一、实验内容

编写 MIPS 应用程序，用完成的 54 条指令 MIPS CPU 运行

二、小程序介绍

程序名称：游程编码解码器

程序介绍：解析内存中以游程编码形式保存的二值图，并把解析后的数据送到 VGA 中，使得 VGA 能够显示该图片。增加图片的数量从而达到播放黑白视频的效果。

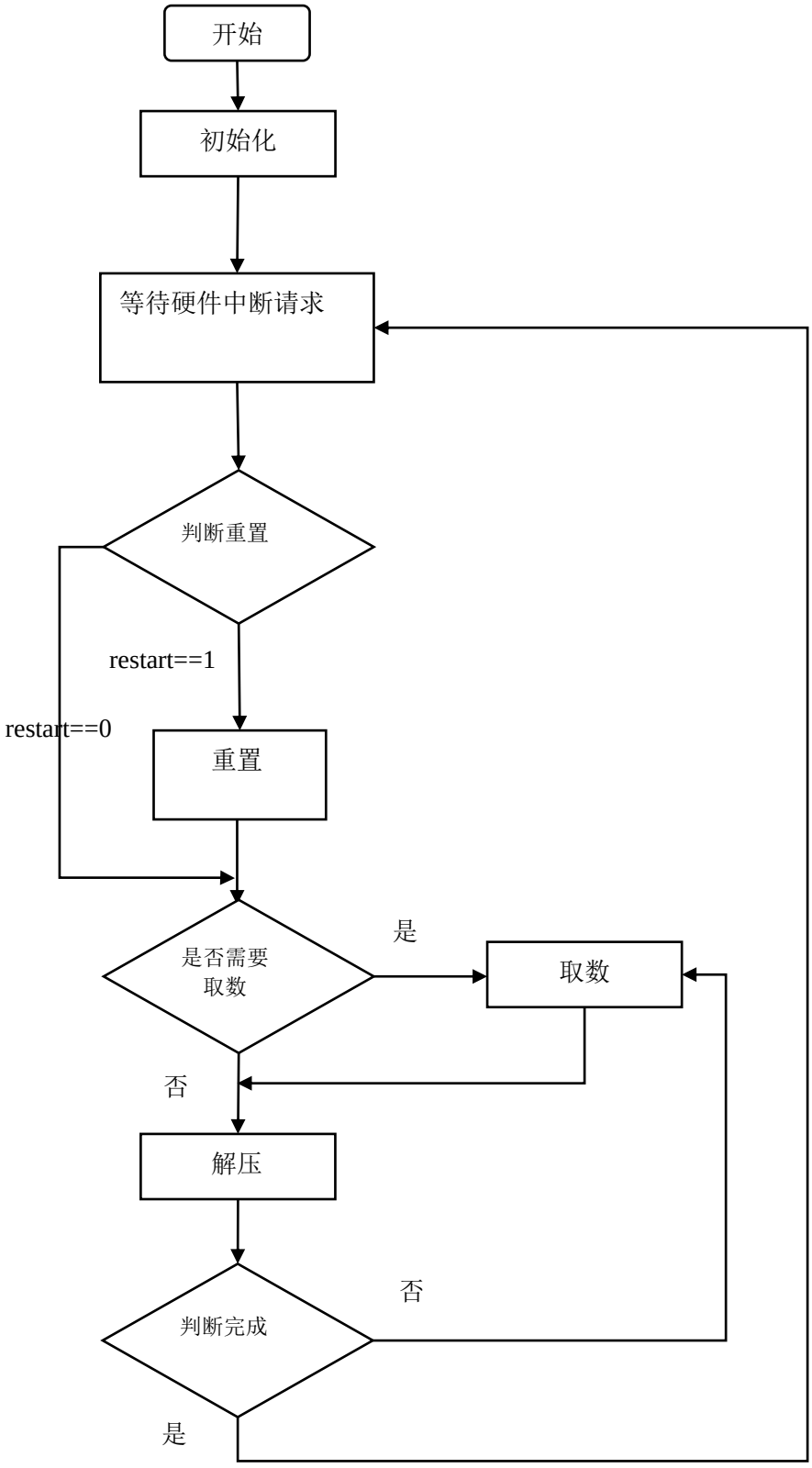
三、外设

输入设备：开发板

确定键/复位键：N17

输出设备：VGA 显示器

四、程序流程图


开始
初始化
等待硬件中断请求
判断重置
restart==1
restart==0
重置
是否需要
取数
是
取数
否
解压
判断完成
否
是

五、演示图片

六、相关模块建模

1. vga

```
module vga(
input clk_in,//50M
input clk_in_25,
input rst_in,
input[31:0]i_data,
input we,
output hsync,
output vsync,
output [3:0]vga_r,
output [3:0]vga_g,
output [3:0]vga_b,
output intr,
output [31:0]o_data
      );

wire clk=clk_in_25;
wire rst=rst_in;

wire vga_valid;
wire[31:0]show_data;
wire restart;

reg[31:0]data2;
```

```
// data2
//cpu 传来写信号  50M
always @(posedge clk_in or posedge rst) begin
   if (rst) begin
//       data2<=32'd0;
         data2 <=32'hffffffff;
   end
   else if (we) begin
      data2<=i_data;//CPU 写入 I/O
   end
   else begin
      data2<=data2;
   end
end

assign o_data={{31{1'b0}},restart};

VGA_640x480 vga_6_4_inst(
.clk(clk),
.rst(rst),
.data2(data2),
.HS(hsync),
.VS(vsync),
.valid(vga_valid),
.o_data(show_data),
.intr(intr),
.rstart(restart));


VGA_Color vc_inst(
.clk(clk),
.rst(rst),
.valid(vga_valid),
.data(show_data),
.red(vga_r),
.green(vga_g),
.blue(vga_b));

Endmodule
```

2. vga_color

```
module VGA_Color(
input clk,
```

```
input rst,
input valid,
input [31:0]data,
output reg[3:0]red,
output reg[3:0]green,
output reg[3:0]blue
    );

reg [4:0]count;//at most of 31
always @(posedge clk or posedge rst) begin
    if (rst) begin
        // reset
        count<=0;
    end
    else if (valid) begin
        count<=count+1;
    end
    else begin
        count <=0;
    end
end

always @(*) begin
    if (valid) begin
        if(data[31-count]==1)begin
            red=4'b0000;green=4'b0000;blue=4'b0000;//black
        end
        else begin
            red=4'b1111;green=4'b1111;blue=4'b1111;//white
        end
    end
    else begin
         red=4'b0000;green=4'b0000;blue=4'b0000;
    end
end
endmodule
```

3. vga_640x480

```
module VGA_640x480(
    input wire clk,//分频的时钟，频率为 25mhz
    input wire rst,
    input [31:0] data2,
    output HS,
    output VS,
```

```
output valid,
output [31:0] o_data,
output intr,
output rstart
);

reg [31:0]h_count;
reg [31:0]v_count;
wire[9:0]xpos;
wire[9:0]ypos;

localparam vga_x=640;
localparam vga_y=480;
localparam pict_x=640;
localparam pict_y=480;

localparam pict_num=1;
localparam play_time_num=4;// every 5 pictures is played. the display time between two pictures is less than 0.1
// localparam addr_num=pict_x/32*pict_y;

reg[14:0]addr;
reg[3:0]pict_count;
reg[4:0]play_time_count;
wire[31:0]tdata;

reg[31:0]data1;
reg req;//32 位的数据用完，向 cpu 请求数据
reg ini;//初始化，向 cpu 请求数据
reg flag;
reg restart;
// reg flag_2_to_1;//rst 之后  把 data2 给 data1
assign rstart = restart;

// assign o_data = ((xpos<pict_x) && (ypos < pict_y))?tdata:32'd0;
// assign o_data = (play_time_count>0)?32'd0:(((xpos<pict_x) && (ypos < pict_y))?data1:32'd0);
assign o_data = (play_time_count>0)?32'd0:(((xpos<pict_x) && (ypos < pict_y))?((xpos%32==0)?data2:data1):32'd0);




//ini flag
```

```
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            ini<=0;
            flag<=0;
        end
        else if (h_count== 10'd96 && flag==0) begin
            ini<=1;
            flag<=1;
        end
        else begin
            ini<=0;
        end
    end


    assign intr = ini|req;




    // req
    // data1
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            req<=0;
            data1<=32'hffffffff;
//          data1<=32'h0;
        end
        else if (valid) begin
            if (xpos % 32 == 0 && (xpos<pict_x) && (ypos <pict_y)) begin//进一步判读是否
需要发出 req
                if ((ypos == pict_y - 1) && (xpos == pict_x - 32)) begin//在一帧图像的结尾
                    if(play_time_count==play_time_num-1)begin//一帧图播放到最后一次
                        req<=1;
                        data1<=data2;
                    end
                    else begin
                        req<=0;
                        data1<=data2;
                    end
                end
                else begin
                    if (play_time_count == 0) begin
                        req<=1;
```

```
                        data1<=data2;
                    end
                    else begin
                        req<=0;
                        data1<=data1;//  play_time_count>0  的时候  上面已经操作过使得
o_data=0
                    end
                end
            end
            else begin
                req<=0;
                data1<=data1;
            end
        end
        else begin
            req<=0;
            data1<=data1;
        end
    end

    // restart
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            restart<=0;
        end
        else if (valid && (xpos % 32 == 0) && (ypos == pict_y - 1) && (xpos == pict_x - 32)
&& (play_time_count==play_time_num-1) &&(pict_count == pict_num - 1)) begin
            restart<=1;
        end
        else if(v_count == 0)begin
            restart<=0;
        end
        else begin
            restart<=restart;
        end
    end




    // // addr
    // always @(posedge clk or posedge rst) begin
    //     if (rst) begin
    //         addr<=0;
```

```
//     end
//     else if (valid) begin
//          if (xpos % 32 == 31 && (xpos<pict_x) && (ypos <pict_y)) begin
//               if ((ypos == pict_y - 1) && (xpos == pict_x -1)) begin
//                    if(play_time_count==play_time_num-1)begin
//                         if(pict_count == pict_num - 1)begin
//                              addr<=0;
//                         end
//                         else begin
//                              addr<=addr+1;
//                         end
//                    end
//                    else begin
//                         addr<=addr;//
//                    end
//               end
//               else begin
//                    if (play_time_count == 0) begin
//                         addr<=addr+1;
//                    end
//                    else begin
//                         addr<=addr;
//                    end

//               end
//          end
//     end
//     else begin
//          addr<=addr;
//     end
// end

//play_time_count
always @(posedge clk or posedge rst) begin
     if (rst) begin
          // reset
          play_time_count<=0;
     end
     else if ((ypos == vga_y - 1) && (xpos == vga_x -1)) begin
          if(play_time_count == play_time_num-1)begin
               play_time_count<=0;
          end
          else begin
               play_time_count<=play_time_count+1;
```

```
            end
        end
        else begin
            play_time_count<=play_time_count;
        end
    end

    // pict_count
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            pict_count<=0;
        end
        else if ((ypos == vga_y - 1) && (xpos == vga_x -1) && (play_time_count == play_time_num-1)) begin
            if(pict_count == pict_num - 1)begin
                pict_count<=0;
            end
            else begin
                pict_count<=pict_count+1;
            end
        end
        else begin
            pict_count<=pict_count;
        end
    end




    //行计数：h_count(0-639+8+8+96+40+8 = 799)
    always@(posedge clk or posedge rst)begin
        if (rst) begin
            h_count<=0;
        end
        else if(h_count == 10'd799)
            h_count <= 10'h0;
        else
            h_count <= h_count + 10'h1;
    end


    //帧计数：v_count(0-524)
    always@(posedge clk or posedge rst)begin
        if (rst) begin
```

```
            v_count<=0;
        end
        else if(h_count == 10'd799)begin
            if(v_count == 10'd524)v_count <= 10'h0;
            else v_count <= v_count + 10'h1;
        end
    end

    assign xpos = valid?(h_count - 10'd143):0;
    assign ypos = valid?(v_count - 10'd35):0;
    assign VS = (v_count >= 10'd2);
    assign HS = (h_count >= 10'd96);
    assign valid = (((h_count >= 10'd143)&&(h_count < 10'd783)) && ((v_count >= 10'd35)
&& (v_count < 10'd515)));
endmodule
```

4. io_sel

```
module io_sel(
    input [31:0] addr,
     input cs,
     input sig_w,
     input sig_r,
     output dmem_cs,
     output vga_cs
    );

// assign seg7_cs = (addr == 32'h10010000 && cs == 1 && sig_w == 1) ? 1 : 0;
// assign switch_cs = (addr == 32'h10010010 && cs == 1 && sig_r == 1) ? 1 : 0;
    assign vga_cs = ((addr == 32'h10810000 | addr == 32'h10810004)&& cs == 1) ? 1 : 0;
    assign dmem_cs = cs & (~vga_cs);
endmodule
```

5. sscpu_dataflow

```
module sccomp_dataflow(
input clk_in,
input reset,
input [15:0] sw,
output [7:0] o_seg,
output [7:0] o_sel,
output hsync,
output vsync,
output [3:0]vga_r,
output [3:0]vga_g,
output [3:0]vga_b
```

```
        );
wire locked;
wire exc;
wire [31:0]status;

wire [31:0]rdata;
wire [31:0]wdata;
wire IM_R,DM_CS,DM_R,DM_W;
wire [31:0]inst,pc,addr;
wire inta,intr;
wire clk;
wire [31:0]data_fmem;
wire [31:0]data_fvga;
wire rst=reset|~locked;
wire [31:0]ip_in;
wire seg7_cs,switch_cs;

assign ip_in = pc-32'h00400000;

wire dmem_cs;
wire vga_cs;
wire clk_vga;

clk_wiz_0 clk_inst
     (
       // Clock out ports
       .clk_out1(clk),          // output clk_out1
       .clk_out2(clk_vga),          // output clk_out2
       // Status and control signals
       .reset(reset), // input reset
       .locked(locked),             // output locked
     // Clock in ports
       .clk_in1(clk_in));

//clk_div #(3)cpu_clk(clk_in,clk);

/*地址译码*/
// io_sel io_mem(addr, DM_CS, DM_W, DM_R, seg7_cs, switch_cs);
io_sel io_mem(
     .addr(addr),
     .cs(DM_CS),
     .sig_w(DM_W),
     .sig_r(DM_R),
     .dmem_cs(dmem_cs),
```

```
    .vga_cs(vga_cs)
      );

CPU54 sccpu(clk,rst,inst,rdata,pc,addr,wdata,IM_R,DM_CS,DM_R,DM_W,intr,inta);
//rdata  从 dmem 中读取来的数据


/*指令存储器*/
//imem imem(ip_in[12:2],inst);
//imemory im(pc,inst);
dist_iram_ip IMEM (
   .a(ip_in[12:2]),           // input wire [10 : 0] a
   .spo(inst)    // output wire [31 : 0] spo
);

wire [31:0]addr_in=addr-32'h10010000;

/*数据存储器*/
dist_dmem_ip DMEM (
   .a(addr_in[16:2]),           // input wire [10 : 0] a
   .d(wdata),           // input wire [31 : 0] d
   .clk(clk),    // input wire clk
   .we(dmem_cs&DM_W),        // input wire we
   .spo(data_fmem)    // output wire [31 : 0] spo
);


//dmem scdmem(~clk,reset,DM_CS,DM_W,DM_R,addr-32'h10010000,wdata,data_fmem);



// seg7x16 seg7(clk, reset, seg7_cs, wdata, o_seg, o_sel);

// sw_mem_sel sw_mem(switch_cs, sw, data_fmem, rdata);

      vga vga_inst(
.clk_in(clk),//50M
.clk_in_25(clk_vga),
.rst_in(rst),
.i_data(wdata),
.we(vga_cs&DM_W),
.hsync(hsync),
.vsync(vsync),
.vga_r(vga_r),
```

```
.vga_g(vga_g),
.vga_b(vga_b),
.intr(intr),
.o_data(data_fvga)
        );

    assign rdata = vga_cs?data_fvga:data_fmem;

endmodule
```

七、MIPS 源程序

```
j _start
sll $0, $0, 0
j _exceptions
sll $0, $0, 0


_start:
lui $a0,0x1001 #a0 为 dmem 取数首地址
lui $a1,0x1081 #a1 为 vga in  地址

lui $a2,0x1081
addiu $a2,$a2,0x0004 #a2 为 vga status  地址

lui $a3,0x1001 #a3 为数据地址
ori $s1,$0,0x8000 #s1 用来判断数据是否压缩
ori $s2,$0,0x4000 #s1 用来判断数据发生压缩时，是 0 还是 1
lui $s6,0x8000 #s6  恒等于 0x8000_0000
sra $s7,$s6,31        # s7=ffff_ffff
ori $t8,$0,0x0020 # t8 恒为 32
ori $t9,$0,0x001f # t9 恒为 31
ori $t7,$0,0x3fff # t7  恒为    0011_1111_1111_1111


cycle:
ori $v1,$0,0x0020
ori $v1,$0,0x0020
ori $v1,$0,0x0020
ori $v1,$0,0x0020
ori $v1,$0,0x0020
j cycle
```

```
_exceptions:
#先选 VGA 状态寄存器地址，查看是否需要把数据地址置为 0
lw $t0,($a2)
and $s5,$0,$s0
beq $t0,$0,no_restart #等于 0 的时候不需要重置，否则重置
#重新播放图片 寄存器全部重置
addu $a3,$a0,$0
and $v0,$0,$0


no_restart:
bne $v0,$0,no_fatch #先判断是否有剩余位
fatch_num:
lhu $s0,($a3) #此时要取数  半字无符号扩展
addiu  $a3,$a3,0x0002 #地址+2
and $t1,$s1,$s0 #判断是否压缩
sltu $s3,$t1,$s1 #0000000 -1    , 0000_8000 - 0

beq $s3,$0,label3
#不压缩
and $s0,$s0,$t7 #留下 14 位有效
addiu $v0,$0,0x000e # v0 =14
j no_fatch

label3:
#压缩
and $t1,$s2,$s0 #判断压缩 0/1
sltu $s4,$t1,$s2 #0000000 -1    , 0000_4000 - 0
and $v0,$s0,$t7 #留下 14 位有效


no_fatch:
bne $s3,$0,no_compress

compress:
sltu $t0,$v0,$v1 # v0<v1 -- 1    , v0>=v1 ---0
bne $t0,$0,cp_2

#v1<=v0
sllv $s5,$s5,$v1  #把输出结果左移
and $t2,$0,$0 #置 0，保证压缩 0 的时候也能得到正确结果
bne $s4,$0,label1
or $t2,$0,$s7
beq $v1,$t8,label1 #v1=32 时的特殊处理
```

```
#对 1 的压缩

subu $t1,$t9,$v1
srav $t2,$s6,$t1
xor $t2,$t2,$s7

label1:
or $s5,$s5,$t2 #拼装出最后结果
subu $v0,$v0,$v1
j output

cp_2:
# v0<v1
sllv $s5,$s5,$v0    #把输出结果左移
and $t2,$0,$0 #置 0，保证压缩 0 的时候也能得到正确结果
bne $s4,$0,label2
#对 1 的压缩
subu $t1,$t9,$v0
srav $t2,$s6,$t1
xor $t2,$t2,$s7

label2:
or $s5,$s5,$t2 #拼装出最后结果
subu $v1,$v1,$v0
j fatch_num



no_compress:
sltu $t0,$v0,$v1 # v0<v1 -- 1      , v0>=v1 ---0
beq $t0,$0,no_cp_1

# v0<v1
sllv $s5,$s5,$v0    #把输出结果左移
or $s5,$s5,$s0    #拼装出 S5
subu $v1,$v1,$v0 #修改剩余  需要的拼装位数
j fatch_num

no_cp_1:
#v1<=v0
sllv $s5,$s5,$v1    #把输出结果左移
subu $v0,$v0,$v1
srlv $t2,$s0,$v0 #右移两个差  位
or $s5,$s5,$t2 #拼装出最后结果
#s0 保留  v0 位
```

```
subu $t2,$t8,$v0
sllv $s0,$s0,$t2
srlv $s0,$s0,$t2

output:
sw $s5,($a1) #把数据送到 vga
#j _epc_plus4

#_epc_plus4:
#sll $0, $0, 0
mfc0 $k0, $14
addi $k0, $k0, 0x4
mtc0 $k0, $14
eret
```