哈爾濱工業大學（深圳）
HARBIN INSTITUTE OF TECHNOLOGY

实验报告

开课学期：2020 春季

课程名称：计算机组成原理（实验）

实验名称：微程序控制器设计

实验性质：综合设计型

实验学时：4 地点：线上

学生班级：1801105

学生学号：180110527

学生姓名：李秋阳

评阅教师：

报告成绩：

实验与创新实践教育中心制

2020 年 5 月

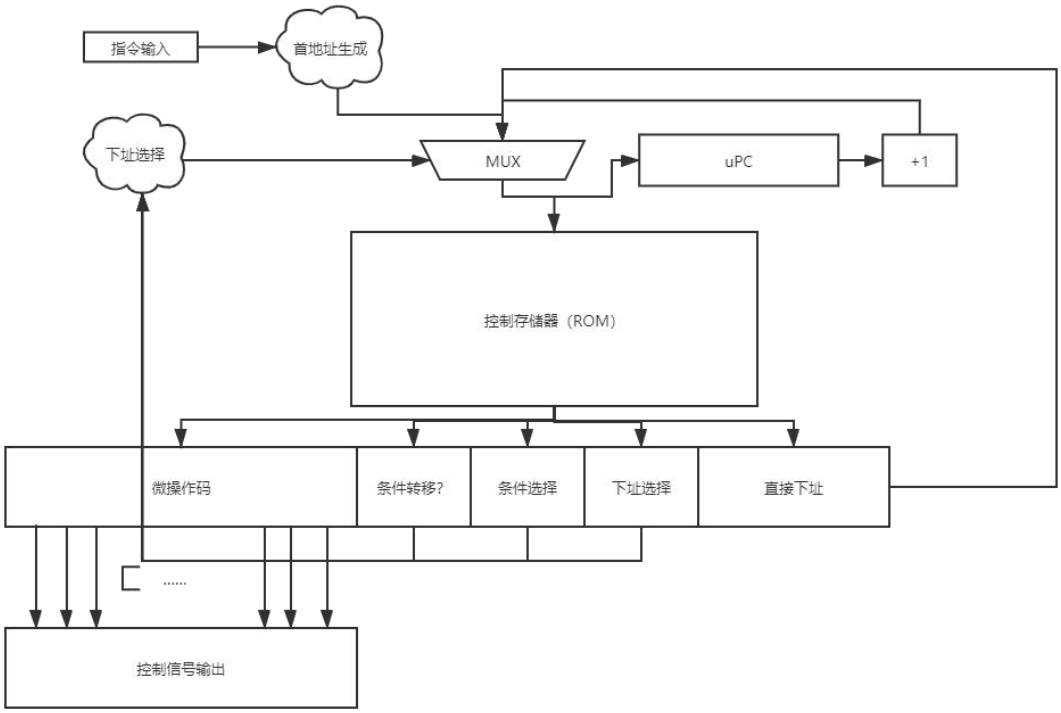
一、 实验项目

微程序控制器设计

二、 系统功能详细设计及实现

1. 对系统进行详细设计描述。用硬件框图描述系统主要功能及各模块之间的相互关系；（此部分内容为本报告重点内容，需详细描述）

微程序控制器硬件框图：



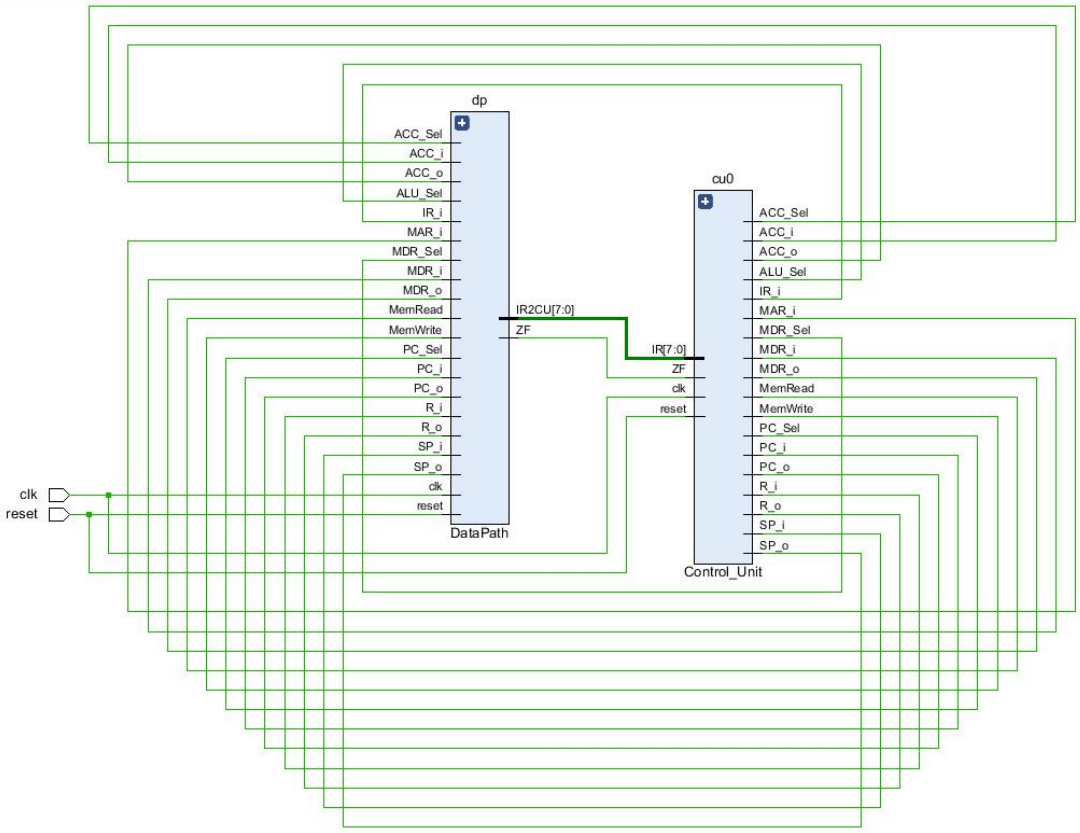
微程序控制器接收来自 IR 的指令，进行分析后给出相对应的微操作的信号。微程序存储在控制存储器中，微程序由一条条的微指令组成。取得微指令后通过分析取到的微指令以及接收到的 IR 指令来决定执行周期需要进行的操作。

由取到的指令，按以下条件进行下址选择——若指令仍在执行周期中，则下一条微指令的地址为当前微指令的地址+1（uPC+1），若执行周期结束，则下一条微指令则返回取指阶段，然后重复以上的过程，实现连续取指分析执行过程。

控制存储器存放了 29 条微指令，11 个微程序(其中有一个是 NOP)，微程序的入口地址分别为 0，4，7，12，16，17，18，19，22，25，28。uPC 记录了当前微指令地址，由此控制存储器可以给出微指令，通过微指令中的微操作码，输出当前操作需要的信号。

同时，由微指令中的下址选择、条件选择、条件跳转等字段，可以结合指令译码给出下一条指令的地址。

2. 模块描述。包括模块功能，输入、输出端口及变量含义，时序等。



微程序指令寄存器

```
reg [31:0] ucode_mem [255:0];
reg [255:0] uPC = 0;
reg [31:0] curr_code;
```

微程序入口地址定义

```
wire [7:0]addr = curr_code[7:0];
wire [1:0]nextAddrSel = curr_code[9:8];
wire condSel = curr_code[10];
wire condJMP = curr_code[11];
assign PC_i = curr_code[29];
assign IR_i = curr_code[28];
assign MAR_i = curr_code[27];
assign MDR_i = curr_code[26];
assign ACC_i = curr_code[25];
assign SP_i = curr_code[24];
assign R_i = curr_code[23];
assign PC_o = curr_code[22];
assign MDR_o = curr_code[21];
assign ACC_o = curr_code[20];
assign SP_o  = curr_code[19];
assign R_o = curr_code[18];
assign PC_Sel = curr_code[17];
assign MDR_Sel = curr_code[16];
assign ACC_Sel = curr_code[15];
assign ALU_Sel = curr_code[14];
assign MemRead = curr_code[13];
assign MemWrite = curr_code[12];
```

通过指令译码 IR 的高 4 位的值,以及下址选择来判断下地址

```
always@(*)
    begin
        curr_code = ucode_mem[uPC];
        case(IR[7:4])
            NOP:begin
                if(nextAddrSel == 2'b00)
                    begin
                        nextAddress <= uPC + 1;
                    end
                else if(nextAddrSel == 2'b01)
                    begin
                        nextAddress <= addr;
                    end
                else
                    begin
                        nextAddress <= 8'h1c;
                    end
            end
            LOAD:begin
                if(nextAddrSel == 2'b00)
                    begin
                        nextAddress <= uPC + 1;
                    end
                else if(nextAddrSel == 2'b01)
                    begin
                        nextAddress <= addr;
                    end
                else
                    begin
                        nextAddress <= 8'h7;
                    end
            end
```

```
STORE:begin
    if(nextAddrSel == 2'b00)
        begin
            nextAddress <= uPC + 1;
        end
    else if(nextAddrSel == 2'b01)
        begin
            nextAddress <= addr;
        end
    else
        begin
            nextAddress <= 8'hc;
        end
end
MOVE:begin
    if(nextAddrSel == 2'b00)
        begin
            nextAddress <= uPC + 1;
        end
    else if(nextAddrSel == 2'b01)
        begin
            nextAddress <= addr;
        end
    else
        begin
            nextAddress <= 8'h10;
        end
end
```

```
ADD:begin
    if(nextAddrSel == 2'b00)
        begin
            nextAddress <= uPC + 1;
        end
    else if(nextAddrSel == 2'b01)
        begin
            nextAddress <= addr;
        end
    else
        begin
            nextAddress <= 8'h11;
        end
end
AND:begin
    if(nextAddrSel == 2'b00)
        begin
            nextAddress <= uPC + 1;
        end
    else if(nextAddrSel == 2'b01)
        begin
            nextAddress <= addr;
        end
    else
        begin
            nextAddress <= 8'h12;
        end
end
```

```
JUMP:begin
    if(nextAddrSel == 2'b00)
        begin
            nextAddress <= uPC + 1;
        end
    else if(nextAddrSel == 2'b01)
        begin
            nextAddress <= addr;
        end
    else
        begin
            nextAddress <= 8'h13;
        end
end
JUMPZ:begin
    if(nextAddrSel == 2'b00)
        begin
            nextAddress <= uPC + 1;
        end
    else if(nextAddrSel == 2'b01)
        begin
            nextAddress <= addr;
        end
    else
        begin
            nextAddress <= 8'h16;
        end
end
```

```
JUMPNZ:begin
    if(nextAddrSel == 2'b00)
        begin
            nextAddress <= uPC + 1;
        end
    else if(nextAddrSel == 2'b01)
        begin
            nextAddress <= addr;
        end
    else
        begin
            nextAddress <= 8'h19;
        end
end
LOADR:begin
    if(nextAddrSel == 2'b00)
        begin
            nextAddress <= uPC + 1;
        end
    else if(nextAddrSel == 2'b01)
        begin
            nextAddress <= addr;
        end
    else
        begin
            nextAddress <= 8'h4;
        end
end
```

```
        default:begin
            if(nextAddrSel == 2'b00)
                begin
                    nextAddress <= uPC + 1;
                end
            else if(nextAddrSel == 2'b01)
                begin
                    nextAddress <= addr;
                end
            else
                begin
                    nextAddress <= 8'h0;
                end
        end
    endcase
end
```

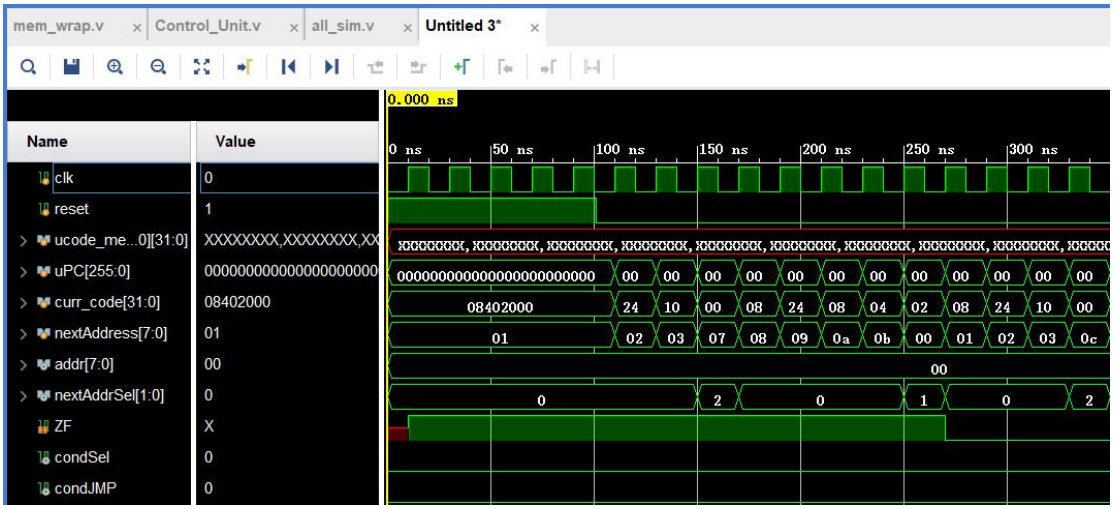
由微指令中的下址选择、条件选择、条件跳转等字段，改变 uPC 的值，维持一时钟周期的来自控制存储器的微指令输出，可以结合指令译码给出下一条指令的地址。

```
    always@(posedge clk)
        begin
            if(reset)
                begin
                    uPC <= addr;
                end
            else if(condJMP == 1 && condSel == 0 && ZF == 0)
                begin
                    uPC <= addr;
                end
            else if(condJMP == 1 && condSel == 1 && ZF == 1)
                begin
                    uPC <= addr;
                end
            else
                begin
                    uPC <= nextAddress;
                end
        end

endmodule
```

三、 调试报告

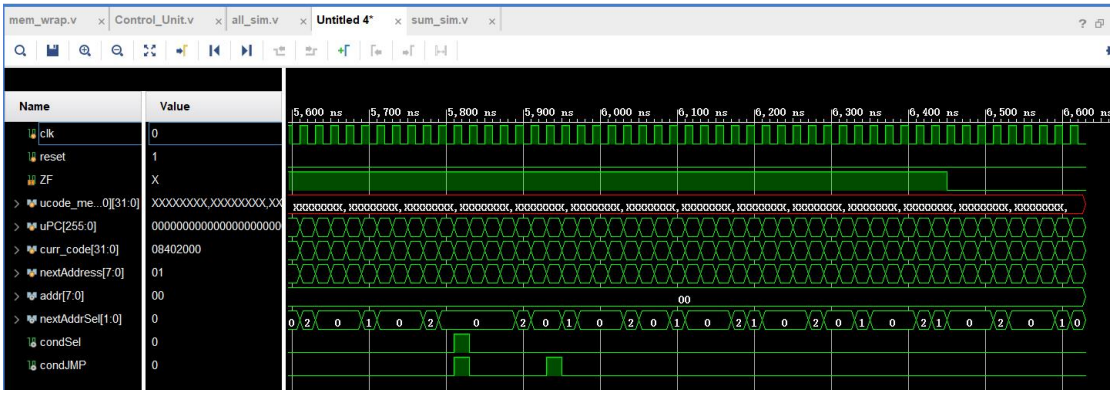
出现的问题、分析原因及解决方案。需要有问题截图，仿真截图。




```
PC: 00001011
 IR: 11111111
 MAR: 00001010
 MDR: xxxxxxxx
 ACC: 11111111
 SP: 00000000
 R: 00000000
 Bus: xxxxxxxx

relaunch_sim: Time (s): cpu = 00:00:01 ; elapsed = 00:00:05 . Memory (MB): peak = 847.082 ; gain = 0.000
```

代码完成之后分别测试无问题，进行完整测试。



```
PC: 00100011
 IR: 00110000
 MAR: 00100010
 MDR: 00100101
 ACC: 11111111
 SP: 00000000
 R: 11111111
 Bus: 00100101

PC: 00100011
 IR: 00110000
 MAR: 00100101
 MDR: 00100101
 ACC: 11111111
 SP: 00000000
 R: 11111111
 Bus: 11111111

PC: 00100011
 IR: 00110000
 MAR: 00100101
 MDR: 11111111
 ACC: 11111111
 SP: 00000000
 R: 11111111
 Bus: 00100011

Gooooooooooooooooooood! Finally Here! 4+3+2+1= 10
```

一次通过，程序无 bug。随后检查是否存在不符合 Verilog 编程规范语句（实验 3 在 always @(posedge clk)里面用阻塞赋值导致提交失败），未发现。提交实验结果，成功。

四、 总结及实验课程感想

计算机组成原理的实验比起上学期同样使用 Verilog 的数字逻辑实验来说没有太多烦琐的东西，都是直接用代码实现组成原理理论课程中的一些东西，没有了上学期写数字逻辑实验时那种课程与实验关联性差的错觉，有一种学以致用的感觉。

除了在写实验 1 时因为忘记了 Verilog 编程的一些规则和对硬件描述语言进行运用的理解还不深导致做了很久，找了很长时间 bug 也找不出来以外，余下 3 个实验还是相对顺利。特别是到了第四次实验，有了前三次的经验，一晚上就搞定了理解题意、编写代码到完成调试的过程，还是说明自己有了不小的收获。

特别感谢老师和助教不仅精心设计了实验指导书，搭建了线上 OJ 平台，提供了代码框架省去我们很多精力，实验过程中的解答也十分细心，极大提升了实验中的个人体验。