Lab 06 多周期 CPU

PB16051448 赵敏帆

实验目的

- 编写 MIPS 指令集多周期 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

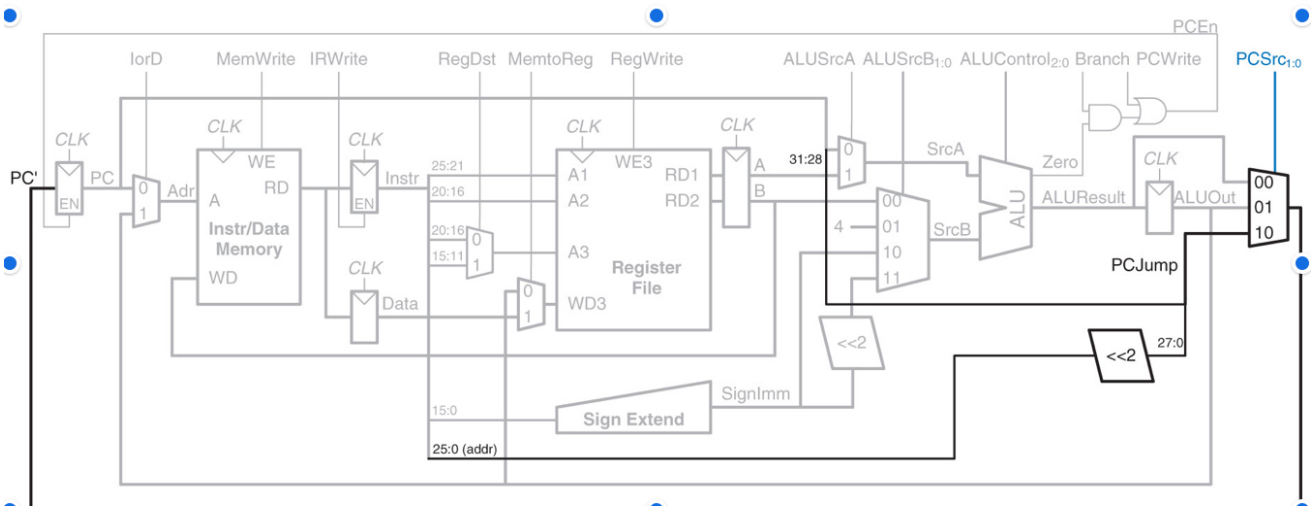
实验平台

Vivado 2016.4

实验过程及结果

CPU 代码编写

数据通路



PC 模块

```
module PCUnit(
input clk,
input PCEn,
input INIT,
input [31:0]PCnext,
output reg [31:0]PC
    );
    always@(posedge clk)

    if(INIT)
```

```
        PC<=23;
    else if(PCEn)
        PC<=PCnext;
endmodule
```

I/D Memory 模块

使用 Vivado 自带的 IP 核，构建一个 Single Port RAM （Block Memory）。

IR 模块

```
module Instr(
input IRWrite,
input clk,
input [31:0]RD,
output reg [31:0]IR
    );
    always@(posedge clk)
    if (IRWrite)
    IR<=RD;
endmodule
```

Regfile 模块

```
module regfile(
input clk,
input rst_n,
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,
input INIT
    );
    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
```

Control 模块

```
module ControlUnit(
input clk,
input [5:0]OP,
input [5:0]Funct,
output reg INIT,
output reg IorD,
output reg MemWrite,
output reg IRWrite,
output reg PCWrite,
output reg Branch,
output reg [1:0]PCSrc,
output reg [2:0]ALUControl,
output reg [1:0]ALUSrcB,
output reg ALUSrcA,
output reg RegWrite,
output reg RegDist,
output reg MemtoReg
    );
    reg [5:0]count;
    parameter Fetch1=6'h0;
    parameter Fetch2=6'h1;
    parameter Fetch3=6'h2;
    parameter Fetch4=6'h3;
    parameter Decode=6'h4;
    parameter MemAdr=6'h5;
    parameter MemRead1=6'h6;
    parameter MemRead2=6'h7;
    parameter MemRead3=6'h8;
    parameter MemWriteBack=6'h9;
    parameter WriteMem=6'ha;
    parameter Execute=6'hb;
    parameter ALUWriteBack=6'hc;
    parameter BRANCH1=6'hd;
    parameter BRANCH2=6'he;
    parameter ADDIExecute=6'hf;
    parameter ADDIWriteBack=6'h10;
    parameter JUMP=6'h11;


    parameter LW=6'h23;
```

```
parameter SW=6'h2b;
parameter ADD_SUB=6'h0;
parameter ORI=6'hd;
parameter BEQ=6'h7;
parameter ADDI=6'h8;
parameter J=6'h2;
always@(posedge clk)
case (count)
Fetch1:
begin
INIT<=0;
IorD<=0;
MemWrite<=0;
IRWrite<=1;
PCWrite<=0;
Branch<=0;
PCSrc<=0;
ALUControl<=3'b010;
ALUSrcB<=1;
ALUSrcA<=0;
RegWrite<=0;
RegDist<=0;
MemtoReg<=0;
count<=count+1;
end
Fetch2:
begin
    PCWrite<=0;//pc<=pc+1;
    count<=count+1;
end
Fetch3:
begin
    PCWrite<=0;
    count<=count+1;
end
Fetch4:
begin
    PCWrite<=1;
    count<=count+1;
end
Decode:
begin
    IorD<=0;
    MemWrite<=0;
    IRWrite<=0;
    PCWrite<=0;
    Branch<=0;
    PCSrc<=0;
    ALUControl<=3'b010;
    ALUSrcB<=3;
    ALUSrcA<=0;
    RegWrite<=0;

    RegDist<=0;
```

```
        MemtoReg<=0;
        case(OP)
        LW: count<=MemAdr;
        SW: count<=MemAdr;
        ADD_SUB: count<=Execute;
        ORI: count<=ADDIExecute;
        BEQ: count<=BRANCH1;
        ADDI: count<=ADDIExecute;
        J: count<=JUMP;
        endcase
    end
    MemAdr:
    begin
        IorD<=0;
        MemWrite<=0;
        IRWrite<=0;
        PCWrite<=0;
        Branch<=0;
        PCSrc<=0;
        ALUControl<=3'b010;
        ALUSrcB<=2;
        ALUSrcA<=1;
        RegWrite<=0;
        RegDist<=0;
        MemtoReg<=0;
        case(OP)
        LW:count<=MemRead1;
        SW:count<=WriteMem;
        endcase
    end
    MemRead1:
    begin
        IorD<=1;
        MemWrite<=0;
        IRWrite<=0;
        PCWrite<=0;
        Branch<=0;
        PCSrc<=0;
        ALUControl<=3'b010;
        ALUSrcB<=2;
        ALUSrcA<=1;
        RegWrite<=0;
        RegDist<=0;
        MemtoReg<=0;
        count<=MemRead2;
    end
    MemRead2: count<=MemRead3;
    MemRead3:
    begin
        count<=MemWriteBack;
    end
    MemWriteBack:

    begin
```

```
        IorD<=1;
        MemWrite<=0;
        IRWrite<=0;
        PCWrite<=0;
        Branch<=0;
        PCSrc<=0;
        ALUControl<=3'b010;
        ALUSrcB<=2;
        ALUSrcA<=1;
        RegWrite<=1;
        RegDist<=0;
        MemtoReg<=1;
        count<=Fetch1;
    end
    WriteMem:
    begin
        IorD<=1;
        MemWrite<=1;
        IRWrite<=0;
        PCWrite<=0;
        Branch<=0;
        PCSrc<=0;
        ALUControl<=3'b010;
        ALUSrcB<=2;
        ALUSrcA<=1;
        RegWrite<=0;
        RegDist<=0;
        MemtoReg<=0;
        count<=Fetch1;
    end
    Execute:
    begin
        IorD<=0;
        MemWrite<=0;
        IRWrite<=0;
        PCWrite<=0;
        Branch<=0;
        PCSrc<=0;
        ALUSrcB<=0;
        ALUSrcA<=1;
        RegWrite<=0;
        RegDist<=0;
        MemtoReg<=0;
        case(Funct)
        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
        count<=ALUWriteBack;
    end

    ALUWriteBack:
```

```
    begin
        IorD<=0;
        MemWrite<=0;
        IRWrite<=0;
        PCWrite<=0;
        Branch<=0;
        PCSrc<=0;
        ALUSrcB<=0;
        ALUSrcA<=1;
        RegWrite<=1;
        RegDist<=1;
        MemtoReg<=0;
        case(Funct)
        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
        count<=Fetch1;
    end
    BRANCH1:
    begin
        IorD<=0;
        MemWrite<=0;
        IRWrite<=0;
        PCWrite<=0;
        Branch<=0;
        PCSrc<=1;
        ALUControl<=3'b010;
        ALUSrcB<=2;
        ALUSrcA<=0;
        RegWrite<=0;
        RegDist<=0;
        MemtoReg<=0;
        count<=BRANCH2;
    end
    BRANCH2:
    begin
        IorD<=0;
        MemWrite<=0;
        IRWrite<=0;
        PCWrite<=0;
        Branch<=1;
        PCSrc<=1;
        ALUControl<=3'b011;
        ALUSrcB<=0;
        ALUSrcA<=1;
        RegWrite<=0;
        RegDist<=0;
        MemtoReg<=0;
        count<=Fetch1;

    end
```

```
ADDIExecute:
begin
    IorD<=0;
    MemWrite<=0;
    IRWrite<=0;
    PCWrite<=0;
    Branch<=0;
    PCSrc<=0;
    case (OP)
    ADDI:ALUControl<=3'b010;
    ORI:ALUControl<=3'b001;
    endcase
    ALUSrcB<=2;
    ALUSrcA<=1;
    RegWrite<=0;
    RegDist<=0;
    MemtoReg<=0;
    count<=ADDIWriteBack;
end
ADDIWriteBack:
begin
    IorD<=0;
    MemWrite<=0;
    IRWrite<=0;
    PCWrite<=0;
    Branch<=0;
    PCSrc<=0;
    ALUControl<=3'b010;
    ALUSrcB<=2;
    ALUSrcA<=1;
    RegWrite<=1;
    RegDist<=0;
    MemtoReg<=0;
    count<=Fetch1;
end
JUMP:
begin
    IorD<=0;
    MemWrite<=0;
    IRWrite<=0;
    PCWrite<=1;
    Branch<=0;
    PCSrc<=2;
    ALUControl<=3'b010;
    ALUSrcB<=3;
    ALUSrcA<=0;
    RegWrite<=0;
    RegDist<=0;
    MemtoReg<=0;
    count<=Fetch1;
end
default:

begin
```

```
        INIT<=1;
        count<=Fetch1;
    end
    endcase


endmodule
```

TOP 模块

```
module Top(
input clk
    );
    wire [31:0]PCnext;
    wire [31:0]PC;
    wire [31:0]Adr;
    wire [31:0]ALUOUT;
    wire [31:0]SrcB_RD;
    wire [31:0]RD;
    wire [31:0]Data;
    wire [31:0]IR;
    wire [4:0]A3;
    wire [31:0]WD3;
    wire [31:0]RD1;
    wire [31:0]RD2;
    wire [31:0]SrcA_RD;
    wire [31:0]SrcB;
    wire [31:0]SrcA;
    wire [31:0]ALUResult;
    wire Zero;
    wire IorD;
    wire PCEn;
    wire MemWrite;
    wire IRWrite;
    wire RegDist;
    wire MemtoReg;
    wire RegWrite;
    wire ALUSrcA;
    wire [1:0]ALUSrcB;
    wire [2:0]ALUControl;
    wire Branch;
    wire PCWrite;
    wire [1:0]PCSrc;
    wire INIT;
    wire [31:0]PCJump;
    wire [31:0]SignImm;
    assign Adr=IorD>0?ALUOUT:PC;
    assign A3=RegDist>0?IR[15:11]:IR[20:16];
    assign WD3=MemtoReg>0?Data:ALUOUT;
    assign SrcA=ALUSrcA>0?SrcA_RD:PC;
    assign PCJump={PC[31:28]+IR[25:0]+2'b00};

    assign PCEn=PCWrite|(Branch&Zero);
```

```
ALU CPU_ALU(
    .alu_a (SrcA),
    .alu_b (SrcB),
    .alu_op (ALUControl),
    .alu_out (ALUResult),
    .Zero (Zero)
);
SrcPC CPU_SrcPC(
    .PCSrc (PCSrc),
    .ALUResult (ALUResult),
    .ALUOUT (ALUOUT),
    .PCJump (PCJump),
    .PCnext (PCnext)
    );
ControlUnit CPU_ControlUnit(
    .clk (clk),
    .OP (IR[31:26]),
    .Funct (IR[5:0]),
    .IorD (IorD),
    .MemWrite (MemWrite),
    .IRWrite (IRWrite),
    .PCWrite (PCWrite),
    .Branch (Branch),
    .PCSrc (PCSrc),
    .ALUControl (ALUControl),
    .ALUSrcB (ALUSrcB),
    .ALUSrcA (ALUSrcA),
    .RegWrite (RegWrite),
    .RegDist (RegDist),
    .MemtoReg (MemtoReg),
    .INIT (INIT)
            );
Instr CPU_Instr(
    .IRWrite (IRWrite),
    .clk (clk),
    .RD (RD),
    .IR (IR)
    );
PCUnit CPU_PCUnit(
    .clk (clk),
    .PCEn (PCEn),
    .INIT (INIT),
    .PCnext (PCnext),
    .PC (PC)
    );
RegALUOUT CPU_RegALUOUT(
    .clk (clk),
    .ALUResult (ALUResult),
    .ALUOUT (ALUOUT)
    );
RegData CPU_RegData(

    .clk (clk),
```

```
        .RD (RD),
        .Data (Data)
        );
    RegRD CPU_RegRD(
        .RD1 (RD1),
        .RD2 (RD2),
        .clk (clk),
        .SrcA_RD (SrcA_RD),
        .SrcB_RD (SrcB_RD)
        );
    SignExtend CPU_SignExtend(
        .data (IR[15:0]),
        .SignImm (SignImm)
        );
    SrcBMUX CPU_SrcBMUX(
        .RD2_B (SrcB_RD),
        .SignImm (SignImm),
        .ALUSrcB (ALUSrcB),
        .SrcB (SrcB)
        );
    regfile CPU_regfile(
        .clk (clk),
        .rst_n (),
        .rAddr1 (IR[25:21]),
        .rAddr2 (IR[20:16]),
        .wAddr (A3),
        .wDin (WD3),
        .rDout1 (RD1),
        .rDout2 (RD2),
        .wEna (RegWrite),
        .INIT(INIT)
        );
    blk_mem_gen_0 cpu_IDMem(
        .clka (clk),
        .ena (1),
        .wea (MemWrite),
        .addra (Adr[15:0]),
        .dina (SrcB_RD),
        .douta (RD)
        );
endmodule
```

其他模块

PC Source

```
module SrcPC(
input [1:0]PCSrc,
input [31:0]ALUResult,
input [31:0]ALUOUT,
input [31:0]PCJump,
output reg [31:0]PCnext
```

```
    );
    always@(*)
    case(PCSrc)
    2'b00:  PCnext=ALUResult;
    2'b01:  PCnext=ALUOUT;
    2'b10:  PCnext=PCJump;
    endcase
endmodule
```

ALUOut

```
module RegALUOUT(
input clk,
input [31:0]ALUResult,
output reg [31:0]ALUOUT
    );
    always@(posedge clk)
        ALUOUT<=ALUResult;
endmodule
```

I/D Memory 读出结果寄存器

```
module RegData(
input clk,
input [31:0]RD,
output reg [31:0]Data
    );
    always @(posedge clk ) begin
        Data<=RD;
    end
endmodule
```

regfile 读出结果寄存器

```
module RegRD(
input [31:0]RD1,
input [31:0]RD2,
input clk,
output reg [31:0]SrcA_RD,
output reg [31:0]SrcB_RD
    );
    always @ (posedge clk)
    begin
        SrcA_RD<=RD1;
        SrcB_RD<=RD2;
    end
endmodule
```

符号扩展

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

ALUSrcB MUX

```
module SrcBMUX(
input [31:0]RD2_B,
input [31:0]SignImm,
input [1:0]ALUSrcB,
output reg [31:0]SrcB
    );
    always@(*)
    case (ALUSrcB)
    2'b00: SrcB<=RD2_B;
    2'b01: SrcB<=1;
    2'b10: SrcB<=SignImm;
    2'b11: SrcB<=SignImm<<2;
    endcase
endmodule
```

仿真代码

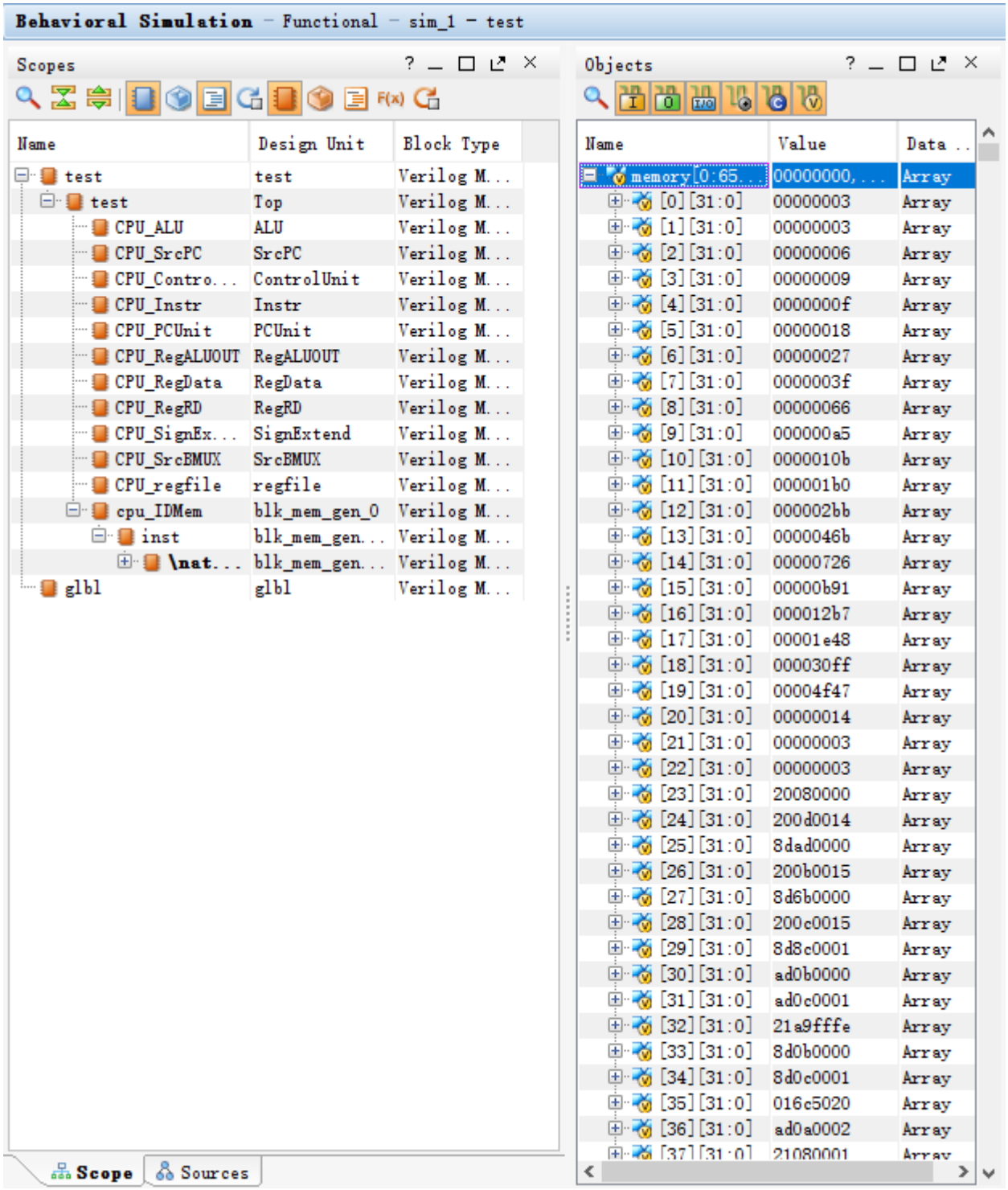
```
module test(

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

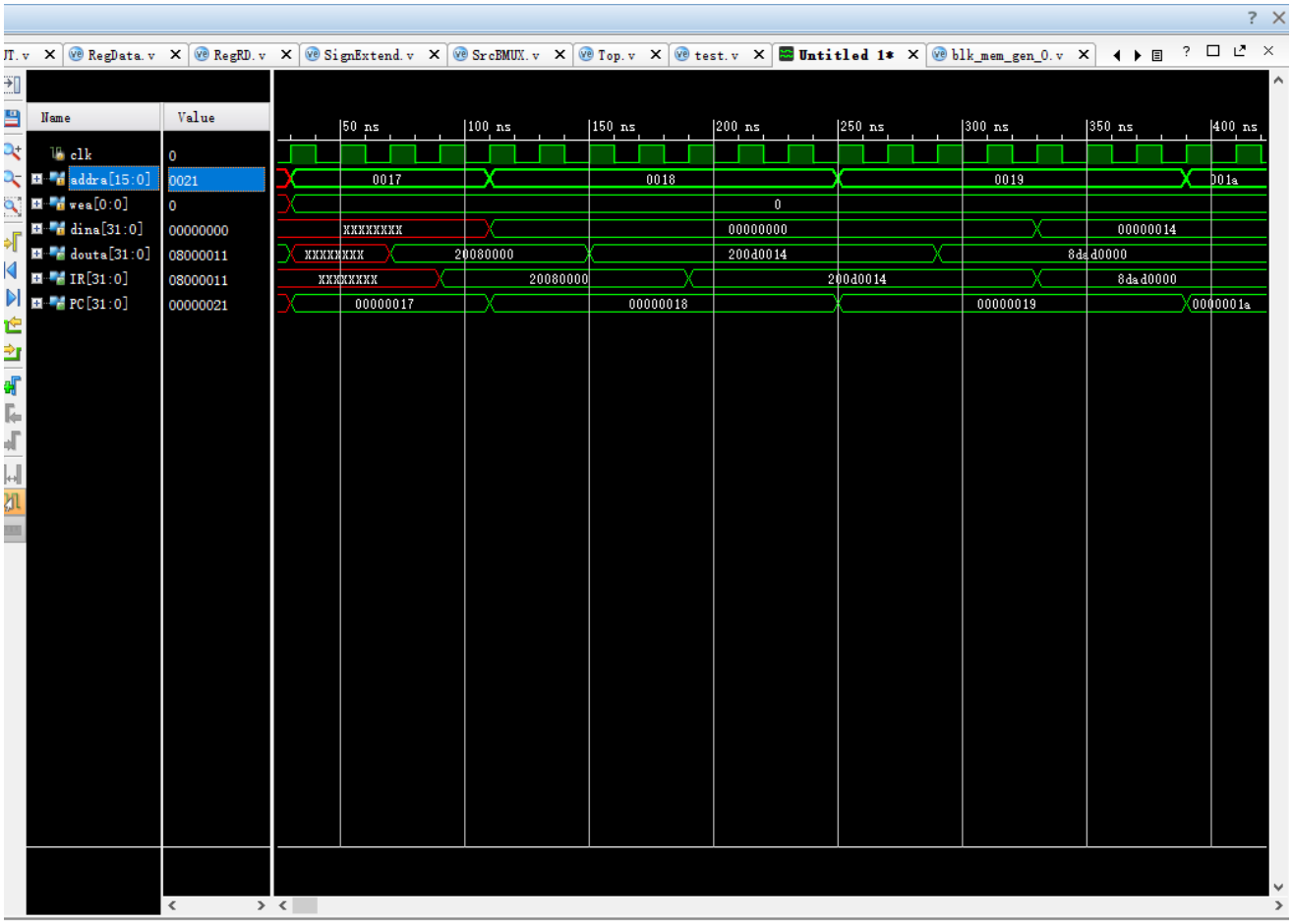
.coe 文件

```
MEMORY_INITIALIZATION_RADIX=16;
MEMORY_INITIALIZATION_VECTOR=
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,14,3,3,
20080000,
```

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

仿真结果


Behavioral Simulation - Functional - sim_1 - test
Scopes
Name
Design Unit
Block Type
test
test
Verilog M...
test
Top
Verilog M...
CPU_ALU
ALU
Verilog M...
CPU_SrcPC
SrcPC
Verilog M...
CPU_Contro...
ControlUnit
Verilog M...
CPU_Instr
Instr
Verilog M...
CPU_PCUnit
PCUnit
Verilog M...
CPU_RegALUOUT
RegALUOUT
Verilog M...
CPU_RegData
RegData
Verilog M...
CPU_RegRD
RegRD
Verilog M...
CPU_SignEx...
SignExtend
Verilog M...
CPU_SrcBMUX
SrcBMUX
Verilog M...
CPU_regfile
regfile
Verilog M...
cpu_IDMem
blk_mem_gen_0
Verilog M...
inst
blk_mem_gen...
Verilog M...
\nat...
blk_mem_gen...
Verilog M...
glbl
glbl
Verilog M...
Scope
Sources
Objects
Name
Value
Data ..
memory[0:65...
00000000,...
Array
[0][31:0]
00000003
Array
[1][31:0]
00000003
Array
[2][31:0]
00000006
Array
[3][31:0]
00000009
Array
[4][31:0]
0000000f
Array
[5][31:0]
00000018
Array
[6][31:0]
00000027
Array
[7][31:0]
0000003f
Array
[8][31:0]
00000066
Array
[9][31:0]
000000a5
Array
[10][31:0]
0000010b
Array
[11][31:0]
000001b0
Array
[12][31:0]
000002bb
Array
[13][31:0]
0000046b
Array
[14][31:0]
00000726
Array
[15][31:0]
00000b91
Array
[16][31:0]
000012b7
Array
[17][31:0]
00001e48
Array
[18][31:0]
000030ff
Array
[19][31:0]
00004f47
Array
[20][31:0]
00000014
Array
[21][31:0]
00000003
Array
[22][31:0]
00000003
Array
[23][31:0]
20080000
Array
[24][31:0]
200d0014
Array
[25][31:0]
8dad0000
Array
[26][31:0]
200b0015
Array
[27][31:0]
8d6b0000
Array
[28][31:0]
200c0015
Array
[29][31:0]
8d8c0001
Array
[30][31:0]
ad0b0000
Array
[31][31:0]
ad0c0001
Array
[32][31:0]
21a9fffe
Array
[33][31:0]
8d0b0000
Array
[34][31:0]
8d0c0001
Array
[35][31:0]
016c5020
Array
[36][31:0]
ad0a0002
Array
[37][31:0]
21080001
Array



心得体会

- 单周期改多周期要注意时序问题
- Block Memory 在读取时是同步读取，有两周期延迟