数字逻辑与处理器基础实验
第五次作业

无 81 马啸阳 2018011054

2020 年 6 月 4 日

1 实验目的

完成一个单周期处理器的控制器部分，实现 MIPS 指令集的一个子集，包括：

lw, sw, lui

add, addu, sub, subu, addi, addiu

and, or, xor, nor, andi, sll, srl, sra, slt, sltu, sltiu

beq, j, jal, jr, jalr

2 实验原理

单周期处理器结构如图 1所示，由指令存储器、指令加法器、指令译码部分、控制单元、寄存器堆、ALU、数据存储器等构成。要实现的指令格式参考 MIPS 指令集格式，此处略去。单周期处理器的具体原理略。

控制单元真值表如表 1所示。

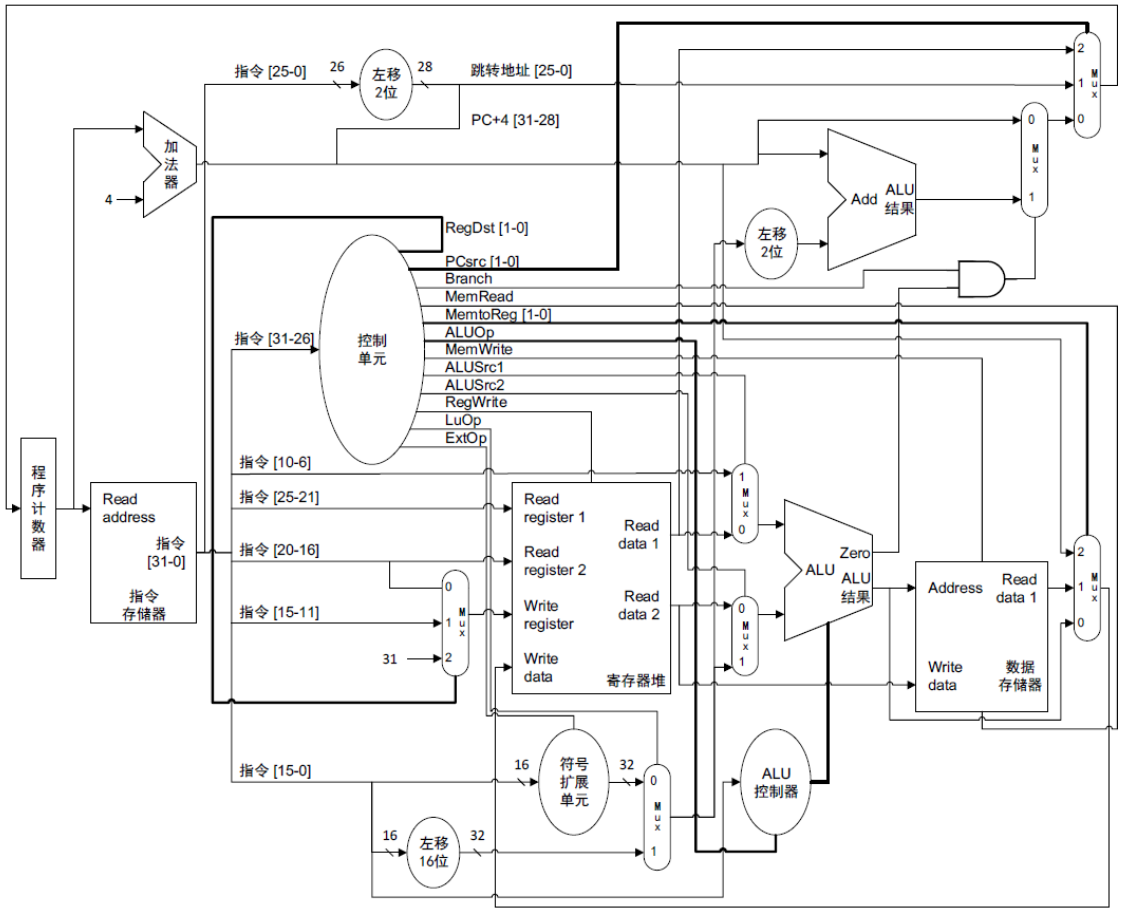
指令 [25-0]
26
左移
2位
28
跳转地址 [25-0]
PC+4 [31-28]
加
法
器
4
RegDst [1-0]
PCsrc [1-0]
Branch
MemRead
MemtoReg [1-0]
ALUOp
MemWrite
ALUSrc1
ALUSrc2
RegWrite
LuOp
ExtOp
指令 [31-26]
控制
单元
Add
ALU
结果
左移
2位
程
序
计
数
器
Read
address
指令
[31-0]
指令
存储器
指令 [10-6]
指令 [25-21]
指令 [20-16]
指令 [15-11]
31
Read
register 1
Read
register 2
Write
register
Write
data
Read
data 1
Read
data 2
寄存器堆
ALU
Zero
ALU
结果
Address
Read
data 1
Write
data
数据
存储器
指令 [15-0]
16
符号
扩展
单元
32
16
左移
16位
32
ALU
控制器
M
u
x

图 1: 单周期处理器结构

	PCSrc[1:0]	Branch	RegWrite	RegDst[1:0]	MemRead	MemWrite	MemtoReg[1:0]	ALUSrc1	ALUSrc2	ExtOp	LuOp
lw	0	0	1	0	1	0	1	0	1	1	0
sw	0	0	0	X	0	1	X	0	1	1	0
lui	0	0	1	0	0	0	0	0	1	X	1
add	0	0	1	1	0	0	0	0	0	X	X
addu	0	0	1	1	0	0	0	0	0	X	X
sub	0	0	1	1	0	0	0	0	0	X	X
subu	0	0	1	1	0	0	0	0	0	X	X
addi	0	0	1	0	0	0	0	0	1	1	0
addiu	0	0	1	0	0	0	0	0	1	1	0
and	0	0	1	1	0	0	0	0	0	X	X
or	0	0	1	1	0	0	0	0	0	X	X
xor	0	0	1	1	0	0	0	0	0	X	X
nor	0	0	1	1	0	0	0	0	0	X	X
andi	0	0	1	0	0	0	0	0	1	0	0
sll	0	0	1	1	0	0	0	1	0	X	X
srl	0	0	1	1	0	0	0	1	0	X	X
sra	0	0	1	1	0	0	0	1	0	X	X
slt	0	0	1	1	0	0	0	0	0	X	X
sltu	0	0	1	1	0	0	0	0	0	X	X
slti	0	0	1	0	0	0	0	0	1	1	0
sltiu	0	0	1	0	0	0	0	0	1	1	0
beq	0	1	0	X	0	0	X	0	0	1	0
j	1	X	0	X	0	0	X	X	X	X	X
jal	1	X	1	2	0	0	2	X	X	X	X
jr	2	X	0	X	0	0	X	X	X	X	X
jalr	2	X	1	1	0	0	2	X	X	X	X

表 1: 控制信号真值表

实验执行如下汇编代码，其中添加了注释。这段代码展现了函数调用，主程序调用 sum(n)（若第一条指令中 3 改为 n）。sum 中为条件分支，忽略其中对寄存器的保存恢复，则 sum 中代码如注释所述。当 $a0<1 时返回 0，反之跳转至 L1，L1 调用 sum，返回 $a0-1+sum($a0-1)。因此此程序完成 $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$，n<=0 时返回 0，计算结果置于 $v0 中，然后跳至 Loop，进入死循环，各寄存器值不变。

```
    addi $a0, $zero, 3      # $a0=3
    jal sum                 # jump to sum, $ra points to Loop
Loop:
    beq $zero, $zero, Loop  # infinite Loop
sum:
    # if($a0<1)
    #   return 0;
    # else
    #   goto L1;
    #   (return $a0+sum($a0-1))
    addi $sp, $sp, -8       # $sp=$sp-8
    sw $ra, 4($sp)          # save $ra in ($sp)[4]
    sw $a0, 0($sp)          # save $a0 in ($sp)[0]
    slti $t0, $a0, 1        # if $a0<1 then $t0=1
    beq $t0, $zero, L1      # if $t0==0 ($a0>=1) then jump to L1
    xor $v0, $zero, $zero   # $v0=0
    addi $sp, $sp, 8        # $sp=$sp+8
    jr $ra                  # jump to $ra
L1:
    # return $a0+sum($a0-1)
    addi $a0, $a0, -1       # $a0=$a0-1
    jal sum                 # jump to sum, $ra points to next instruction
    lw $a0, 0($sp)          # load ($sp)[0] to $a0
    lw $ra, 4($sp)          # load ($sp)[4] to $ra
    addi $sp, $sp, 8        # $sp=$sp+8
    add $v0, $a0, $v0       # $v0=$a0+$v0
    jr $ra                  # jump to $ra
```

根据各指令格式将各语句翻译为机器码，得到如下 18 条机器码指令（以下为十六进制）。

```
20040003
0c000003
1000ffff
23bdfff8
```

```
afbf0004
afa40000
28880001
11000003
00001026
23bd0008
03e00008
2084ffff
0c000003
8fa40000
8fbf0004
23bd0008
00821020
03e00008
```

PC 的值每周期为当前执行的指令地址。例如第二周期 PC=0x00000004，执行第二条指令 jal sum，跳转至 sum，从而第三周期 PC=0x0000000b，到达 sum 的开始位置。

各寄存器与 PC 变化按时间变化列举如下，以下均为十六进制。

第一周期将 3 置入 $a0，第二周期开始 $a0=0x00000003。

第二周期将当前指令下一条地址置入 $ra，第三周期开始 $ra=0x00000008，同时 PC 跳转至 sum，即 0x0000000c。

第三周期将 $sp 减 8，在后两个周期进行压栈，第四周期开始 $sp 变为 0xfffffff8（此处 $sp 初始化为 0，从而栈在数据存储器的最后反向增长）。

第七周期执行 beq $t0, $zero, L1，跳转至 L1，第八周期开始 PC 变为 0x0000002c。

第八周期 $a0 减 1，第九周期开始 $a0=0x00000002 准备调用 sum(2)。

第九周期执行 jal sum，调用 sum(2)，将下一条指令地址置入 $ra，第十周期开始 $ra=0x00000034，同时 PC 跳转至 sum，即 0x0000000c。

第十周期与第三周期指令一致，第十一周期开始 $sp 减 8 变为 0xfffffff0。

第十四周期与第七周期指令一致，跳转至 L1。

第十五周期与第八周期指令一致，$a0 减 1，第十六周期开始 $a0=0x00000001 准备调用 sum(1)。

第十六周期与第九周期指令一致，执行 jal sum，调用 sum(1)，下一条指令仍然是 0x00000034，$ra 没有实际变化，PC 跳转至 sum。

第十七至二十三周期与第三至九、第十至十六周期一致。第十八周期开始 $sp 变为 0xffffffe8，

第二十三周期开始 $a0 变为 0x00000000 准备调用 sum(0)。

第二十四周期将 $sp 减 8，第二十五周期开始 $sp 变为 0xfffffe0。

第二十八周期执行 beq $t0, $zero, L1，由于 $a0 已递归至 0，因此不跳转。

第二十九周期置 $v0 为 0。sum(0) 返回 0。

第三十周期开始弹栈，$sp 增 8，第三十一周期开始 $sp=0xfffffe8。

第三十一周期执行 jr $ra，sum(0) 执行完毕，进入 sum(1) 的函数栈，第三十二周期开始 PC 跳转至 lw $a0, 0($sp) 一句指令，即 0x00000034。

第三十二周期由栈中恢复 $a0，第三十三周期开始 $a0=0x00000001。

第三十三周期由栈中恢复 $ra，而 sum(1) 的 $ra 仍是 0x00000034，实际不变。

第三十四周期开始弹栈，$sp 增 8，第三十五周期开始 $sp=0xffffff0。

第三十五周期执行 add $v0, $a0, $v0，第三十六周期开始 $v0=0x00000001。sum(1) 返回 1。

第三十六周期执行 jr $ra，sum(1) 执行完毕，进入 sum(2) 函数栈，第三十七周期开始 PC 跳转至 0x00000034。

第三十七至四十一周期与第三十二至三十六周期指令一致。第三十八周期开始 $a0 变为 0x00000002，第三十九周期开始恢复 $ra，第四十周期开始 $sp 变为 0xffffff8，第四十一周期开始 $v0=2+$v0=3（sum(2) 返回 3）。

第四十二周期 PC 又跳回 0x00000034，进入 sum(3) 函数栈。同上，第四十三周期开始 $a0 变为 0x00000004，第四十四周期开始恢复 $ra，$ra 变为 0x00000008 为 Loop 的地址，第四十五周期开始 $sp 变为 0x00000000，第四十六周期开始 $v0=3+$v0=6（sum(3) 返回 6）。第四十六周期执行 jal $ra 后，第四十七周期 PC 跳转为 0x00000008，跳转到 Loop。此后所有寄存器与 PC 值不再改变。

3 实验代码

3.1 文件清单

```
.
├── ALU.v ..... ALU
├── ALUControl.v ..... ALU 控制器
├── Control.v ..... 控制单元
├── CPU_tb.v ..... CPU 测试模块
├── CPU.v ..... 单周期 CPU
└── CPU.xdc ..... CPU 管脚约束
```

3.2 控制单元

控制单元按照控制信号真值表指定每一个控制信号在特定指令下的值，其中真值表中的 X 可任意指定。

```
module Control(OpCode, Funct,
        PCSrc, Branch, RegWrite, RegDst,
        MemRead, MemWrite, MemtoReg,
        ALUSrc1, ALUSrc2, ExtOp, LuOp, ALUOp);
        input [5:0] OpCode;
        input [5:0] Funct;
        output [1:0] PCSrc;
        output Branch;
        output RegWrite;
        output [1:0] RegDst;
        output MemRead;
        output MemWrite;
        output [1:0] MemtoReg;
        output ALUSrc1;
        output ALUSrc2;
        output ExtOp;
        output LuOp;
        output [3:0] ALUOp;

        assign PCSrc[1:0] =
                (OpCode == 6'h02 || OpCode == 6'h03)? 2'b01:
                (OpCode == 6'h00 && (Funct == 5'h08 || Funct == 5'h09))? 2'b10:
                2'b00;

        assign Branch =
                (OpCode == 6'h04)? 1'b1:
                1'b0;

        assign RegWrite =
                (OpCode == 6'h2b || OpCode == 6'h02 || OpCode == 6'h04 ||
```

```
31                 (OpCode == 6'h00 && Funct == 5'h08))? 1'b0:
32                 1'b1;
33
34         assign RegDst[1:0] =
35                 (OpCode == 6'h23 || OpCode == 6'h0f || OpCode == 6'h08 ||
36                 OpCode == 6'h09 || OpCode == 6'h0c || OpCode == 6'h0a ||
37                 OpCode == 6'h0b)? 2'b00:
38                 (OpCode == 6'h03)? 2'b10:
39                 2'b01;
40
41         assign MemRead =
42                 (OpCode == 6'h23)? 1'b1:
43                 1'b0;
44
45         assign MemWrite =
46                 (OpCode == 6'h2b)? 1'b1:
47                 1'b0;
48
49         assign MemtoReg[1:0] =
50                 (OpCode == 6'h23)? 2'b01:
51                 (OpCode == 6'h03 ||
52                 (OpCode == 6'h00 && Funct == 5'h09))? 2'b10:
53                 2'b00;
54
55         assign ALUSrc1 =
56                 (OpCode == 6'h00 && (Funct == 5'h00 ||
57                 Funct == 5'h02 || Funct == 5'h03))? 1'b1:
58                 1'b0;
59
60         assign ALUSrc2 =
61                 (OpCode == 6'h23 || OpCode == 6'h2b ||
62                 OpCode == 6'h0f || OpCode == 6'h08 ||
63                 OpCode == 6'h09 || OpCode == 6'h0c ||
64                 OpCode == 6'h0a || OpCode == 6'h0b)? 1'b1:
65                 1'b0;
66
67         assign ExtOp =
68                 (OpCode == 6'h0c)? 1'b0:
69                 1'b1;
```

```
70
71          assign LuOp =
72                  (OpCode == 6'h0f)? 1'b1:
73                  1'b0;
74
75          assign ALUOp[2:0] =
76                  (OpCode == 6'h00)? 3'b010:
77                  (OpCode == 6'h04)? 3'b001:
78                  (OpCode == 6'h0c)? 3'b100:
79                  (OpCode == 6'h0a || OpCode == 6'h0b)? 3'b101:
80                  3'b000;
81
82          assign ALUOp[3] = OpCode[0];
83
84  endmodule
```

3.3 寄存器堆

为了避免因缺少输出，工具将大量电路优化掉，在寄存器堆加入一个两位的控制输入和一个 8 bit 输出端，两位的控制输入用以选择输出端是 $a0，$v0，$sp 或 $ra 的低 8 位。

```
1   module RegisterFile(control, reset, clk, RegWrite, Read_register1,
2                       Read_register2, Write_register, Write_data,
3                       Read_data1, Read_data2, register);
4           input reset, clk;
5           input RegWrite;
6           input [4:0] Read_register1, Read_register2, Write_register;
7           input [31:0] Write_data;
8           input [1:0] control;
9           output [31:0] Read_data1, Read_data2;
10          output [7:0] register;
11
12          reg [31:0] RF_data[31:1];
13
14          assign Read_data1 = (Read_register1 == 5'b00000)? 32'h00000000:
15                              RF_data[Read_register1];
16          assign Read_data2 = (Read_register2 == 5'b00000)? 32'h00000000:
17                              RF_data[Read_register2];
18          assign register = (control == 2'b00)? RF_data[4][7:0]:
```

```
                        (control == 2'b01)? RF_data[2][7:0]:
                        (control == 2'b10)? RF_data[29][7:0]:
                        RF_data[31][7:0];

        integer i;
        always @(posedge reset or posedge clk)
                if (reset)
                        for (i = 1; i < 32; i = i + 1)
                                RF_data[i] <= 32'h00000000;
                else if (RegWrite && (Write_register != 5'b00000))
                        RF_data[Write_register] <= Write_data;

endmodule
```

3.4 管脚约束

在 CPU 中也添加控制输入和 8 bit 输出端，CPU 代码略去，详见代码附件。管脚上将控制输入绑定 SW0、SW1，输出绑定 LED0 至 LED7。

```
set_property -dict {PACKAGE_PIN W5 IOSTANDARD LVCMOS33} [get_ports {clk}]
set_property -dict {PACKAGE_PIN T18 IOSTANDARD LVCMOS33} [get_ports {reset}]
set_property -dict {PACKAGE_PIN V16 IOSTANDARD LVCMOS33} [get_ports {control[1]}]
set_property -dict {PACKAGE_PIN V17 IOSTANDARD LVCMOS33} [get_ports {control[0]}]
set_property -dict {PACKAGE_PIN E19 IOSTANDARD LVCMOS33} [get_ports {register[7]}]
set_property -dict {PACKAGE_PIN U19 IOSTANDARD LVCMOS33} [get_ports {register[6]}]
set_property -dict {PACKAGE_PIN V19 IOSTANDARD LVCMOS33} [get_ports {register[5]}]
set_property -dict {PACKAGE_PIN W18 IOSTANDARD LVCMOS33} [get_ports {register[4]}]
set_property -dict {PACKAGE_PIN U15 IOSTANDARD LVCMOS33} [get_ports {register[3]}]
set_property -dict {PACKAGE_PIN U14 IOSTANDARD LVCMOS33} [get_ports {register[2]}]
set_property -dict {PACKAGE_PIN V14 IOSTANDARD LVCMOS33} [get_ports {register[1]}]
set_property -dict {PACKAGE_PIN U16 IOSTANDARD LVCMOS33} [get_ports {register[0]}]
create_clock -period 10.000 -name CLK -waveform {0.000 5.000} [get_ports clk]
```

3.5 测试模块

测试模块四次重置处理器，每次以不同控制信号查看不同寄存器的值。依次查看 $a0，$v0，$sp 或 $ra 低 8 位的变化。

```
`timescale 1ns/1ps
`define PERIOD 10

```

```
4  module CPU_tb;
5  reg [1:0] control;
6  reg reset;
7  reg clk;
8  wire [7:0] register;
9
10 CPU cpu1(control, reset, clk, register);
11
12 initial begin
13         control = 2'b00;
14     reset = 1;
15     clk = 1;
16     #100 reset = 0;
17     #900 reset = 1;
18     control = 2'b01;
19     #100 reset = 0;
20     #900 reset = 1;
21     control = 2'b10;
22     #100 reset = 0;
23     #900 reset = 1;
24     control = 2'b11;
25     #100 reset = 0;
26     #900 $finish;
27 end
28
29 always #(`PERIOD/2) clk = ~clk;
30
31 endmodule
```

4　仿真结果与分析

单周期处理器资源消耗情况如图 2所示。

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	F8 Muxes (8150)	Slice (8150)	LUT as Logic (20800)	Bonded IOB (106)	BUFGCTRL (32)
N CPU	3882	8710	1287	512	2926	3882	12	1
alu1 (ALU)	446	0	39	0	166	446	0	0
alu_control1 (ALUControl)	11	0	0	0	5	11	0	0
control1 (Control)	18	0	0	0	14	18	0	0
data_memory1 (DataMemory)	2547	8192	1056	512	2780	2547	0	0
instruction_memory1 (InstructionMemory)	38	0	0	0	22	38	0	0
register_file1 (RegisterFile)	658	480	192	0	263	658	0	0

图 2: 单周期处理器资源消耗情况

总时序性能如图 3所示，建立时间裕量-6.988ns，可工作的最短周期 10+6.988=16.988ns，最大工作频率 1/16.988ns=58.86MHz。

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	-6.988 ns	Worst Hold Slack (WHS):	0.328 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	-33762.011 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	9176	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	17382	Total Number of Endpoints:	17382	Total Number of Endpoints:	8711

Timing constraints are not met.

图 3: 单周期处理器时序性能

限制工作频率的关键路径如图 4所示（这里压缩了模块），是程序计数器至指令存储器，到寄存器堆，经过多路选择器以及 ALU，再到数据存储器，经多路选择器写入寄存器堆。这是 lw 的路径，符合预期 lw 耗时最长。

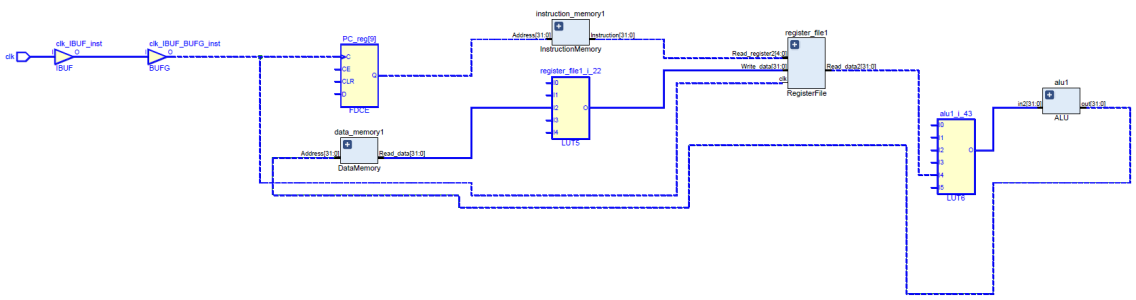

图 4: 关键路径

该关键路径上最耗时的单元是 ALU 前的一个 LUT，触发器延时 0.319ns。该 LUT 应为接收控制信号选择 ALU 输入的多路选择器的一部分。而最长的导线是从 ALU 到数据存储器

的导线，延迟 2.031ns。这比较符合预期，该 LUT 由于有 6 个输入端，控制信号较为复杂，因而耗时较长。

Data Path

Delay Type	Inc... ↓1	Path (...	Location	Netlist Resource(s)
net (fo=2225, routed)	2.031	16.203		data_memory1/Address[3]
net (fo=292, routed)	1.675	8.656		register_file1/Read_register2[4]
net (fo=1, routed)	1.456	19.626		data_memory1/Read_data[14]_INST_0_i_1_n_0
net (fo=15, routed)	1.218	21.873		register_file1/Write_data[14]
net (fo=1, routed)	1.021	10.046		register_file1/Read_data2[21]_INST_0_i_2_n_0
net (fo=14, routed)	0.856	11.875		alu1/in2[21]
net (fo=1, routed)	0.835	17.634		data_memory1/Read_data[14]_INST_0_i_20_n_0
net (fo=2, routed)	0.810	13.553		alu1/out[3]_INST_0_i_12_n_0
net (fo=20, routed)	0.743	6.338		instruction_memory1/Address[9]
net (fo=5, routed)	0.620	12.619		alu1/out[1]_INST_0_i_12_n_0
net (fo=1, routed)	0.606	20.531		Read_data[14]
net (fo=257, routed)	0.552	10.896		Databus2[21]
net (fo=1, routed)	0.395	6.857		instruction_memory1/g0_b9_n_0
net (fo=1, routed)	0.154	13.832		alu1/out[3]_INST_0_i_6_n_0
net (fo=1, routed)	0.000	8.780		register_file1/Read_data2[21]_INST_0_i_8_n_0
net (fo=1, routed)	0.000	13.956		alu1/out[3]_INST_0_i_2_n_0
net (fo=1, routed)	0.000	16.502		data_memory1/Read_data[14]_INST_0_i_112_n_0
net (fo=1, routed)	0.000	16.711		data_memory1/Read_data[14]_INST_0_i_51_n_0
net (fo=1, routed)	0.000	17.953		data_memory1/Read_data[14]_INST_0_i_5_n_0
LUT6 (Prop_lut6_I3_O)	(r) 0.319	17.953	Site: SLICE_X37Y40	data_memory1/Read_data[14]_INST_0_i_5/O
LUT6 (Prop_lut6_I2_O)	(r) 0.299	16.502	Site: SLICE_X46Y40	data_memory1/Read_data[14]_INST_0_i_112/O
LUT6 (Prop_lut6_I1_O)	(r) 0.299	19.925	Site: SLICE_X35Y65	data_memory1/Read_data[14]_INST_0/O
LUT6 (Prop_lut6_I1_O)	(r) 0.298	10.344	Site: SLICE_X30Y56	register_file1/Read_data2[21]_INST_0/O
MUXF7 (Prop...uxf7_I1_O)	(r) 0.245	9.025	Site: SLICE_X11Y66	register_file1/Read_data2[21]_INST_0_i_2/O
MUXF7 (Prop...uxf7_I1_O)	(r) 0.217	14.173	Site: SLICE_X33Y54	alu1/out[3]_INST_0/O
MUXF7 (Prop...uxf7_I1_O)	(r) 0.217	18.170	Site: SLICE_X37Y40	data_memory1/Read_data[14]_INST_0_i_1/O
MUXF7 (Prop...uxf7_I0_O)	(r) 0.209	16.711	Site: SLICE_X46Y40	data_memory1/Read_data[14]_INST_0_i_51/O
LUT3 (Prop_lut3_I0_O)	(r) 0.124	6.462	Site: SLICE_X14Y57	instruction_memory1/g0_b9_comp/O
LUT6 (Prop_lut6_I4_O)	(r) 0.124	6.981	Site: SLICE_X13Y57	instruction_memory1/Instruction[16]_INST_0_comp/O

图 5: 关键路径 Data Path（部分截取）

行为级仿真结果如图 6所示，由于行为级仿真可以同时查看所有寄存器，因而只观察一轮。PC 与各寄存器的值符合第 2 节中的分析。这里需要注意的是，reset 信号为高电平相当于处

理器在执行第一条指令，只是寄存器堆和数据存储器会清空，因此观察到 reset 信号结束时刻开始的时钟周期已经是第二个周期。

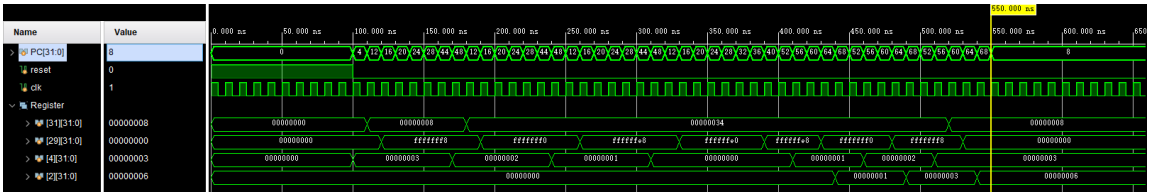

图 6: 行为级仿真

门级仿真，$a0、$v0、$sp、$ra 分别如图 7、8、9、10所示，符合分析。

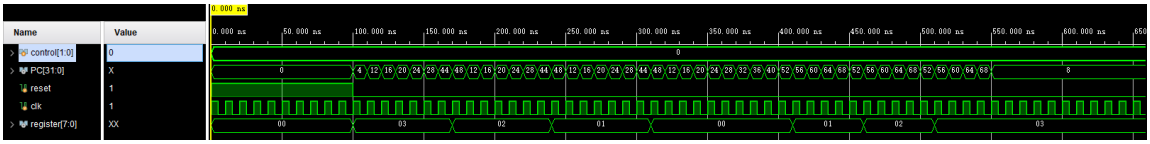

图 7: 门级仿真 $a0

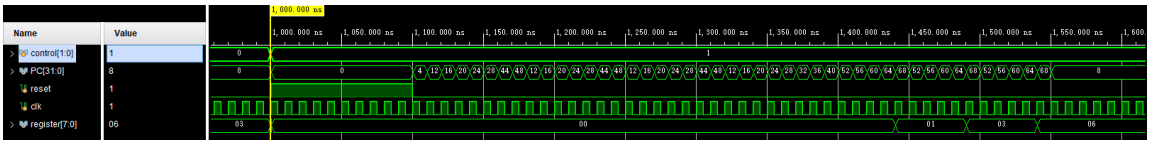

图 8: 门级仿真 $v0

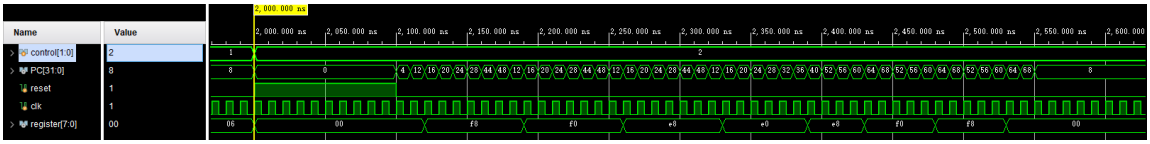

图 9: 门级仿真 $sp

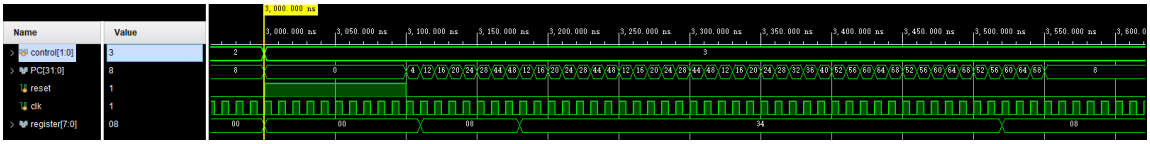

图 10: 门级仿真 $ra