中国科学技术大学计算机学院
《计算机组成原理实验》报告

实验题目：COD 综合设计

学生姓名：黄博韬_____

学生学号：PB20071414__

完成日期：2022.5.20____

计算机实验教学中心制

2020 年 09 月

一. 实验实现内容

在流水线 CPU 的基础上实现 cache，分支预测，指令集扩展，并设计了 VGA，利用 CPU+程序实现颜色渐变输出画图。

二. Cache 和 mem 详解

```
module cache(
input clk,
input rst,
//cpu<->cache
input [7:0]cpu_req_addr,
input cpu_req_rw,
input cpu_req_valid,
input [31:0]cpu_data_write,
output reg [31:0]cpu_data_read,
output reg cpu_ready,
//cache<->memory
output reg [7:0]mem_req_addr,
output reg mem_req_rw,
output reg mem_req_valid,
output reg [127:0]mem_data_write,
input [127:0]mem_data_read,
input mem_ready
);

parameter V = 133;
parameter D = 132;
parameter TagMSB = 131;
parameter TagLSB = 128;
parameter BlockMSB = 127;
parameter BlockLSB = 0 ;

parameter IDLE=0;
parameter CompareTag=1;
parameter Allocate=2;
parameter WriteBack=3;
parameter RST=4;


reg [133:0] cache_data[0:3];
//133:V,132:D,[131:128]:TAG,[127:0]DATA
reg [2:0]state,next_state;
wire hit;

wire [1:0]cpu_req_index;
wire [3:0]cpu_req_tag;
wire [1:0]cpu_req_offset;
```

```
42.
43. assign cpu_req_offset=cpu_req_addr[1:0];
44. assign cpu_req_index=cpu_req_addr[3:2];
45. assign cpu_req_tag=cpu_req_addr[7:4];
46.
47. integer i;
48. //初始化 cache
49. initial
50. begin
51.     for(i=0;i<3;i=i+1)
52.         cache_data[i]=0;
53. end
54.
55. always@(posedge clk,posedge rst)
56. if(rst)
57. begin
58.     state<=RST;
59. end
60. else
61.     state<=next_state;
62. //
63. always@(*)
64. case(state)
65.     RST:next_state=IDLE;
66.     IDLE:if(cpu_req_valid)
67.             next_state=CompareTag;
68.          else
69.             next_state=IDLE;
70.     CompareTag:if(hit)
71.                     next_state=IDLE;
72.                else if(cache_data[cpu_req_index][V:D]==2'b11)
73.                //if the block is valid and dirty then go to WriteBack
74.                     next_state=WriteBack;
75.                else
76.                     next_state=Allocate;
77.     Allocate:if(mem_ready)
78.                     next_state=CompareTag;
79.              else
80.                     next_state=Allocate;
81.     WriteBack:if(mem_ready)
82.                     next_state=Allocate;
83.               else
84.                     next_state=WriteBack;
85.       default:next_state=IDLE;
86. endcase
87.
88. assign hit = cache_data[cpu_req_index][V]&(cache_data[cpu_req_index][TagMSB:TagLSB]==cpu_re
    q_tag)? 1:0;
89. always@(posedge clk)
```

```
90. if(state==RST)
91.     for(i=0;i<3;i=i+1)
92.         cache_data[i]<=0;
93. else if(state==IDLE)
94.     begin
95.     cpu_ready<=1'b0;
96.     mem_req_valid=1'b0;
97.     end
98. else if(state==CompareTag)
99.     if(hit)
100.          if(cpu_req_rw==1'b0)              //read hit
101.          begin
102.              cpu_ready<=1'b1;
103.              cpu_data_read<=cache_data[cpu_req_index][cpu_req_offset*32 +:32];
104.          end
105.          else                               //write hit,置脏位为 1
106.          begin
107.              cpu_ready<=1'b1;
108.              cache_data[cpu_req_index][cpu_req_offset*32 +:32]=cpu_data_write;
109.              cache_data[cpu_req_index][D]=1'b1;
110.          end
111.      else
112.          cpu_ready<=1'b0;
113.  else if(state==Allocate)                 //read new block from memory to cache
114.      if(!mem_ready)
115.      begin
116.          mem_req_addr<={cpu_req_addr[7:2],2'b00};
117.          mem_req_rw<=1'b0;
118.          mem_req_valid<=1'b1;
119.      end
120.      else
121.      begin
122.          mem_req_valid<=1'b0;
123.          cache_data[cpu_req_index][BlockMSB:BlockLSB]<=mem_data_read;
124.          cache_data[cpu_req_index][V:D]<=2'b10;
125.          cache_data[cpu_req_index][TagMSB:TagLSB]<=cpu_req_tag;
126.      end
127.  else if(state==WriteBack)                        //write dirty block to memory
128.      if(!mem_ready)
129.      begin
130.          mem_req_addr<={cache_data[cpu_req_index][TagMSB:TagLSB],cpu_req_index,2'b00};
131.          mem_req_rw<=1'b1;
132.          mem_data_write<=cache_data[cpu_req_index][BlockMSB:BlockLSB];
133.          mem_req_valid<=1'b1;
134.      end
135.      else
136.      begin
137.          mem_req_valid<=1'b0;
138.      end
```

```
139.   else
140.   begin
141.       cpu_ready<=1'b0;
142.       mem_req_valid=1'b0;
143.   end
144.   endmodule
```

四行，每行 4 字直接映射 cache。

三段式状态机，5 个状态 IDLE、CompareTag、Allocate、WriteBack、RST，按下 rst 后，进入 RST 态，把 cache 所有行清空，之后进入初态 IDLE，如果 cpu 请求有效，进入 CompareTag 态，如果 hit,再回到 IDLE 态，如果对应位置有脏块，则进入 Allocate 态，否则进入 WriteBack 态。在 Allocate 态，如果 mem_ready，则进入 CompareTag 态。在 WriteBack 态，如果 mem_ready，则进入 WriteBack 态。

在 RST 态，把 cache 清空。在 CompareTag 段，如果 hit，输出 cpu_ready=1，根据是读还是写，把 cache 对应值输出或者把对应值写入，写入时置脏位。如果没有 hit，则 cpu_ready=0。在 Allocate 态，如果 mem_ready=0，则保持 mem_req_valid=1，mem_req_rw=0(代表读)，将对应地址输入 mem，否则 mem_req_valid=0,并将数据送入 cache。在 WriteBack 段,如果 mem_ready=0，则保持 mem_req_valid=1，mem_req_rw（代表写）=1，否则 mem_req_valid=0。

```
1.  module mem(
2.  input clk,
3.  input rst,
4.  input [7:0]mem_req_addr,                    //[1:0]块内地址
5.  input mem_req_rw,
6.  input mem_req_valid,
7.  input [127:0]mem_data_write,
8.  output reg [127:0]mem_data_read,
9.  output reg mem_ready,
10. input [7:0]mem_req_addr2,
11. output [31:0]data
12.     );
13.
14. reg [31:0] mem [0:255];                 //256 个字，64 个块
15. assign data = mem[mem_req_addr2];
16. integer i;
17. initial
```

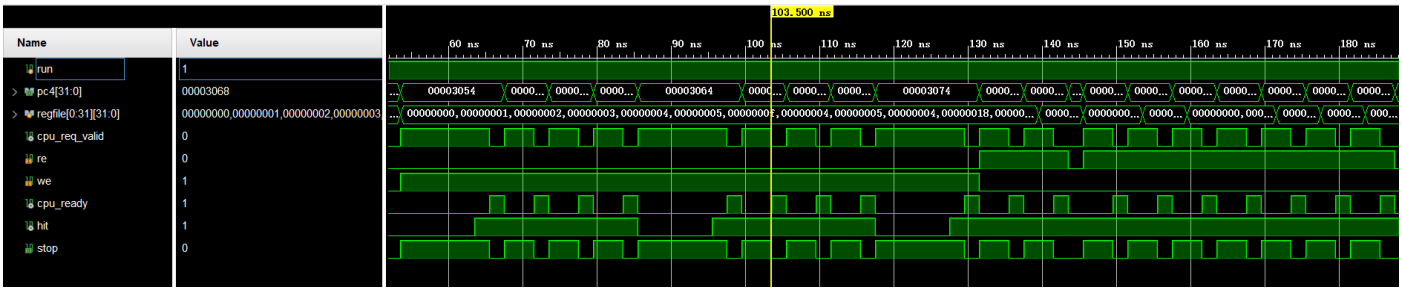
```
begin
    for(i=0;i<256;i=i+1)
        mem[i]=32'd0;
end

always@(posedge clk,posedge rst)
if(rst)begin
    mem_ready<=1'b0;
    mem_data_read<=0;
end
else if(mem_req_valid&&mem_req_rw==1'b1)                    //write
begin
    mem[mem_req_addr+3]<=mem_data_write[127:96];
    mem[mem_req_addr+2]<=mem_data_write[95:64];
    mem[mem_req_addr+1]<=mem_data_write[63:32];
    mem[mem_req_addr]<=mem_data_write[31:0];
    mem_ready<=1'b1;
end
else if(mem_req_valid&&mem_req_rw==1'b0)                    //read
begin
    mem_data_read<={mem[mem_req_addr+3],mem[mem_req_addr+2],mem[mem_req_addr+1],mem[mem_req
_addr]};
    mem_ready<=1'b1;
end
else
    mem_ready<=1'b0;

endmodule
```

当 mem_req_vaild=1 时，当四个字一起读出/写入，完成后把 mem_ready 置为 1，完成握手。

具体代码仿真和汇编如图，



```
sw x1 0(zero)
sw x2 4(zero)
sw x3 8(zero)
sw x4 12(zero)
sw x5 16(zero)
sw x6 20(zero)
sw x7 24(zero)
sw x8 28(zero)
sw x9 32(zero)
lw x9 0(zero)
lw x8 4(zero)
lw x7 8(zero)
lw x6 12(zero)
lw x5 16(zero)
lw x4 20(zero)
lw x3 24(zero)
lw x2 28(zero)
lw x1 32(zero)
```

如果没有 hit，stop 信号保持较长时间（只有写分配 3 clk，写分配加写返回 96clk），如果 hit，stop 信号（1 clk）。一旦 cpu_ready=1（cache 完成数据读出/写入），stop=0，CPU 继续运行，否则等待 cache 写分配以及可能的写返回。

三. 分支预测（BHT）

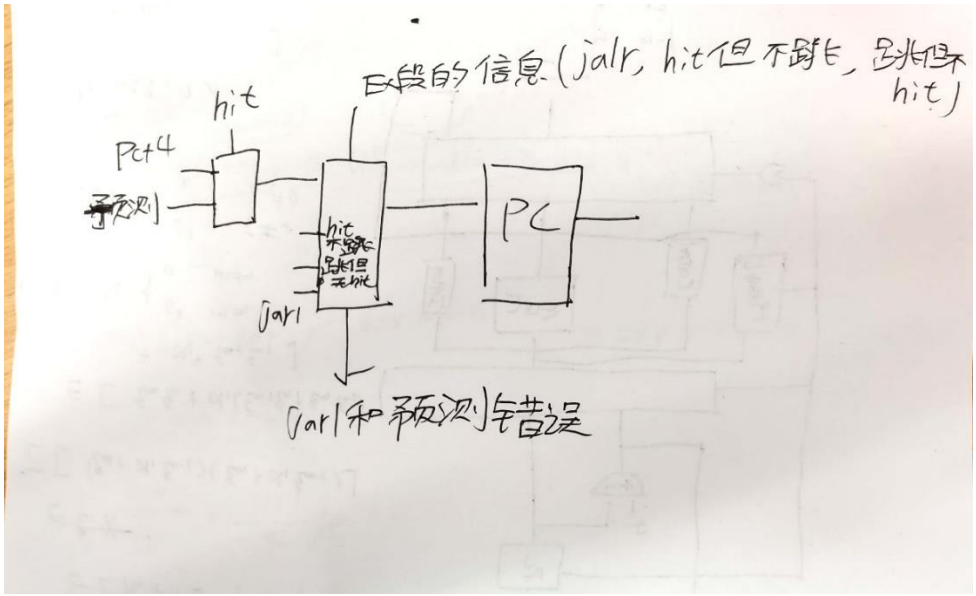
```
module BHT(
input clk,
input rst,
input [31:0] pc,
input [31:0] brpc_update,
input [31:0] prepc_update,
input is_br_jal,
input really,
output reg [31:0] pcout,
output hit);
reg [1:0] ptr;
reg [31:0] brpc[0:3];
reg [31:0] prepc[0:3];
reg [1:0] state[0:3];
integer i;
initial
begin
   ptr=0;
   for(i=0;i<4;i=i+1)
```

```
20.    begin
21.       brpc[i]=0;
22.       prepc[i]=0;
23.       state[i]=0;
24.    end
25. end
26. assign hit=((pc==brpc[0])&state[0][1])|((pc==brpc[1])&state[1][1])|((pc==brpc[2])&state[2][
    1])|((pc==brpc[3])&state[3][1]);
27. always@(*)
28. begin
29.    if((pc==brpc[0]) && state[0][1])
30.       pcout=prepc[0];
31.    else if((pc==brpc[1]) && state[1][1])
32.       pcout=prepc[1];
33.    else if((pc==brpc[2]) && state[2][1])
34.       pcout=prepc[2];
35.    else if((pc==brpc[3]) && state[3][1])
36.       pcout=prepc[3];
37.    else
38.       pcout=pc;
39. end
40. always@(posedge clk or posedge rst)
41. begin
42.    if(rst)
43.    begin
44.       ptr<=0;
45.       for(i=0;i<4;i=i+1)
46.       begin
47.          brpc[i]<=0;
48.          prepc[i]<=0;
49.          state[i]<=0;
50.       end
51.    end
52.    else if(is_br_jal)
53.    begin
54.       if(really)
55.       begin
56.          if(brpc_update==brpc[0])
57.          begin
58.             case(state[0])
59.             2'b00:state[0]<=2'b01;
60.             2'b01:state[0]<=2'b11;
61.             2'b10:state[0]<=2'b11;
62.             2'b11:state[0]<=2'b11;
63.             endcase
64.          end
65.          else if(brpc_update==brpc[1])
66.          begin
67.             case(state[1])
```

```
68.             2'b00:state[1]<=2'b01;
69.             2'b01:state[1]<=2'b11;
70.             2'b10:state[1]<=2'b11;
71.             2'b11:state[1]<=2'b11;
72.             endcase
73.          end
74.          else if(brpc_update==brpc[2])
75.          begin
76.             case(state[2])
77.             2'b00:state[2]<=2'b01;
78.             2'b01:state[2]<=2'b11;
79.             2'b10:state[2]<=2'b11;
80.             2'b11:state[2]<=2'b11;
81.             endcase
82.          end
83.          else if(brpc_update==brpc[3])
84.          begin
85.             case(state[3])
86.             2'b00:state[3]<=2'b01;
87.             2'b01:state[3]<=2'b11;
88.             2'b10:state[3]<=2'b11;
89.             2'b11:state[3]<=2'b11;
90.             endcase
91.          end
92.          else
93.          begin
94.             brpc[ptr]<=brpc_update;
95.             prepc[ptr]<=prepc_update;
96.             state[ptr]<=2'b01;
97.             ptr<=ptr+2'b1;
98.          end
99.       end
100.         else
101.         begin
102.            if(brpc_update==brpc[0])
103.            begin
104.               case(state[0])
105.               2'b00:state[0]<=2'b00;
106.               2'b01:state[0]<=2'b00;
107.               2'b10:state[0]<=2'b00;
108.               2'b11:state[0]<=2'b10;
109.               endcase
110.           end
111.           else if(brpc_update==brpc[1])
112.           begin
113.              case(state[1])
114.              2'b00:state[1]<=2'b00;
115.              2'b01:state[1]<=2'b00;
116.              2'b10:state[1]<=2'b00;
```

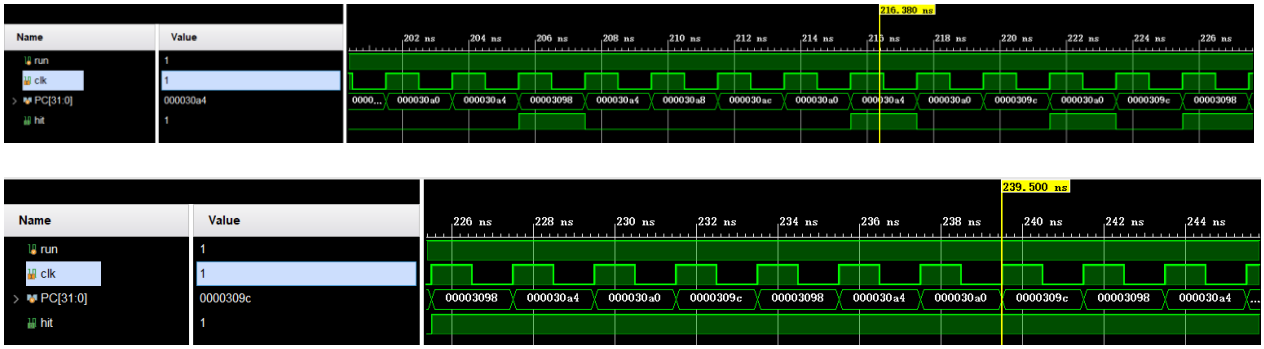
```
                2'b11:state[1]<=2'b10;
                endcase
            end
            else if(brpc_update==brpc[2])
            begin
                case(state[2])
                2'b00:state[2]<=2'b00;
                2'b01:state[2]<=2'b00;
                2'b10:state[2]<=2'b00;
                2'b11:state[2]<=2'b10;
                endcase
            end
            else if(brpc_update==brpc[3])
            begin
                case(state[3])
                2'b00:state[3]<=2'b00;
                2'b01:state[3]<=2'b00;
                2'b10:state[3]<=2'b00;
                2'b11:state[3]<=2'b10;
                endcase
            end
        end
    end
end
endmodule
```

2bit 动态分支预测(beq,blt,jal)，在 IF 段进行预测，输入当下 PC 值，输出预测 PC 值和 hit，hit 在模块外面选择 PC+4 和预测值，EX 段进行状态转移和更新内部预测表，如果 EX 段发现是跳转指令，并跳转，那么就将该指令的 PC 值和跳转目的地写入，或者（如果已经有对应 PC 和跳转目的地时）对应状态增加，如果是跳转指令，但不跳转，那么不写入或者（如果已经有对应 PC 和跳转目的地时）对应状态减少。输出的预测 PC 值为状态>=2（2’b10）且 PC 撞上，对应预测 PC 值。在模块外面,pcin 到 PC 输入有一个 2 选 1 寄存器和 4 选 1 寄存器，如图。



仿真和汇编如图，

```
1.  a:jal x0,d
2.  e:jal x0 a
3.  c:jal x0 e
4.  d:jal x0 c
```



四个 jal 循环，汇编如图，从初始未命中到不断命中，命中后不会有 flush。

四． 指令扩展(alu 和 control)

```
1.  module control
2.  #(parameter
3.  ADDI=7'b0010011,
4.  ADD=7'b0110011,
5.  JAL=7'b1101111,
6.  BEQ=7'b1100011,
7.  LW=7'b0000011,
8.  SW=7'b0100011,
9.  JALR=7'b1100111,
10. AUIPC=7'b0010111
11. )
12. (
13. input hit,
14. input [31:0]in,
15. output[31:0]ctrl
```

```
    );
reg MemWrite, MemRead, RegWrite;
reg [1:0]ALUop; //只有 beq 是减 1
reg [1:0] RegSrc;//写回选择 00PC4,01memr,10y 写回寄存器
reg ALUSrc;//立即数 //0 选择 rd2,1 选择 imm
reg jal,beq,blt,jalr,ALUSrc0;
wire [6:0] opcode;
wire [2:0] funct3;
wire [6:0] funct7;
assign opcode=in[6:0];
assign funct3=in[14:12];
assign funct7=in[31:25];
always@(*)
begin
    case(in[6:0])
    ADDI:begin
    MemWrite <= 0;
    MemRead <= 0;
    RegWrite <= 1;
    RegSrc <= 2;
    ALUSrc0 <= 0;
    ALUSrc <= 1;
    ALUop <= 0;
    jal <= 0;
    beq <= 0;
    blt <= 0;
    jalr <= 0;
    end
    ADD:begin
    if(funct3==0)
    begin
    MemWrite <= 0;
    MemRead <= 0;
    RegWrite <= 1;
    RegSrc <= 2;
    ALUSrc0 <= 0;
    ALUSrc <= 0;
    ALUop <= funct7==0? 0:1;
    jal <= 0;
    beq <= 0;
    blt <= 0;
    jalr <= 0;
    end
    else
    begin
    MemWrite <= 0;
    MemRead <= 0;
    RegWrite <= 1;
    RegSrc <= 2;
```

```
65.     ALUSrc0 <= 0;
66.     ALUSrc <= 0;
67.     ALUop <= funct3==3'b111 ? 2'b10:2'b11;
68.     jal <= 0;
69.     beq <= 0;
70.     blt <= 0;
71.     jalr <= 0;
72.     end
73.     end
74.     JAL:begin
75.     MemWrite <= 0;
76.     MemRead <= 0;
77.     RegWrite <= 1;
78.     RegSrc <= 0;
79.     ALUSrc0 <= 0;
80.     ALUSrc <= 1;
81.     ALUop <= 0;
82.     jal <= 1;
83.     beq <= 0;
84.     blt <= 0;
85.     jalr <= 0;
86.     end
87.     JALR:begin
88.     MemWrite <= 0;
89.     MemRead <= 0;
90.     RegWrite <= 1;
91.     RegSrc <= 0;
92.     ALUSrc0 <= 0;
93.     ALUSrc <= 1;
94.     ALUop <= 0;
95.     jal <= 0;
96.     beq <= 0;
97.     blt <= 0;
98.     jalr <= 1;
99.     end
100.        BEQ:begin
101.        MemWrite <= 0;
102.        MemRead <= 0;
103.        RegWrite <= 0;
104.        RegSrc <= 0;
105.        ALUSrc0 <= 0;
106.        ALUSrc <= 0;
107.        ALUop <= 1;
108.        jal <= 0;
109.        beq <= funct3==0? 1:0;
110.        blt <= funct3==0? 0:1;
111.        jalr <= 0;
112.        end
113.        LW:begin
```

```
114.        MemWrite <= 0;
115.        MemRead <= 1;
116.        RegWrite <= 1;
117.        RegSrc <= 1;
118.        ALUSrc0 <= 0;
119.        ALUSrc <= 1;
120.        ALUop <= 0;
121.        jal <= 0;
122.        beq <= 0;
123.        blt <= 0;
124.        jalr <= 0;
125.        end
126.        SW:begin
127.        MemWrite <= 1;
128.        MemRead <= 0;
129.        RegWrite <= 0;
130.        RegSrc <= 0;
131.        ALUSrc0 <= 0;
132.        ALUSrc <= 1;
133.        ALUop <= 0;
134.        jal <= 0;
135.        beq <= 0;
136.        blt <= 0;
137.        jalr <= 0;
138.        end
139.        AUIPC:begin
140.        MemWrite <= 0;
141.        MemRead <= 0;
142.        RegWrite <= 1;
143.        RegSrc <= 2;
144.        ALUSrc0 <= 1;
145.        ALUSrc <= 1;
146.        ALUop <= 0;
147.        jal <= 0;
148.        beq <= 0;
149.        blt <= 0;
150.        jalr <= 0;
151.        end
152.        default:begin
153.        MemWrite <= 0;
154.        MemRead <= 0;
155.        RegWrite <= 0;
156.        RegSrc <= 0;
157.        ALUSrc0 <= 0;
158.        ALUSrc <= 0;
159.        ALUop <= 0;
160.        jal <= 0;
161.        beq <= 0;
162.        blt <= 0;
```

```
163.       jalr <= 0;
164.       end
165.       endcase
166.   end
167.   assign ctrl={hit,3'b0,4'b0,4'b0,1'b0,RegWrite,RegSrc,2'b0,MemRead,MemWrite,blt,jalr,jal,
   beq,2'b0,ALUSrc0,ALUSrc,2'b0,ALUop};
168.   endmodule
1.  module alu#(parameter WIDTH=32)
2.  (
3.  input [WIDTH-1:0]a,b,
4.  input [1:0]sel,
5.  output reg[WIDTH-1:0]c,
6.  output zero,
7.  output less
8.      );
9.  always@(*)
10.     begin
11.     case(sel)
12.     2'b00:c=a+b;
13.     2'b01:c=a-b;
14.     2'b10:c=a&b;
15.     2'b11:c=a|b;
16.     endcase
17.     end
18. //assign c= sel==0? a+b:a-b;
19. assign zero= c==0? 1:0;
20. assign less= c[31]==1? 1:0;
21. endmodule
```

扩展了基本的 blt, auipc, sub, jalr, or, and。

运行正确性由 VGA 展示和对应汇编保证

五. VGA 及对应汇编说明。

```
1.  la s8 back
2.  addi x2 zero 1
3.  addi x3 zero 2
4.  addi x4 zero 4
5.  addi x5 zero 8
6.  addi x6 zero 16
7.  addi x7 zero 32
8.  addi t6 zero 15
9.  again:
10. lw x1 0x408(zero)
11. blt x0 x1 judge
12. jal x0 again
13. judge:
14. and x8 x2 x1
15. beq x8 x2 N1
```

```
and x8 x3 x1
beq x8 x3 blue1
jal x0 N3
N1:
and x8 x3 x1
beq x8 x3 N2
blue:
jal s10 color
lw x1 0x408(zero)
and x8 x2 x1
beq x8 zero again
beq t6 x10 blue
addi x10 x10 1
jal x0 blue
blue1:
jal s10 color
lw x1 0x408(zero)
and x8 x3 x1
beq x8 zero again
beq zero x10 blue1
addi x11 zero 1
sub x10 x10 x11
jal x0 blue1
N2:
lw x1 0x408(zero)
and x8 x2 x1
beq x8 zero again
lw x1 0x408(zero)
and x8 x3 x1
beq x8 zero again
jal x0 N2
N3:
and x8 x4 x1
beq x8 x4 N4
and x8 x5 x1
beq x8 x5 green1
jal x0 N6
N4:
and x8 x5 x1
beq x8 x5 N5
green:
jal s10 color
lw x1 0x408(zero)
and x8 x4 x1
beq x8 zero again
beq t6 x9 green
addi x9 x9 1
jal x0 green
green1:
```

```
jal s10 color
lw x1 0x408(zero)
and x8 x5 x1
beq x8 zero again
beq zero x9 green1
addi x11 zero 1
sub x9 x9 x11
jal x0 green1
N5:
lw x1 0x408(zero)
and x8 x4 x1
beq x8 zero again
lw x1 0x408(zero)
and x8 x5 x1
beq x8 zero again
jal x0 N5
N6:
and x8 x6 x1
beq x8 x6 N7
and x8 x7 x1
beq x8 x7 red1
jal x0 again
N7:
and x8 x7 x1
beq x8 x7 N8
red:
jal s10 color
lw x1 0x408(zero)
and x8 x6 x1
beq x8 zero again
beq t6 x16 red
addi x16 x16 1
jal x0 red
red1:
jal s10 color
  lw x1 0x408(zero)
  and x8 x7 x1
  beq x8 zero again
  beq zero x16 red1
  addi x11 zero 1
  sub x16 x16 x11
  jal x0 red1
  N8:
  lw x1 0x408(zero)
  and x8 x6 x1
  beq x8 zero again
  lw x1 0x408(zero)
  and x8 x7 x1
  beq x8 zero again
```

```
jal x0 N8
color:
jal x0 slow
back:
addi s11 zero 0
add s11 s11 x10
add t5 zero x9
add t4 zero x16
add x9 x9 x9
add x9 x9 x9
add x9 x9 x9
add x9 x9 x9
add x16 x16 x16
add x16 x16 x16
add x16 x16 x16
add x16 x16 x16
add x16 x16 x16
add x16 x16 x16
add x16 x16 x16
add x16 x16 x16
add s11 s11 x16
add s11 s11 x9
sw s11 0x408(zero)
add x9 zero t5
add x16 zero t4
jalr x0 0(s10)
slow:
addi s9 zero 0
addi t3 zero 1024
add t3 t3 t3
add t3 t3 t3
add t3 t3 t3
add t3 t3 t3
add t3 t3 t3
add t3 t3 t3
add t3 t3 t3
add t3 t3 t3
add t3 t3 t3
add t3 t3 t3
add t3 t3 t3
add t3 t3 t3
slow1:
addi s9 s9 1
blt s9 t3 slow1
jalr x0 0(s8)
```

汇编程序运用了所有指令，控制画笔红绿蓝颜色增减，进行绘图，保证 CPU 实现的指令是正确的。