实验报告五
PB14000556 陈晓彤

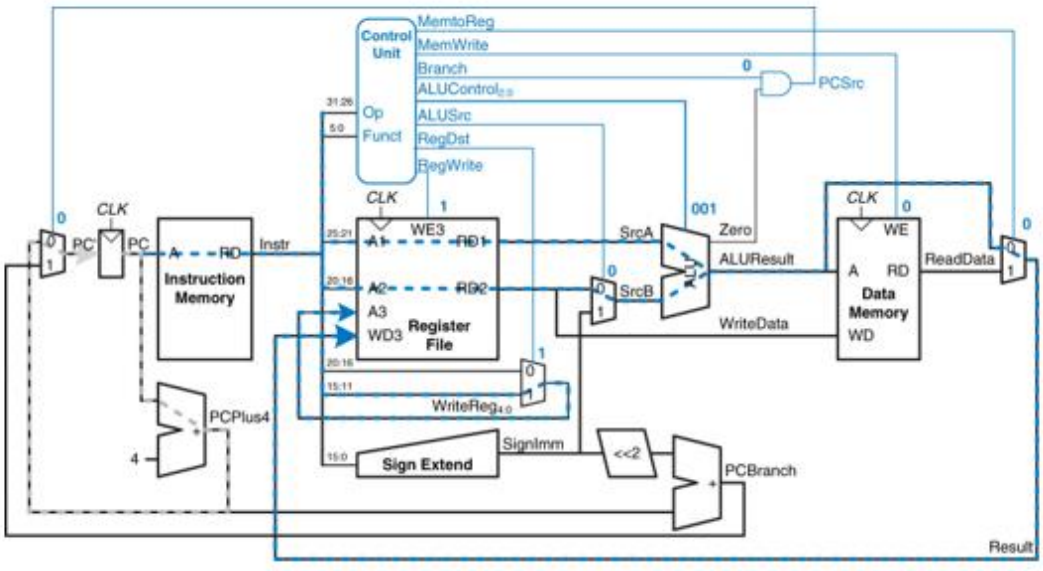
实验题目：设计一个单周期 CPU 并成功运行一个计算 Fibonacci 数列的程序
实验要求：

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 InstructionMemory RegisterFile ALU DataMemory ControlUnit
4. 设计合理的控制信号，保证系统在单周期执行完指令

设计思路：

1. DataMemory 由 ipcore—distributed single port RAM 实现，异步读取，同步写入，初始化文件命名为 RAM_initial.coe 其内容为 Mars 编译出的汇编指令数据段，但一定要注意修改格式：各个单元数据之间要加逗号，开头两行要加上 MEMORY_INITIALIZATION_RADIX=10; MEMORY_INITIALIZATION_VECTOR=
2. Instruction Memory 由 ipcore—distributed ROM 实现，异步读取，初始化文件命名为 RAM_initial.coe 其内容为 Mars 编译出的汇编指令代码段，同样注意修改格式
3. RegisterFile 由 32 个 32 位寄存器实现，采用异步读取，同步写入的方式，初始化使 0 号寄存器内容为 0
4. ControlUnit 发出信号：MemtoReg MemWrite Bgtz RegDst RegWrite ALUSrc j
   a. MemtoReg—选择存入寄存器的数据来源 ALU/Memory
   b. MemWrite RegWrite—数据内存和寄存器写使能
   c. Bgtz j—跳转指令控制信号
   d. RegDst—寄存器写地址来源 rt/rd
   e. ALUSrc—alu_b 来源 reg_read_2/imm16
5. ALU 只进行加法计算，控制信号固定为 3' b010,alu_a 来源为寄存器，alu_b 来源由 ALUSrc 决定
6. PC 同步更新，在 PC+4 PC+4+bgtz offset j_address 三者间选择 PC_next
7. I-type 指令需要 signextended 部件保证数据对齐

具体原理图：



源代码：
综合系统：

```
module cpu(
    input clk,
    output [31:0] ALU_result
    );
    parameter Instruction_Memory_size=10,Data_Memory_size=7;
    reg    [31:0] PC;
    wire    [4:0]    Reg_write_address;
    reg sign_gtz;
    wire [31:0]      Reg_write_data;
    reg [31:0]    PC_next;
    wire [31:0]    alu_b;

    wire [31:0] Instruction;
    wire [31:0] signextended;
    wire [31:0] j_address;

    wire [31:0] Mem_read_data;
    wire [31:0] Reg_read_data1;
    wire [31:0] Reg_read_data2;

    wire RegDst,j,bgtz,MemtoReg,ALUSrc,RegWrite,MemWrite;

    ROM Instruction_Memory(
    .a                  (PC[Instruction_Memory_size+1:2]),
    .spo                (Instruction[31:0])
    );
    ControlUnit ControlUnit1(
    .opcode             (Instruction[31:26]),
    .RegDst             (RegDst),
    .j                  (j),
    .bgtz               (bgtz),
    .MemtoReg           (MemtoReg),
    .MemWrite           (MemWrite),
    .ALUSrc             (ALUSrc),
    .RegWrite           (RegWrite)
    );
    RAM Data_Memory(
    .clk                (clk),
    .a                  (ALU_result[Data_Memory_size+1:2]),
    .d                  (Reg_read_data2),
    .we                 (MemWrite),
    .spo                (Mem_read_data)
```

```
);
Registers Registers1(
.clk                (clk),
.read_address1      (Instruction[25:21]),
.read_address2      (Instruction[20:16]),
.write_address      (Reg_write_address),
.write_data         (Reg_write_data),
.write_enable       (RegWrite),
.read_data1         (Reg_read_data1),
.read_data2         (Reg_read_data2)
);
ALU ALU1(
.A                  (Reg_read_data1),
.B                  (alu_b),
.op                 (3'b010),
.ALU_result         (ALU_result)
);

SignExtend16 SignExtend16_1(
.in                 (Instruction[15:0]),
.out                (signextended)
);
SignExtend26 SignExtend26_1(
.in                 (Instruction[25:0]),
.pc                 (PC[31:28]),
.out                (j_address)
);
always@(ALU_result)
begin
    sign_gtz=(ALU_result>0);
end

always@(*)
begin
    if(j)
        PC_next=j_address;
    else if(sign_gtz && bgtz)
        PC_next=PC+32'b100+(signextended<<2);
    else
        PC_next=PC+32'b100;
end
always@(posedge clk)
begin
    if(PC_next)
```

```
                PC=PC_next;

    end

    assign Reg_write_address=(RegDst)?(Instruction[15:11]):(Instruction[20:16]);
    assign alu_b=(ALUSrc)?(signextended):(Reg_read_data2);
    assign Reg_write_data=(MemtoReg)?(Mem_read_data):(ALU_result);

    initial
    begin
        PC=32'h0;
    end

endmodule
```

各个部件：

```
module ControlUnit(
    input [5:0] opcode,
    output bgtz,
    output j,
    output RegDst,
    output ALUSrc,
    output MemtoReg,
    output MemWrite,
    output RegWrite
    );

    wire add,lw,sw,addi;
    assign bgtz=(opcode==6'd7);
    assign j=(opcode==6'd2);
    assign add=(opcode==6'b0);
    assign lw=(opcode==6'd35);
    assign sw=(opcode==6'd43);
    assign addi=(opcode==6'd8);

    assign RegDst=add;
    assign ALUSrc=lw+sw+addi;
    assign MemWrite=sw;
    assign MemtoReg=lw;
    assign RegWrite=add+lw+addi;

endmodule
```

```
module Registers(
	input clk,
	input	[4:0] read_address1,
	input	[4:0] read_address2,
	input	[4:0] write_address,
	input	[31:0] write_data,
	input	write_enable,
	output	[31:0] read_data1,
	output	[31:0] read_data2
	);
	reg	[31:0] registers [31:0];

	initial
	begin
		registers[0]=32'b0;
	end
	assign read_data1=registers[read_address1];
	assign read_data2=registers[read_address2];
	always@(posedge clk)
		begin
		if(write_enable)
			registers[write_address]=write_data;

		end
endmodule

module ALU(
	input [31:0] A,
	input [31:0] B,
	input [2:0] op,
	output [31:0] ALU_result

	);

	assign ALU_result=(A+B)*(op==3'b010)+(A-B)*(op==3'b110)+(A|B)*(op==3'b001);

endmodule

module SignExtend16(
	input [15:0] in,
	output [31:0] out
	);
	assign out[15:0]=in[15:0];
	assign out[31]=in[15];assign out[30]=in[15];assign out[29]=in[15];assign out[28]=in[15];
```

```
    assign out[27]=in[15];assign out[26]=in[15];assign out[25]=in[15];assign out[24]=in[15];
    assign out[23]=in[15];assign out[22]=in[15];assign out[21]=in[15];assign out[20]=in[15];
    assign out[19]=in[15];assign out[18]=in[15];assign out[17]=in[15];assign out[16]=in[15];

endmodule

module SignExtend26(
    input [25:0] in,
    input [3:0]    pc,
    output [31:0] out
    );
    assign out[31]=pc[3];assign out[30]=pc[2];assign out[29]=pc[1];assign out[28]=pc[0];
    assign out[27:2]=in[25:0];
    assign out[1]=0;assign out[0]=0;

endmodule
```

ROM 与 RAM 初始化文件：

ROM（指令流）

```
MEMORY_INITIALIZATION_RADIX=16;
MEMORY_INITIALIZATION_VECTOR=
20080000,
200d0050,
8dad0000,
200b0054,
8d6b0000,
200c0054,
8d8c0004,
ad0b0000,
ad0c0004,
21a9fffe,
8d0b0000,
8d0c0004,
016c5020,
ad0a0008,
21080004,
2129ffff,
1d20fff9,
08000011,
0;
```

RAM（初始化数据）

```
MEMORY_INITIALIZATION_RADIX=16;
MEMORY_INITIALIZATION_VECTOR=
```

00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000014,
00000003,
00000003,
00000000;

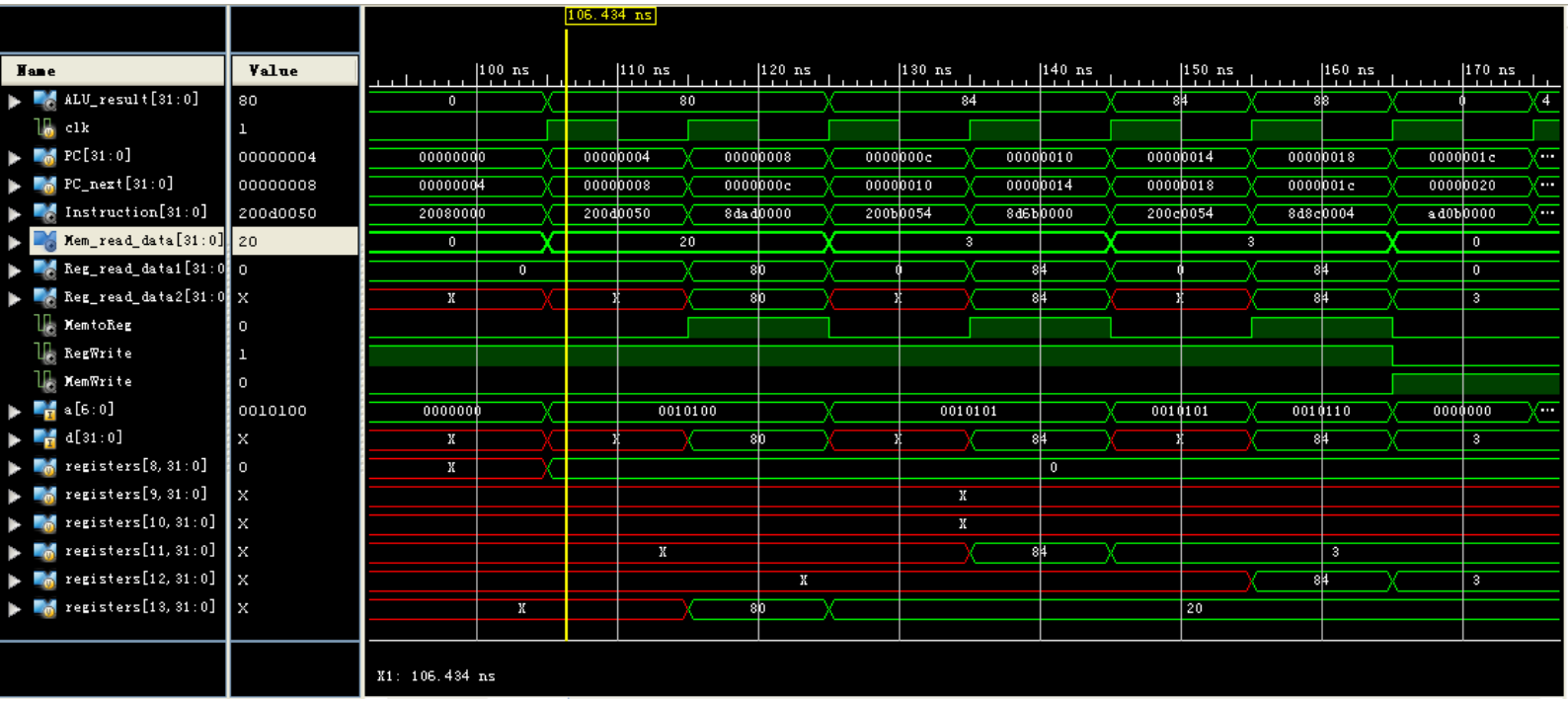
仿真结果：

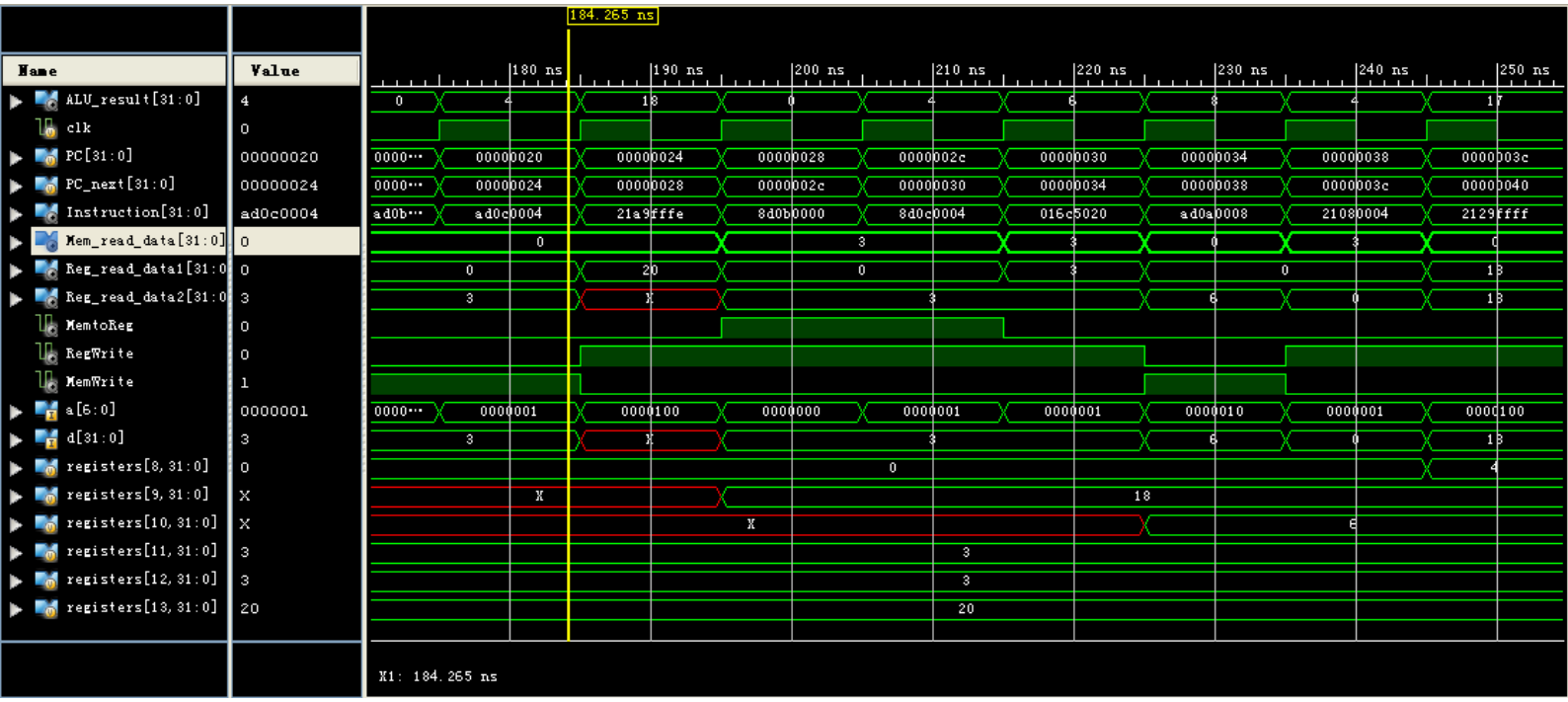
| | | |
|---|---|---|
| 0x17 | 0 | 3 |
| 0x15 | 3 | 20 |
| 0x13 | 0 | 0 |
| 0x11 | 0 | 0 |
| 0xF | 0 | 0 |
| 0xD | 0 | 0 |
| 0xB | 0 | 0 |
| 0x9 | 0 | 0 |
| 0x7 | 0 | 0 |
| 0x5 | 0 | 0 |
| 0x3 | 0 | 0 |
| 0x1 | 0 | 0 |

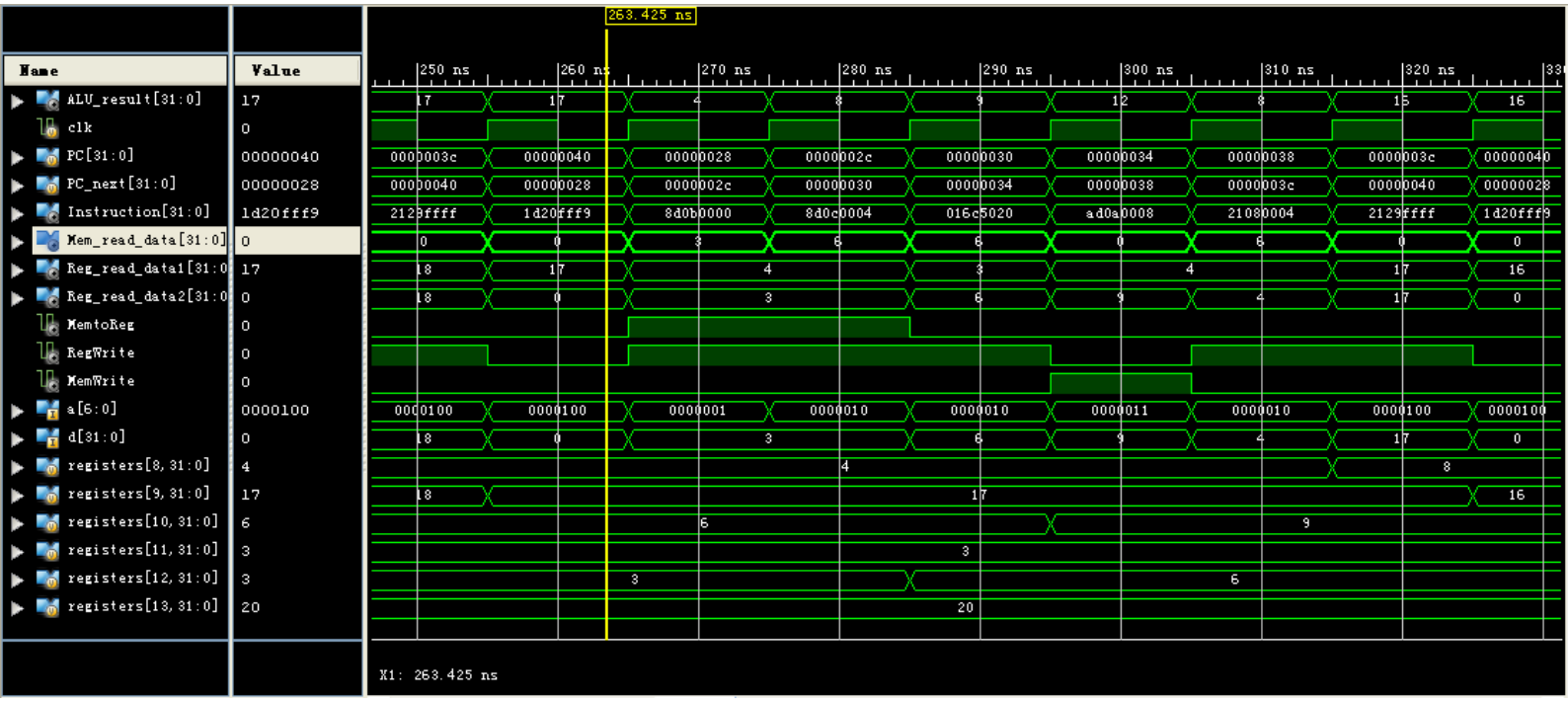
| | | |
|---|---|---|
| 0x17 | 0 | 3 |
| 0x15 | 3 | 20 |
| 0x13 | 20295 | 12543 |
| 0x11 | 7752 | 4791 |
| 0xF | 2961 | 1830 |
| 0xD | 1131 | 699 |
| 0xB | 432 | 267 |
| 0x9 | 165 | 102 |
| 0x7 | 63 | 39 |
| 0x5 | 24 | 15 |
| 0x3 | 9 | 6 |
| 0x1 | 3 | 3 |

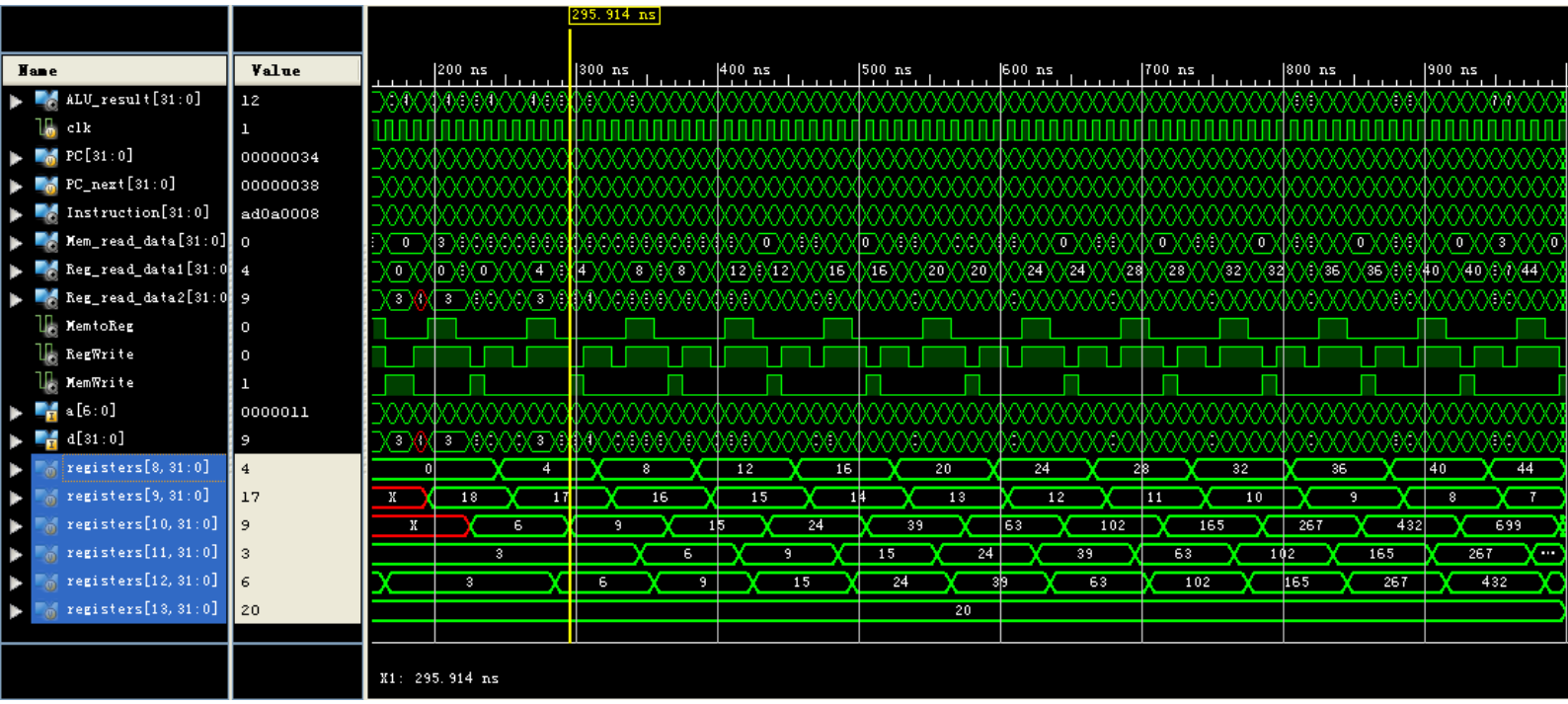
| | | |
|---|---|---|
| 0x11 | X | X |
| 0xF | X | X |
| 0xD | 20 | 12543 |
| 0xB | 7752 | 20295 |
| 0x9 | 0 | 72 |
| 0x7 | X | X |
| 0x5 | X | X |
| 0x3 | X | X |
| 0x1 | X | 0 |

| | | |
|---|---|---|
| 0x11 | 08000011 | 1D20FFF9 |
| 0xF | 2129FFFF | 21080004 |
| 0xD | AD0A0008 | 016C5020 |
| 0xB | 8D0C0004 | 8D0B0000 |
| 0x9 | 21A9FFFE | AD0C0004 |
| 0x7 | AD0B0000 | 8D8C0004 |
| 0x5 | 200C0054 | 8D6B0000 |
| 0x3 | 200B0054 | 8DAD0000 |
| 0x1 | 200D0050 | 20080000 |




263.425 ns
Name
Value
ALU_result[31:0]
17
clk
0
PC[31:0]
00000040
PC_next[31:0]
00000028
Instruction[31:0]
1d20fff9
Mem_read_data[31:0]
0
Reg_read_data1[31:0]
17
Reg_read_data2[31:0]
0
MemtoReg
0
RegWrite
0
MemWrite
0
a[6:0]
0000100
d[31:0]
0
registers[8,31:0]
4
registers[9,31:0]
17
registers[10,31:0]
6
registers[11,31:0]
3
registers[12,31:0]
3
registers[13,31:0]
20
250 ns
260 ns
270 ns
280 ns
290 ns
300 ns
310 ns
320 ns
X1: 263.425 ns


295.914 ns
Name
Value
ALU_result[31:0]
12
clk
1
PC[31:0]
00000034
PC_next[31:0]
00000038
Instruction[31:0]
ad0a0008
Mem_read_data[31:0]
0
Reg_read_data1[31:0]
4
Reg_read_data2[31:0]
9
MemtoReg
0
RegWrite
0
MemWrite
1
a[6:0]
0000011
d[31:0]
9
registers[8,31:0]
4
registers[9,31:0]
17
registers[10,31:0]
9
registers[11,31:0]
3
registers[12,31:0]
6
registers[13,31:0]
20
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns
800 ns
900 ns
X1: 295.914 ns