浙江大学

# 本科实验报告

课程名称：计算机体系结构

姓　　名：吴同

学　　院：计算机科学与技术学院

　　系：计算机科学与技术系

专　　业：计算机科学与技术

学　　号：3170104848

指导教师：陈文智

2019 年　　9 月　　25 日

# 浙江大学实验报告


专业：计算机科学与技术
姓名：吴同
学号：3170104848
日期：2019 年 9 月 25 日
地点：曹西 301


课程名称：计算机体系结构　指导老师：陈文智　电子邮件：wutongcs@zju.edu.cn
实验名称：单周期 CPU 设计　实验类型：综合型　同组同学：徐欣苑

## 1 Experiment Purpose and Task

### 1.1 Experiment Purpose

- To understand the principles of single-cycle CPU controller and master methods of single-cycle CPU controller design.
- To understand the principles of datapath and master methods of datapath design.
- To understand the principles of single-cycle CPU and master methods of single-cycle CPU design.
- To master methods of program verification of CPU.

### 1.2 Experiment Task

- To design the CPU controller and datapath, and bring together the basic units into single-cycle CPU.
- To verify the single-cycle CPU with program and observe the execution of program.

## 2 Experiment Contents and Principles

### 2.1 Experiment Contents

- To complete the controller and datapath modules based on the given experimental materials.
- To synthesise the project and do simulation test.
- To download the binary file to the SWORD board for verification.

### 2.2 Experiment Principles

The organization is composed of a data meory, a instruction memory, and a single-cycle CPU with an ALU, 32 registers and a PC register.

The instruction set which is implemented in the single-cycle CPU:

| inst | 31…26 | 25…21 | 20…16 | 15…11 | 10…6 | 5…0 | operation |
|---|---|---|---|---|---|---|---|
| add | 000000 | rs | rt | rd | 00000 | 100000 | $rd = $rs + $rt; PC = PC + 4 |
| sub | | rs | rt | rd | 00000 | 100010 | $rd = $rs - $rt; PC = PC + 4 |
| and | | rs | rt | rd | 00000 | 100100 | $rd = $rs & $rt; PC = PC + 4 |
| or | | rs | rt | rd | 00000 | 100101 | $rd = $rs \| $rt; PC = PC + 4 |
| sll | | 00000 | rt | rd | sa | 000000 | $rd = $rt « $sa; PC = PC + 4 |
| srl | | 00000 | rt | rd | sa | 000010 | $rd = $rt » $sa; PC = PC + 4 |
| slt | | rs | rt | rd | 00000 | 101010 | $rd = ($rs < $rt) ? 1 : 0; PC = PC + 4 |
| jr | | rs | 00000 | 00000 | 00000 | 001000 | PC = $rs |
| addi | 001000 | rs | rt | imm | | | $rt = $rs + (signed_ext)imm; PC = PC + 4 |
| andi | 001100 | rs | rt | imm | | | $rt = $rs & (unsigned_ext)imm; PC = PC + 4 |
| ori | 001101 | rs | rt | imm | | | $rt = $rs \| (unsigned_ext)imm; PC = PC + 4 |
| lw | 100011 | rs | rt | imm | | | $rt = memory[$rs + (signed_ext)imm]; PC = PC + 4 |
| sw | 101011 | rs | rt | imm | | | memory[$rs + (signed_ext)imm] = $rt; PC = PC + 4 |
| beq | 000100 | rs | rt | imm | | | PC += ($rs == $rt) ? (4 + (signed_ext)imm«2) : 4 |
| bne | 000101 | rs | rt | imm | | | PC += ($rs != $rt) ? (4 + (signed_ext)imm«2) : 4 |
| j | 000010 | addr | | | | | PC = (PC+4)[31:28], addr«2 |
| jal | 000011 | addr | | | | | PC = (PC+4)[31:28], addr«2; $(31) = PC + 4 |

The hardware implementation of single-cycle MIPS core:

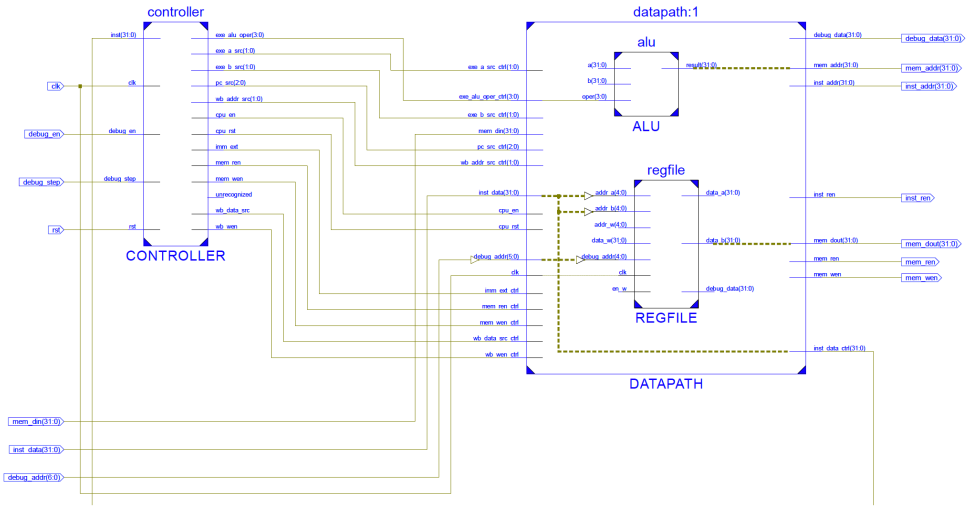


Figure 1: RTL diagram of single-cycle MIPS core

## 3 Experiment Procedure and Result

### 3.1 Procedure 1: completing the controller and datapath modules

In this procedure, I have completed the Verilog HDL codes in **controller.v** and **datapath.v**. The modified codes in **controller.v**:

```
R_FUNC_SUB: begin
   exe_alu_oper = EXE_ALU_SUB;
   wb_addr_src = WB_ADDR_RD;
   wb_data_src = WB_DATA_ALU;
   wb_wen = 1;
end
INST_J: begin
   pc_src = PC_JUMP;
end
INST_JAL: begin
   pc_src = PC_JUMP;
   exe_a_src = EXE_A_LINK;
   exe_b_src = EXE_B_LINK;
   exe_alu_oper = EXE_ALU_ADD;
   wb_addr_src = WB_ADDR_LINK;
   wb_data_src = WB_DATA_ALU;
   wb_wen = 1;
end
INST_BEQ: begin
   pc_src = PC_BEQ;
   exe_a_src = EXE_A_BRANCH;
   exe_b_src = EXE_B_BRANCH;
   exe_alu_oper = EXE_ALU_ADD;
   imm_ext = 1;
end
INST_BNE: begin
   pc_src = PC_BNE;
   exe_a_src = EXE_A_BRANCH;
   exe_b_src = EXE_B_BRANCH;
   exe_alu_oper = EXE_ALU_ADD;
   imm_ext = 1;
end
INST_ANDI: begin
   imm_ext = 0;
   exe_b_src = EXE_B_IMM;
   exe_alu_oper = EXE_ALU_AND;
   wb_addr_src = WB_ADDR_RT;
   wb_data_src = WB_DATA_ALU;
   wb_wen = 1;
end
INST_LW: begin
   imm_ext = 1;
   exe_b_src = EXE_B_IMM;
   exe_alu_oper = EXE_ALU_ADD;
   mem_ren = 1;
   wb_addr_src = WB_ADDR_RT;
   wb_data_src = WB_DATA_MEM;
   wb_wen = 1;
end
```

```
INST_SW: begin
   imm_ext = 1;
   exe_b_src = EXE_B_IMM;
   exe_alu_oper = EXE_ALU_ADD;
   mem_wen = 1;
end
```

The modified codes in **datapath.v**:

```
always @(posedge clk) begin
   if (cpu_rst) begin
      inst_addr <= 0;
   end
   else if (cpu_en) begin
      case (pc_src_ctrl)
         PC_JUMP: inst_addr <=
            {inst_addr_next[31:28],inst_data[25:0],2'b0};
         PC_JR: inst_addr <= data_rs;
         PC_BEQ: inst_addr <= rs_rt_equal ? alu_out : inst_addr_next;
         PC_BNE: inst_addr <= rs_rt_equal ? inst_addr_next : alu_out;
         default: inst_addr <= inst_addr_next;
      endcase
   end
end
always @(*) begin
   regw_addr = inst_data[15:11];
   case (wb_addr_src_ctrl)
      WB_ADDR_RD: regw_addr = addr_rd;
      WB_ADDR_RT: regw_addr = addr_rt;
      WB_ADDR_LINK: regw_addr = GPR_RA;
   endcase
end
always @(*) begin
   opa = data_rs;
   opb = data_rt;
   case (exe_a_src_ctrl)
      EXE_A_RS: opa = data_rs;
      EXE_A_LINK: opa = inst_addr_next;
      EXE_A_BRANCH: opa = inst_addr_next;
   endcase
   case (exe_b_src_ctrl)
      EXE_B_RT: opb = data_rt;
      EXE_B_IMM: opb = data_imm;
      EXE_B_LINK: opb = 32'h0;
      EXE_B_BRANCH: opb = {data_imm[29:0],2'b0};
   endcase
end
always @(*) begin
   regw_data = alu_out;
   case (wb_data_src_ctrl)
```

```
        WB_DATA_ALU: regw_data = alu_out;
        WB_DATA_MEM: regw_data = mem_din;
    endcase
end
```

### 3.2 Procedure 2: simluating with the MIPS core

In this procedure, I have done simulating with the completed MIPS core. The signals and register data were checked with the standard simulation result in the experiment guidelines.

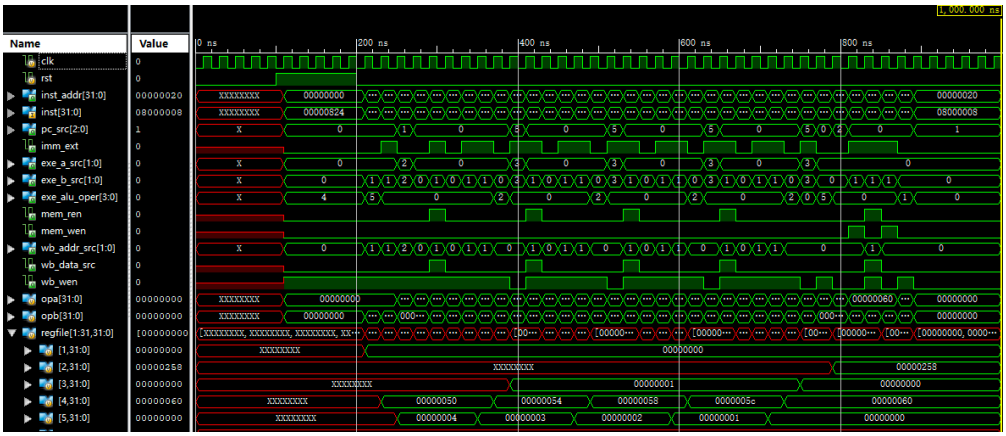


Figure 2: result of simulating for 1000 ns

### 3.3 Procedure 3: downloading to FPGA board for physical verification

After the simulation procedure, I built the whole ISE project, and downloaded the binary file to the FPGA board. Under debugging mode, the CPU can run step by step. The debugging data is in correspond with the simulation result.


Figure 3: single-step debugging mode

### 3.4 Data for running test program

The details of running test program:

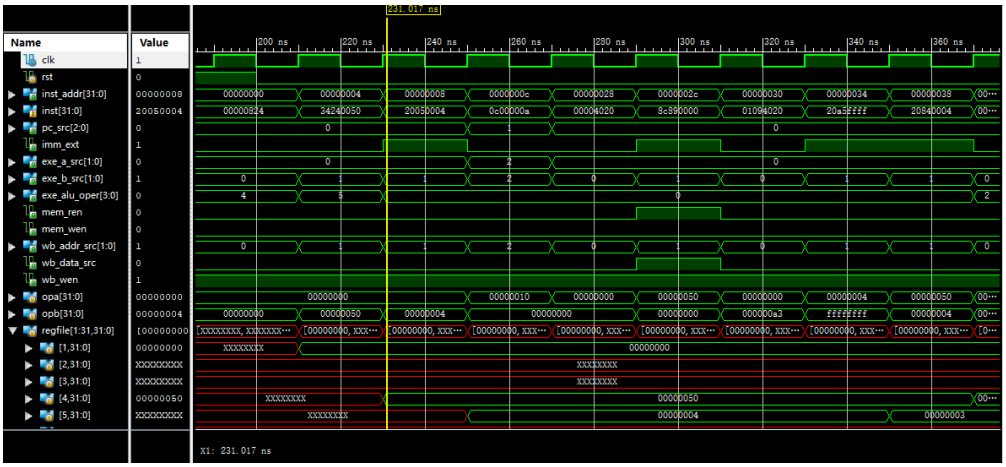

Figure 4: running detail 1

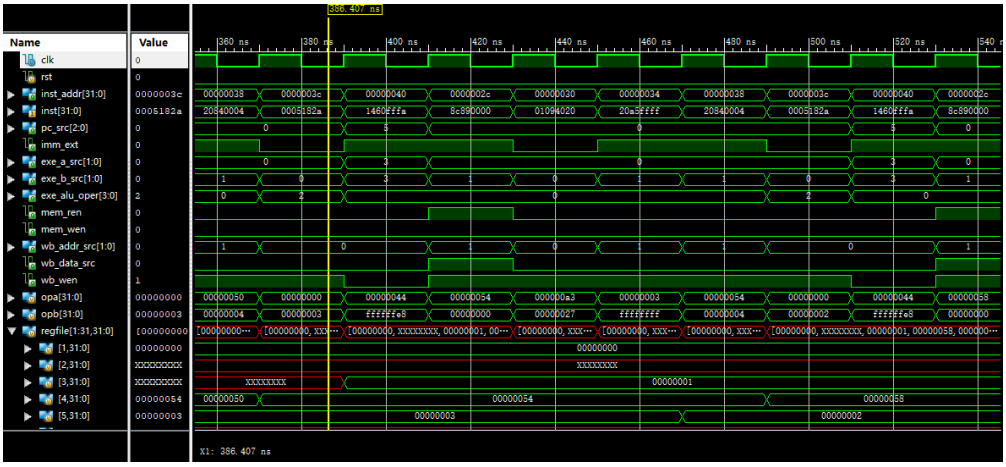

Figure 5: running detail 2

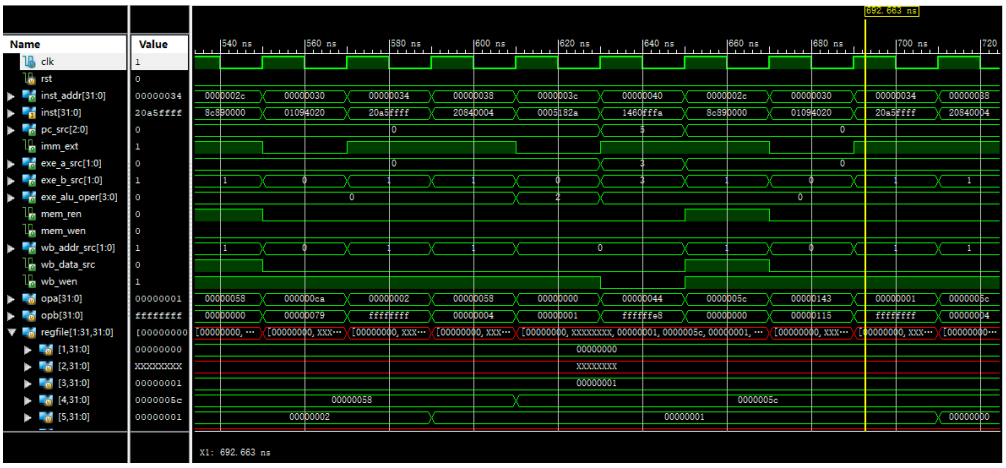

Figure 6: running detail 3

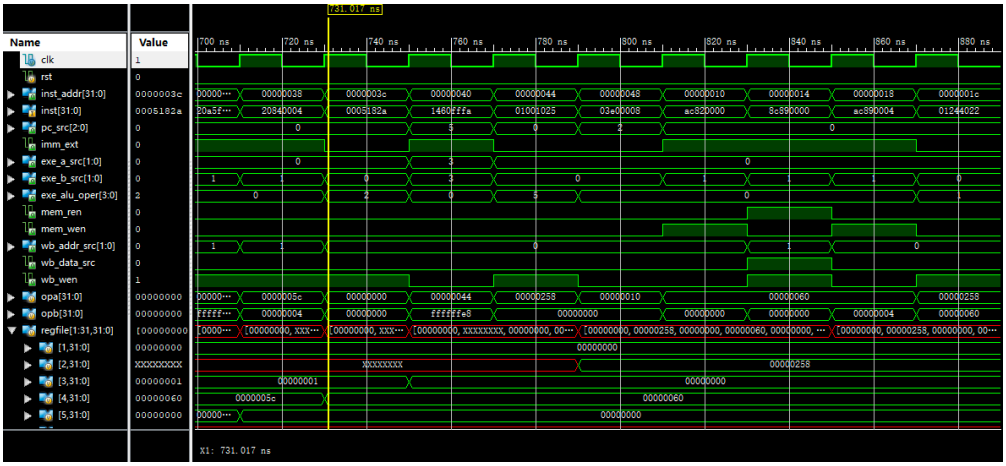

Figure 7: running detail 4

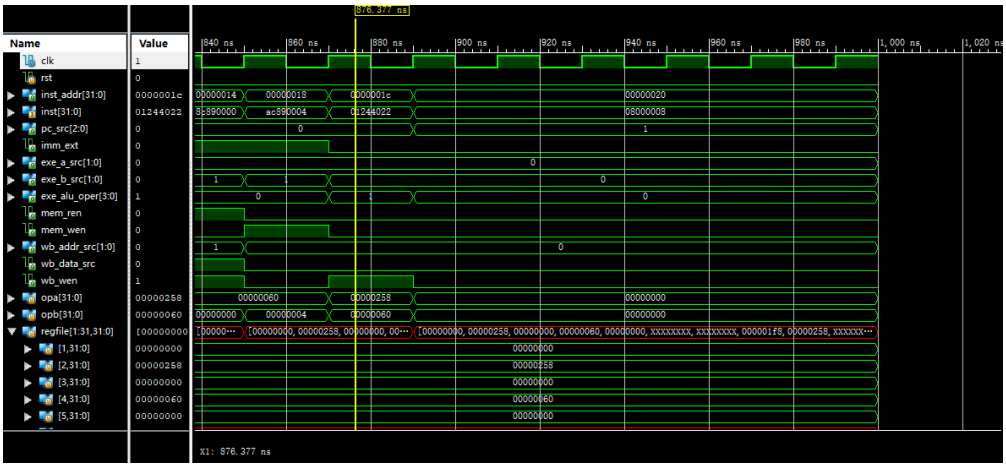

Figure 8: running detail 5

## 4 Analysis of Experiment Result

Now I do a step-by-step analysis of the execution of the test program.

| addr | inst | | ALUsrcA | ALUsrcB | ALUopt | Controller and PC | RegFile |
|---|---|---|---|---|---|---|---|
| 0x00000000 | 0x00000824 | and $1, $0, $0 | $rs | $rt | AND | enable write register<br>write data from ALU to $rd<br>PC = 0x00000004 | $1 = 0 |
| 0x00000004 | 0x34240050 | ori $4, $1, 80 | $rs | (u)imm | OR | enable write register<br>write data from ALU to $rt<br>PC = 0x00000008 | $4 = 80 |
| 0x00000008 | 0x20050004 | addi $5, $0, 4 | $rs | imm | ADD | enable write register<br>write data from ALU to $rt<br>PC = 0x0000000C | $5 = 4 |
| 0x0000000C | 0x0C00000A | jal 10 | PC+4 | 0 | ADD | enable write register<br>write data from ALU to $31<br>PC = 0x00000028 | $31 = 16 |
| 0x00000028 | 0x00004020 | add $8, $0, $0 | $rs | $rt | ADD | enable write register<br>write data from ALU to $rd<br>PC = 0x0000002C | $8 = 0 |
| 0x0000002C | 0x8C890000 | lw $9, 0($4) | $rs | imm | ADD | enable write register<br>write data from MEM to $rt<br>PC = 0x00000030 | $9 = 163 |
| 0x00000030 | 0x01094020 | add $8, $8, $9 | $rs | $rt | ADD | enable write register<br>write data from ALU to $rd<br>PC = 0x00000034 | $8 = 163 |
| 0x00000034 | 0x20A5FFFF | addi $5, $5, -1 | $rs | imm | ADD | enable write register<br>write data from ALU to $rt<br>PC = 0x00000038 | $5 = 3 |
| 0x00000038 | 0x20840004 | addi $4, $4, 4 | $rs | imm | ADD | enable write register<br>write data from ALU to $rt<br>PC = 0x0000003C | $4 = 84 |
| 0x0000003C | 0x0005182A | slt $3, $0, $5 | $rs | $rt | SLT | enable write register<br>write data from ALU to $rd<br>PC = 0x00000040 | $3 = 1 |
| 0x00000040 | 0x1460FFFA | bne $3, $0, -6 | PC + 4 | imm«2 | ADD | PC = ALU = 0x0000002C | |
| 0x0000002C | 0x8C890000 | lw $9, 0($4) | $rs | imm | ADD | enable write register<br>write data from MEM to $rt<br>PC = 0x00000030 | $9 = 39 |
| 0x00000030 | 0x01094020 | add $8, $8, $9 | $rs | $rt | ADD | enable write register<br>write data from ALU to $rd<br>PC = 0x00000034 | $8 = 202 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0x00000034 | 0x20A5FFFF | addi $5, $5, -1 | $rs | imm | ADD | enable write register<br>write data from ALU to $rt<br>PC = 0x00000038 | $5 = 2 |
| 0x00000038 | 0x20840004 | addi $4, $4, 4 | $rs | imm | ADD | enable write register<br>write data from ALU to $rt<br>PC = 0x0000003C | $4 = 88 |
| 0x0000003C | 0x0005182A | slt $3, $0, $5 | $rs | $rt | SLT | enable write register<br>write data from ALU to $rd<br>PC = 0x00000040 | $3 = 1 |
| 0x00000040 | 0x1460FFFA | bne $3, $0, -6 | PC + 4 | imm«2 | ADD | PC = ALU = 0x0000002C | |
| 0x0000002C | 0x8C890000 | lw $9, 0($4) | $rs | imm | ADD | enable write register<br>write data from MEM to $rt<br>PC = 0x00000030 | $9 = 121 |
| 0x00000030 | 0x01094020 | add $8, $8, $9 | $rs | $rt | ADD | enable write register<br>write data from ALU to $rd<br>PC = 0x00000034 | $8 = 323 |
| 0x00000034 | 0x20A5FFFF | addi $5, $5, -1 | $rs | imm | ADD | enable write register<br>write data from ALU to $rt<br>PC = 0x00000038 | $5 = 1 |
| 0x00000038 | 0x20840004 | addi $4, $4, 4 | $rs | imm | ADD | enable write register<br>write data from ALU to $rt<br>PC = 0x0000003C | $4 = 92 |
| 0x0000003C | 0x0005182A | slt $3, $0, $5 | $rs | $rt | SLT | enable write register<br>write data from ALU to $rd<br>PC = 0x00000040 | $3 = 1 |
| 0x00000040 | 0x1460FFFA | bne $3, $0, -6 | PC + 4 | imm«2 | ADD | PC = ALU = 0x0000002C | |
| 0x0000002C | 0x8C890000 | lw $9, 0($4) | $rs | imm | ADD | enable write register<br>write data from MEM to $rt<br>PC = 0x00000030 | $9 = 277 |
| 0x00000030 | 0x01094020 | add $8, $8, $9 | $rs | $rt | ADD | enable write register<br>write data from ALU to $rd<br>PC = 0x00000034 | $8 = 600 |
| 0x00000034 | 0x20A5FFFF | addi $5, $5, -1 | $rs | imm | ADD | enable write register<br>write data from ALU to $rt<br>PC = 0x00000038 | $5 = 0 |
| 0x00000038 | 0x20840004 | addi $4, $4, 4 | $rs | imm | ADD | enable write register<br>write data from ALU to $rt<br>PC = 0x0000003C | $4 = 96 |
| 0x0000003C | 0x0005182A | slt $3, $0, $5 | $rs | $rt | SLT | enable write register<br>write data from ALU to $rd<br>PC = 0x00000040 | $3 = 0 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0x00000040 | 0x1460FFFA | bne $3, $0, -6 | PC + 4 | imm«2 | ADD | PC = ALU = 0x00000044 | |
| 0x00000044 | 0x01001025 | or $2, $8, $0 | $rs | $rt | OR | enable write register<br>write data from ALU to $rd<br>PC = 0x00000048 | $2 = 600 |
| 0x00000048 | 0x03E00008 | jr $ra | | | | PC = 0x00000010 | |
| 0x00000010 | 0xAC820000 | sw $2, 0($4) | $rs | imm | ADD | enable write memory<br>write data from $rs to *96<br>PC = 0x00000014 | |
| 0x00000014 | 0x8C890000 | lw $9, 0($4) | $rs | imm | ADD | enable write register<br>write data from MEM to $rt<br>PC = 0x00000018 | $9 = 600 |
| 0x00000018 | 0xAC890004 | sw $9, 4($4) | $rs | imm | ADD | enable write memory<br>write data from $rs to *100<br>PC = 0x0000001C | |
| 0x0000001C | 0x01244022 | sub $8, $9, $4 | $rs | $rt | SUB | enable write register<br>write data from ALU to $rd<br>PC = 0x00000020 | $8 = 504 |
| 0x00000020 | 0x08000008 | j 32 | | | | PC = 0x00000020 | |

## 5 Discussion and Review

In this experiment, I have reviewed the knowledge of single-cycle CPU design which has been learned in the computer organization course. The framework of this experiment is much different from the framework used in the computer organization course. In this experiment, the main job is to analyze the code of the experimental framework and then complete the vacancy. Besides, I have added BNE instruction to ensure that the test program can run normally.

During the process of debugging on SWORD board, I met a fatal bug that there was no input signal into the display screen. I checked the VGA module and did much analysis, but found nothing wrong. In the end, I pressed the mode switch button on the display. This bug was caused by a wrong input mode of display, which was HDMI instead of VGA. Although this was a small bug, it had been bothering me for a long time. So it is important to be careful when doing experiment.