实验报告六
PB14000556 陈晓彤

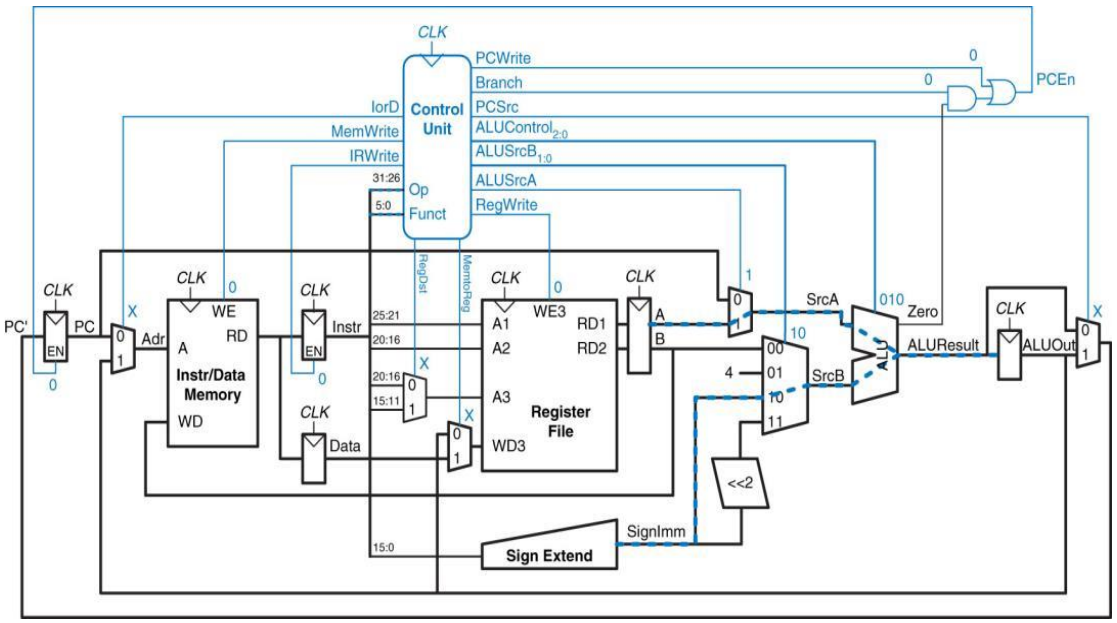
实验题目：设计一个多周期 CPU 并成功运行一个计算 Fibonacci 数列的程序
实验要求：

1. CPU 类型为 MIPS 体系结构
2. 能运行 add addi lw sw bgtz j 六条指令
3. 核心部件为 Memory RegisterFile ALU ControlUnit
4. 设计合理的状态机产生分时控制信号，保证系统在多周期执行完指令

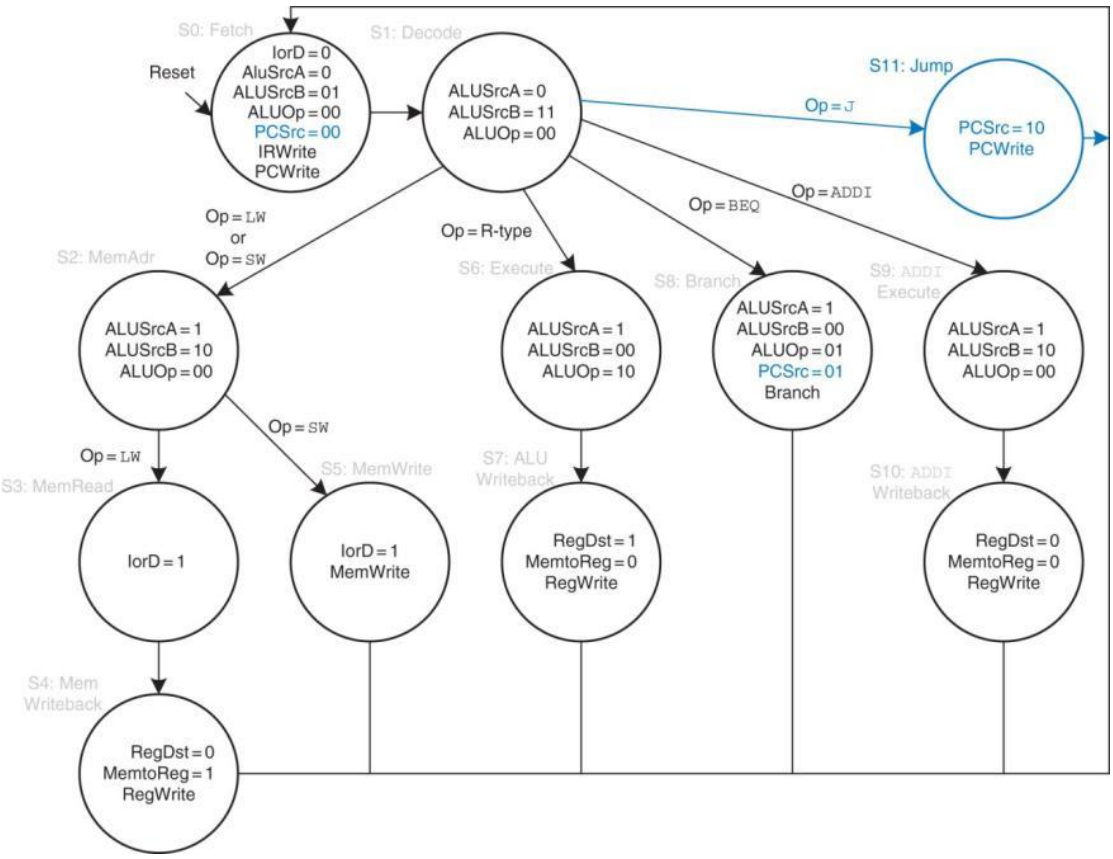
设计思路：

1. Memory 由 ipcore—blocked RAM 实现，同步读取，初始化文件 Initial.coe 其内容为两部分，指令段为 Mars 编译出的汇编指令代码段，注意到指令有 18 行，所以为了凑整从第 32 个地址开始存数据段，所以中间要加上 14 个 0，数据段前 20 个地址同样要填充 0 以存放后来算出的数列，所以总共要留出 34 个空位放 0
2. RegisterFile 由 32 个 32 位寄存器实现，采用异步读取，同步写入的方式，初始化时 0 号寄存器内容为 0
3. ControlUnit 发出两类信号：一类是部件使能信号，包括 PCWrite IRWrite MemWrite RegWrite Branch ;另一类为数据选择信号，包括 RegDst ALUSrcA ALUSrcB MemtoReg IorD PCSrc，作用时间见状态图，注意使能信号如下一状态不使用需要改为关闭状态以免发生数据错误，数据选择信号如下一周期不更改可不予赋值
4. ALU 只进行加法计算,数据来源由 ALUSrcA ALUSrcB 决定
5. PC 同步更新，在 PC+4 PC+4+bgtz offset j_address 三者间选择 PC_next
6. I-type 指令需要 signextended 部件保证数据对齐

具体原理图：



注意：此状态图为异步时钟情况，改成同步时钟需要在 S0 和 S1 之间加一个状态，因为读 Memory 数据还需要延时一个时钟周期



源代码：

综合系统：

```
module cpu(
    input clk,
    input reset
    );
    reg [31:0] PC_Reg;
    reg [31:0] Instr_Reg;
    reg [31:0] ALU_Reg;
    reg [31:0] Data;
    wire [31:0] Mem_Addr;
    wire [31:0] Mem_Out;
    reg [31:0] Instruction;
    wire [31:0] Reg_Addr3;
    wire [31:0] Reg_In;
    reg [31:0] Reg_ALU;
    wire [31:0] Signextended;
    wire [31:0] Shifted;
    wire [31:0] A;
    wire [31:0] B;
```

```
reg [31:0] Reg_RD1;
reg [31:0] Reg_RD2;
wire [31:0] ALU_Result;
wire [31:0] ALU_A;
wire [31:0] ALU_B;
wire ALU_Zero;
reg [31:0] ALU_Out;
wire [31:0] PC_Next;
wire [31:0] PC_Jump;
wire PCen;
wire MemRead;
wire MemWrite;
wire RegDst;
wire IRWrite;
wire PCWrite;
wire bgtz;
wire MemtoReg;
wire ALUSrcA;
wire [1:0] ALUSrcB;
wire [1:0] PCSrc;
wire IorD;
wire RegWrite;
Memory Memory1(
.clka            (clk),
.wea             (MemWrite),
.addra           (Mem_Addr[11:2]),
.dina                (Reg_RD2),
.douta           (Mem_Out)
);

ControlUnit ControlUnit1(
.reset           (reset),
.opcode          (Instruction[31:26]),
.clk             (clk),
.RegDst          (RegDst),
.IRWrite         (IRWrite),
.PCWrite         (PCWrite),
.bgtz                (bgtz),
.MemtoReg        (MemtoReg),
.MemWrite        (MemWrite),
.ALUSrcA         (ALUSrcA),
.ALUSrcB         (ALUSrcB),
.PCSrc           (PCSrc),
.RegWrite        (RegWrite),
```

```
    .IorD                   (IorD)
    );

    Registers Registers1(
    .clk                (clk),
    .read_address1      (Instruction[25:21]),
    .read_address2      (Instruction[20:16]),
    .write_address      (Reg_Addr3),
    .write_data         (Reg_In),
    .write_enable (RegWrite),
    .read_data1         (A),
    .read_data2         (B)
    );
    ALU ALU1(
    .A                      (ALU_A),
    .B                      (ALU_B),
    .ALU_Zero           (ALU_Zero),
    .ALU_result         (ALU_Result)
    );

    SignExtend SignExtend_1(
    .in                 (Instruction[15:0]),
    .out                (Signextended)
    );


    Get_PC_Jump Get_PC_Jump_1(
    .in                 (Instruction[25:0]),
    .pc                 (PC_Reg[31:28]),
    .out                (PC_Jump)
    );
    assign Shifted[31:2]=Signextended[29:0];
    assign Shifted[1:0]=2'b00;
    assign Mem_Addr=IorD?ALU_Reg:PC_Reg;
    assign Reg_Addr3=RegDst?Instruction[15:11]:Instruction[20:16];
    assign Reg_In=MemtoReg?Mem_Out:ALU_Reg;
    assign ALU_A=ALUSrcA?Reg_RD1:PC_Reg;
    assign
ALU_B=(ALUSrcB==2'b00)?Reg_RD2:((ALUSrcB==2'b01)?4:((ALUSrcB==2'b10)?Signextende
d:Shifted));

    always@(posedge clk or negedge reset)
    begin
        if(~reset)
```

```
            PC_Reg<=32'b0;
        else if(PCen)
            PC_Reg<=PC_Next;
    end

    always@(posedge clk)
    begin
        Reg_RD1<=A;
        Reg_RD2<=B;
    end

    always@(posedge clk)
    begin
        if(IRWrite)
            Instruction<=Mem_Out;
    end

    always@(posedge clk)
    begin
        ALU_Reg<=ALU_Result;
    end
    assign PC_Next=(PCSrc==2'b00)?ALU_Result:((PCSrc==2'b01)?ALU_Reg:PC_Jump);

    assign PCen=PCWrite+(ALU_Zero*bgtz);

endmodule
```

RAM 初始化文件：

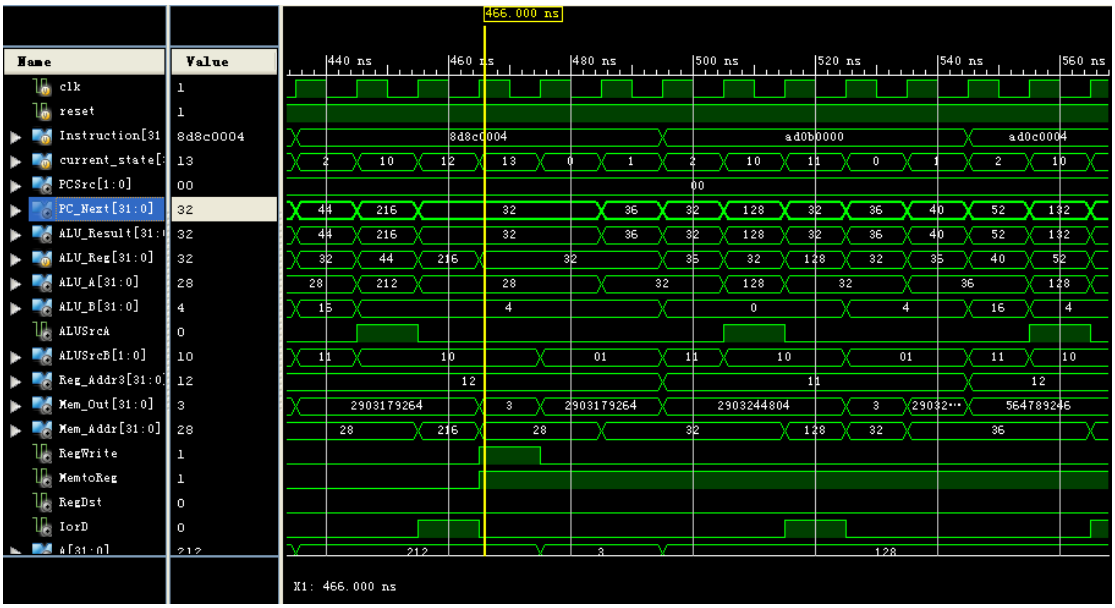
```
MEMORY_INITIALIZATION_RADIX=16;
MEMORY_INITIALIZATION_VECTOR=
20080080,
200d00d0,
8dad0000,
200b00d4,
8d6b0000,
200c00d4,
8d8c0004,
ad0b0000,
ad0c0004,
21a9fffe,
8d0b0000,
8d0c0004,
016c5020,
ad0a0008,
```

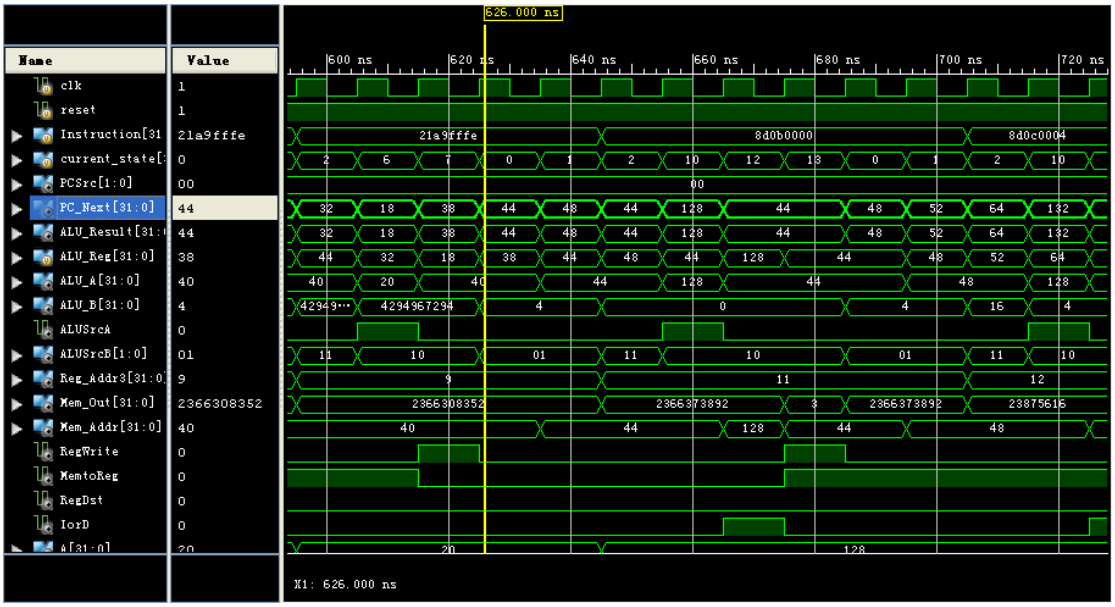
21080004,
2129ffff,
1d20fff9,
08000011,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000014,
00000003,
00000003,
00000000;

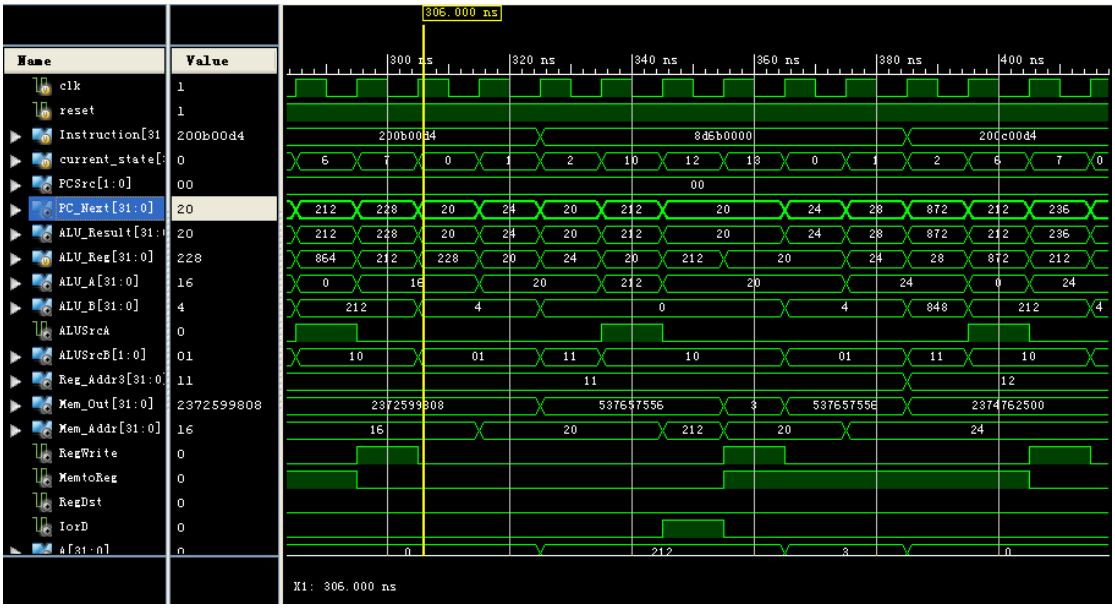
仿真结果

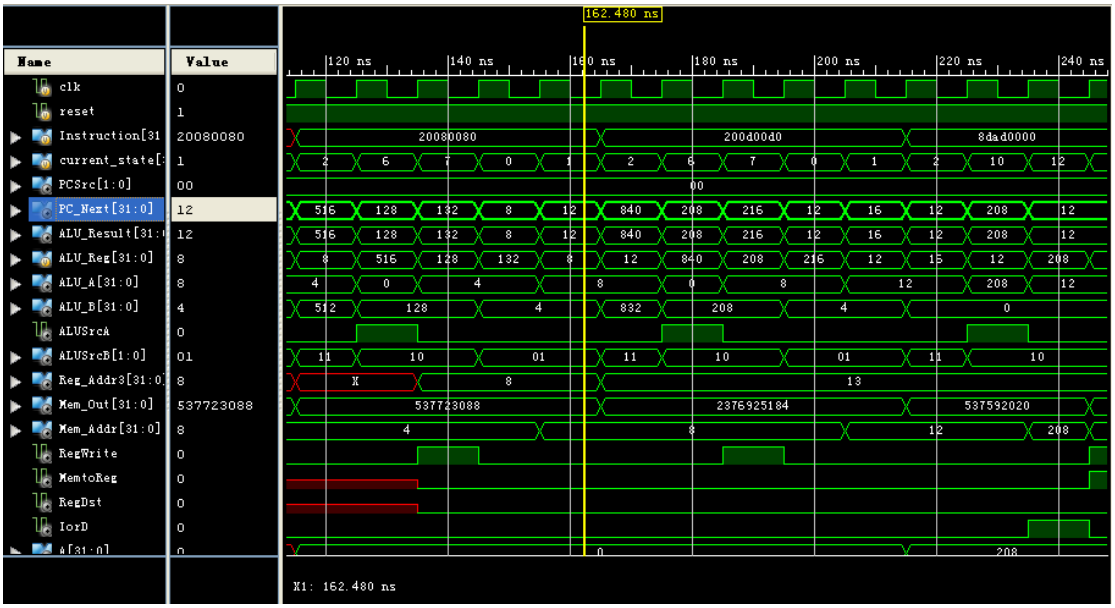
	0	1
0x12	0	0
0x14	0	0
0x16	0	0
0x18	0	0
0x1A	0	0
0x1C	0	0
0x1E	0	0
0x20	3	3
0x22	6	9
0x24	15	24
0x26	39	63
0x28	102	165
0x2A	267	432
0x2C	699	1131
0x2E	1830	2961
0x30	4791	7752
0x32	12543	20295

	0	1
0x0	20080080	200D00D0
0x2	8DAD0000	200B00D4
0x4	8D6B0000	200C00D4
0x6	8D8C0004	AD0B0000
0x8	AD0C0004	21A9FFFE
0xA	8D0B0000	8D0C0004
0xC	016C5020	AD0A0008
0xE	21080004	2129FFFF
0x10	1D20FFF9	08000011
0x12	00000000	00000000




626.000 ns
Name
Value
clk
reset
Instruction[31
current_state[
PCSrc[1:0]
PC_Next[31:0]
ALU_Result[31:
ALU_Reg[31:0]
ALU_A[31:0]
ALU_B[31:0]
ALUSrcA
ALUSrcB[1:0]
Reg_Addr3[31:0]
Mem_Out[31:0]
Mem_Addr[31:0]
RegWrite
MemtoReg
RegDst
IorD
600 ns
620 ns
640 ns
660 ns
680 ns
700 ns
720 ns
X1: 626.000 ns


306.000 ns
Name
Value
clk
reset
Instruction[31
current_state[
PCSrc[1:0]
PC_Next[31:0]
ALU_Result[31:
ALU_Reg[31:0]
ALU_A[31:0]
ALU_B[31:0]
ALUSrcA
ALUSrcB[1:0]
Reg_Addr3[31:0]
Mem_Out[31:0]
Mem_Addr[31:0]
RegWrite
MemtoReg
RegDst
IorD
300 ns
320 ns
340 ns
360 ns
380 ns
400 ns
X1: 306.000 ns


162.480 ns
Name
Value
clk
reset
Instruction[31
current_state[
PCSrc[1:0]
PC_Next[31:0]
ALU_Result[31:
ALU_Reg[31:0]
ALU_A[31:0]
ALU_B[31:0]
ALUSrcA
ALUSrcB[1:0]
Reg_Addr3[31:0]
Mem_Out[31:0]
Mem_Addr[31:0]
RegWrite
MemtoReg
RegDst
IorD
120 ns
140 ns
160 ns
180 ns
200 ns
220 ns
240 ns
X1: 162.480 ns