Lab 05 单周期 CPU

PB16051448 赵敏帆

实验目的

- 编写 MIPS 指令集单周期 CPU
- 实现指令集中的 6 条指令（addi、add、lw、sw、bgtz、j）
- 使用编写的 CPU 完成斐波那契数列计算的程序

实验平台

Vivado 2016.4

实验过程及结果

CPU 代码编写

ram 构建

使用 Vivado 自带 IP 核，构建两个 single port Distributed Memory，一个作为 Instruction Memory，一个作为 Data Memory。

Regfile 模块

```
module regfile(
input clk,
input rst_n,
input INIT,
input [4:0]rAddr1,
input [4:0]rAddr2,
input [4:0]wAddr,
input [31:0]wDin,
output [31:0]rDout1,
output [31:0]rDout2,
input wEna
    );
    reg [31:0]regfiles[0:31];
    always@(posedge clk)
    if(!INIT)begin
        regfiles[0]<=0;
        regfiles[1]<=0;
        regfiles[2]<=0;
        regfiles[3]<=0;
        regfiles[4]<=0;
        regfiles[5]<=0;
        regfiles[6]<=0;
        regfiles[7]<=0;
        regfiles[8]<=0;
        regfiles[9]<=0;
        regfiles[10]<=0;
        regfiles[11]<=0;
        regfiles[12]<=0;
        regfiles[13]<=0;
        regfiles[14]<=0;
        regfiles[15]<=0;
        regfiles[16]<=0;
        regfiles[17]<=0;
        regfiles[18]<=0;
        regfiles[19]<=0;
        regfiles[20]<=0;
        regfiles[21]<=0;
        regfiles[22]<=0;
        regfiles[23]<=0;
        regfiles[24]<=0;
        regfiles[25]<=0;
        regfiles[26]<=0;
        regfiles[27]<=0;
        regfiles[28]<=0;
        regfiles[29]<=0;
        regfiles[30]<=0;
        regfiles[31]<=0;
        end
    else if(wEna)
        regfiles[wAddr]<=wDin;
    assign rDout1=regfiles[rAddr1];
    assign rDout2=regfiles[rAddr2];
endmodule
```

ALU 模块

```
module ALU(
    input signed  [31:0] alu_a,
    input signed  [31:0] alu_b,
    input  [2:0] alu_op,
    output reg [31:0] alu_out,
    output reg Zero
    );
        parameter     A_OR= 3'h01;
        parameter    A_ADD= 3'h02;
        parameter    A_SUB= 3'h06;
        parameter    A_AND= 3'h00;
        parameter    A_SMALL = 3'h07;
        parameter    A_POSI= 3'h3;
          always@(*)
          begin
           case(alu_op)
            A_ADD: alu_out = alu_a+alu_b;
            A_SUB: alu_out = alu_a-alu_b;

            A_AND: alu_out = alu_a &alu_b;
            A_OR:  alu_out = alu_a|alu_b;
            A_SMALL:alu_out = alu_a<alu_b?1:0;
            A_POSI: alu_out= alu_a>0?0:1;
            default: alu_out=0;
            endcase
          if (alu_out==0)
              Zero=1;
          else
              Zero=0;
          end

endmodule
```

PC 模块

```
module PCins(
input clk,
input [31:0] PCdata,
input JUMP,
input [31:0]IR,
output reg INIT,
output reg [31:0] PC

    );
    reg [31:0]data;
    always@(posedge clk)
    if(JUMP)
    begin
      PC=PC+1;
      PC={PC[31:28]+IR[25:0]+2'b00};
      INIT<=1;
    end
    else if(!JUMP)
    begin
        PC<=PCdata;
        INIT<=1;
    end
    else
    begin
         PC<=0;
         INIT<=0;
    end
endmodule
```

Control 模块

```
module controlunit(
input [5:0]OP,
input [5:0]func,
output reg RegWrite,

output reg RegDist,
output reg ALUSrc,
output reg [2:0]ALUControl,
output reg Branch,
output reg MemWrite,
output reg MemtoReg,
output reg jump
    );
     parameter    LOAD = 6'h23;
     parameter    STORE= 6'h2b;
     parameter    ADD_SUB = 6'h0;
     parameter    ORI = 6'hd;
     parameter    BGTZ = 6'h7;
     parameter    JUMP = 6'h2;
     parameter    ADDI = 6'h8;
    always@(*)
    case(OP)
    LOAD:begin
            RegWrite<=1;
            RegDist<=0;
            ALUSrc<=1;
            Branch<=0;
            MemWrite<=0;
            MemtoReg<=1;
            jump<=0;
            ALUControl<=3'b010;
            end
    STORE:begin
                RegWrite<=0;
                RegDist<=0;
                ALUSrc<=1;
                Branch<=0;
                MemWrite<=1;
                MemtoReg<=1;
                jump<=0;
                ALUControl<=3'b010;
    end
    ADD_SUB:begin
                RegWrite<=1;
                RegDist<=1;
                ALUSrc<=0;
                Branch<=0;
                MemWrite<=0;
                MemtoReg<=0;
                jump<=0;
                case(func[3:0])
                4'b0000:ALUControl<=3'b010;
                4'b0010:ALUControl<=3'b110;
                4'b0100:ALUControl<=3'b000;
                4'b0101:ALUControl<=3'b001;
                4'b1010:ALUControl<=3'b111;
                endcase
    end

    ORI:begin
```

```
                RegWrite<=1;
                RegDist<=0;
                ALUSrc<=1;
                Branch<=0;
                MemWrite<=0;
                MemtoReg<=0;
                jump<=0;
                ALUControl<=3'b001;
    end
    BGTZ:begin
                RegWrite<=0;
                RegDist<=1;
                ALUSrc<=0;
                Branch<=1;
                MemWrite<=0;
                MemtoReg<=0;
                jump<=0;
                ALUControl<=3'b011;//自定义了一个与0比较的alu命令
    end
    JUMP:begin
                RegWrite<=0;
                RegDist<=1;
                ALUSrc<=0;
                Branch<=1;
                MemWrite<=0;
                MemtoReg<=0;
                jump<=1;
    end
    ADDI:begin
                    RegWrite<=1;
                    RegDist<=0;
                    ALUSrc<=1;
                    Branch<=0;
                    MemWrite<=0;
                    MemtoReg<=0;
                    jump<=0;
                    ALUControl<=3'b010;
    end
    default:begin
                RegWrite<=0;
                RegDist<=0;
                ALUSrc<=1;
                Branch<=0;
                MemWrite<=0;
                MemtoReg<=0;
                jump<=0;
                ALUControl<=3'b010;
                //INIT<=0;
    end
    endcase

endmodule
```

TOP 模块

```
module CPU(
input clk
    );
    wire [31:0]IR;
    wire [31:0]PCdata;
    wire [31:0]PC;
    wire [31:0]NPC;
    wire PCSrc;
    wire MemtoReg;
    wire MemWrite;
    wire Branch;
    wire [2:0]ALUControl;
    wire ALUSrc;
    wire RegDist;
    wire RegWrite;
    wire [4:0]WriteReg;
    wire [31:0]SrcB;
    wire [31:0]SrcA;
    wire [31:0]Result;
    wire [31:0]PCBranch;
    wire [31:0]SignImm;
    wire [31:0]RD2;
    wire Zero;
    wire [31:0]ALUResult;
    wire [31:0]ReadData;
    wire jump;
    wire INIT;
    assign PCSrc=Branch & Zero;
    assign WriteReg=RegDist>0?IR[15:11]:IR[20:16];
    assign SrcB=ALUSrc>0?SignImm:RD2;
    assign Result=MemtoReg>0?ReadData:ALUResult;
    assign PCdata=PCSrc>0?PCBranch:NPC;
    PCins CPU_PCins(
    .clk (clk),
    .PCdata (PCdata),
    .PC (PC),
    .IR (IR),
    .JUMP (jump),
    .INIT (INIT)
    );
    dist_mem_gen_0 Instruction_Memory(
    .a (PC[7:0]),
    .d (),
    .clk (clk),
    .we (0),
    .spo (IR)
    );
    regfile CPU_RegFile(

    .clk (clk),
    .rst_n (),
    .rAddr1 (IR[25:21]),
    .rAddr2 (IR[20:16]),
    .wAddr (WriteReg),
    .wDin (Result),
    .rDout1 (SrcA),
    .rDout2 (RD2),
    .wEna (RegWrite),
    .INIT (INIT)
    );
    ALU CPU_ALU(
    .alu_a (SrcA),
    .alu_b (SrcB),
    .alu_op (ALUControl),
    .alu_out (ALUResult),
    .Zero (Zero)
    );
    dist_mem_gen_1 Data_Memory(
   .a (ALUResult[15:0]),
   .d (RD2),
   .clk (clk),
   .we (MemWrite),
   .spo (ReadData)
    );
    PCplus CPU_PCplus(
    .PC (PC),
    .NPC (NPC)
    );
    PCBranch CPU_PCBranch(
    .SignImm (SignImm),
    .NPC (NPC),
    .ALUControl (ALUControl),
    .PCBranch (PCBranch)
    );
    SignExtend CPU_SignExtend(
    .data (IR[15:0]),
    .SignImm (SignImm)
    );
    controlunit CPU_ControlUnit(
    .OP (IR[31:26]),
    .func (IR[5:0]),
    .RegWrite (RegWrite),
    .RegDist (RegDist),
    .ALUSrc (ALUSrc),
    .ALUControl (ALUControl),
    .Branch (Branch),
    .MemWrite (MemWrite),
    .MemtoReg (MemtoReg),
    .jump (jump)
    );
endmodule
```

其他模块

计算 PC 增量

```
module PCplus(
input [31:0]PC,
output [31:0]NPC
    );
    assign NPC=PC+1;
endmodule
```

带符号扩展

```
module SignExtend(
input signed [15:0]data,
output signed [31:0]SignImm
    );
    assign SignImm=data;
endmodule
```

PCBranch

```
module PCBranch(
input signed [31:0] SignImm,
input signed [31:0] NPC,
input [2:0]ALUControl,
output reg signed [31:0] PCBranch
    );
    always @(*)
    if (ALUControl==3'b011)//bgtz
        PCBranch<=SignImm+NPC;
    else
        PCBranch<=(SignImm<<2)+NPC;
endmodule
```

.coe 文件

利用 MARS 导出 .coe 文件并做相应修改

data.coe

```
MEMORY_INITIALIZATION_RADIX=10;
MEMORY_INITIALIZATION_VECTOR=
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,20,3,3;
```

IR.coe

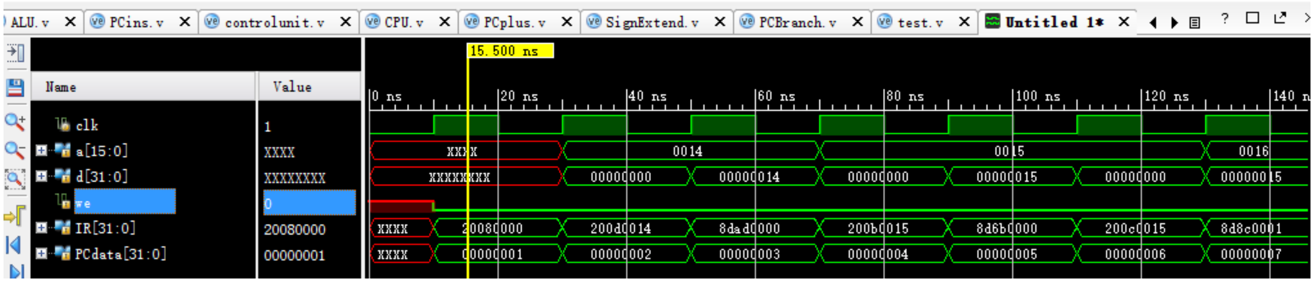
```
MEMORY_INITIALIZATION_RADIX=16;
MEMORY_INITIALIZATION_VECTOR=
20080000,
200d0014,
8dad0000,
200b0015,
8d6b0000,
200c0015,
8d8c0001,
ad0b0000,
ad0c0001,
21a9fffe,
8d0b0000,
8d0c0001,
016c5020,
ad0a0002,
21080001,
2129ffff,
1d20fff9,
08000011;
```

仿真文件

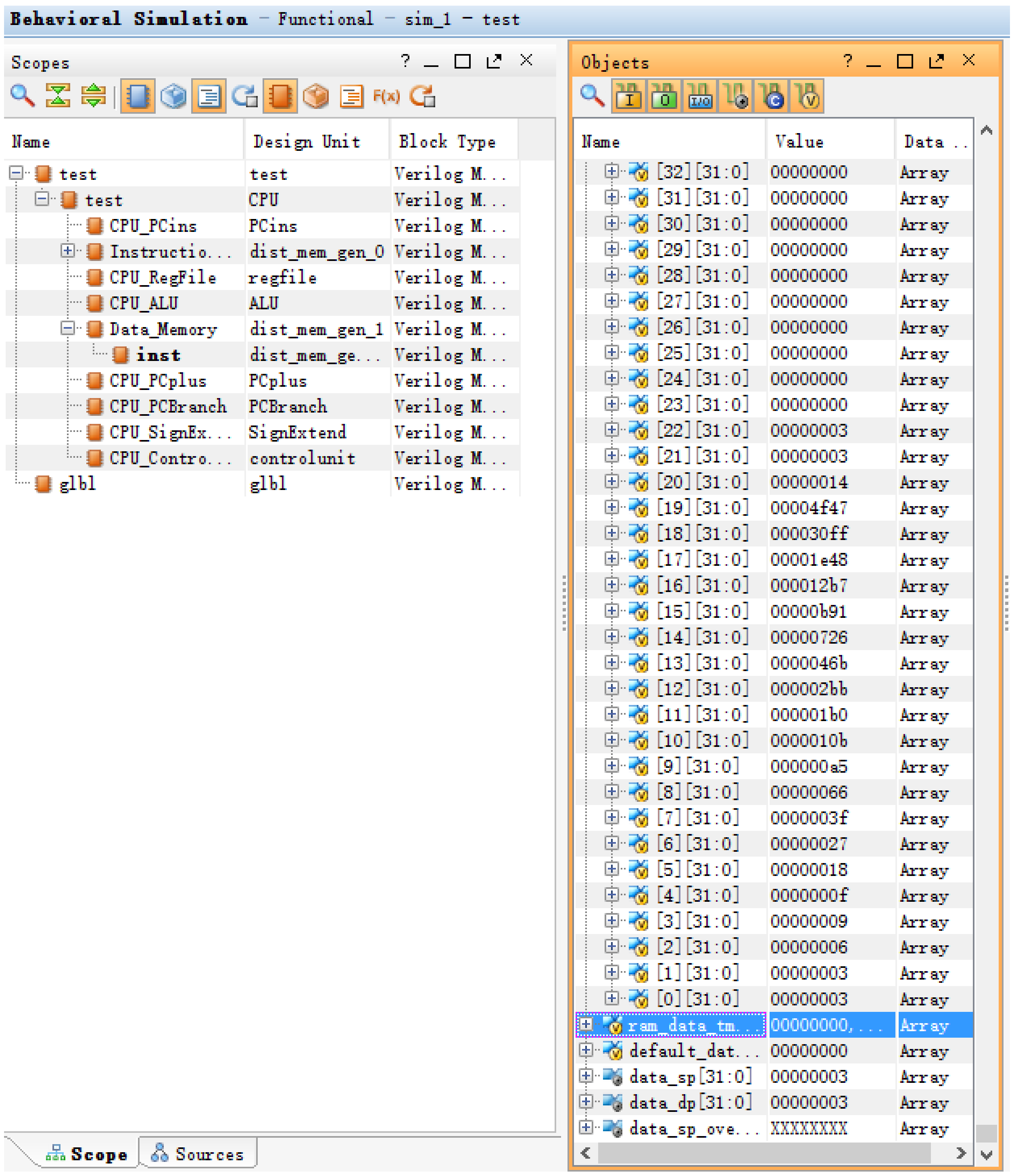
```
module test(

    );
    reg clk;
    CPU test(.clk (clk));
    always #10 clk=~clk;
    initial begin
    clk=0;
    end
endmodule
```

仿真结果

部分周期情况



成功计算斐波那契数列并写入到 Data Memory 中。



Name	Design Unit	Block Type
test	test	Verilog M...
test	CPU	Verilog M...
CPU_PCins	PCins	Verilog M...
Instructio...	dist_mem_gen_0	Verilog M...
CPU_RegFile	regfile	Verilog M...
CPU_ALU	ALU	Verilog M...
Data_Memory	dist_mem_gen_1	Verilog M...
inst	dist_mem_ge...	Verilog M...
CPU_PCplus	PCplus	Verilog M...
CPU_PCBranch	PCBranch	Verilog M...
CPU_SignEx...	SignExtend	Verilog M...
CPU_Contro...	controlunit	Verilog M...
glbl	glbl	Verilog M...

Name	Value	Data ..
[32][31:0]	00000000	Array
[31][31:0]	00000000	Array
[30][31:0]	00000000	Array
[29][31:0]	00000000	Array
[28][31:0]	00000000	Array
[27][31:0]	00000000	Array
[26][31:0]	00000000	Array
[25][31:0]	00000000	Array
[24][31:0]	00000000	Array
[23][31:0]	00000000	Array
[22][31:0]	00000003	Array
[21][31:0]	00000003	Array
[20][31:0]	00000014	Array
[19][31:0]	00004f47	Array
[18][31:0]	000030ff	Array
[17][31:0]	00001e48	Array
[16][31:0]	000012b7	Array
[15][31:0]	00000b91	Array
[14][31:0]	00000726	Array
[13][31:0]	0000046b	Array
[12][31:0]	000002bb	Array
[11][31:0]	000001b0	Array
[10][31:0]	0000010b	Array
[9][31:0]	000000a5	Array
[8][31:0]	00000066	Array
[7][31:0]	0000003f	Array
[6][31:0]	00000027	Array
[5][31:0]	00000018	Array
[4][31:0]	0000000f	Array
[3][31:0]	00000009	Array
[2][31:0]	00000006	Array
[1][31:0]	00000003	Array
[0][31:0]	00000003	Array
ram_data_tm...	00000000,...	Array
default_dat...	00000000	Array
data_sp[31:0]	00000003	Array
data_dp[31:0]	00000003	Array
data_sp_ove...	XXXXXXXX	Array

心得体会

- PC=PC+4 是对于字地址差值为 4 的情况设定的，若字地址差值为 1，则应在编写 CPU 时做相应改动。
- 异步读，同步写应使用 Distributed Memory。
- CPU 开始运行前要对寄存器的值初始化。