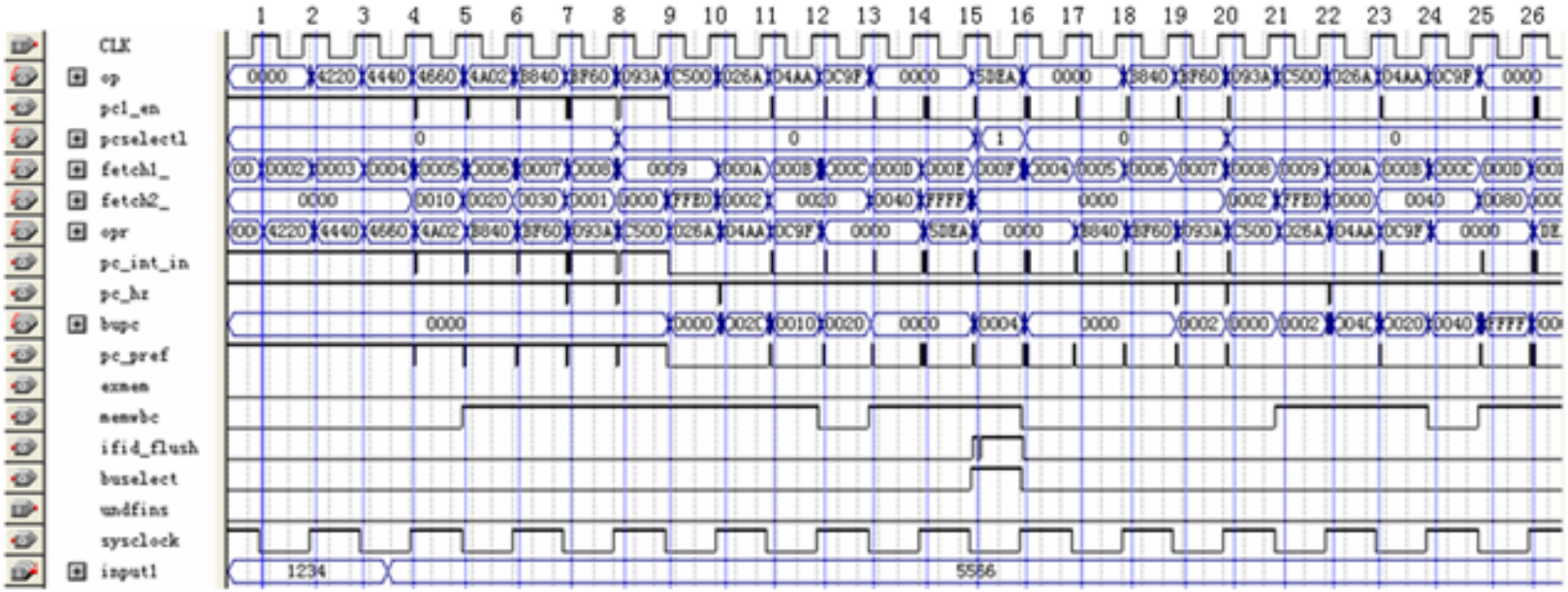


图 8-31 流水线 CPU 的时序仿真波形

实验与设计

8-7. 流水线CPU综合设计

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　R3,25H	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		