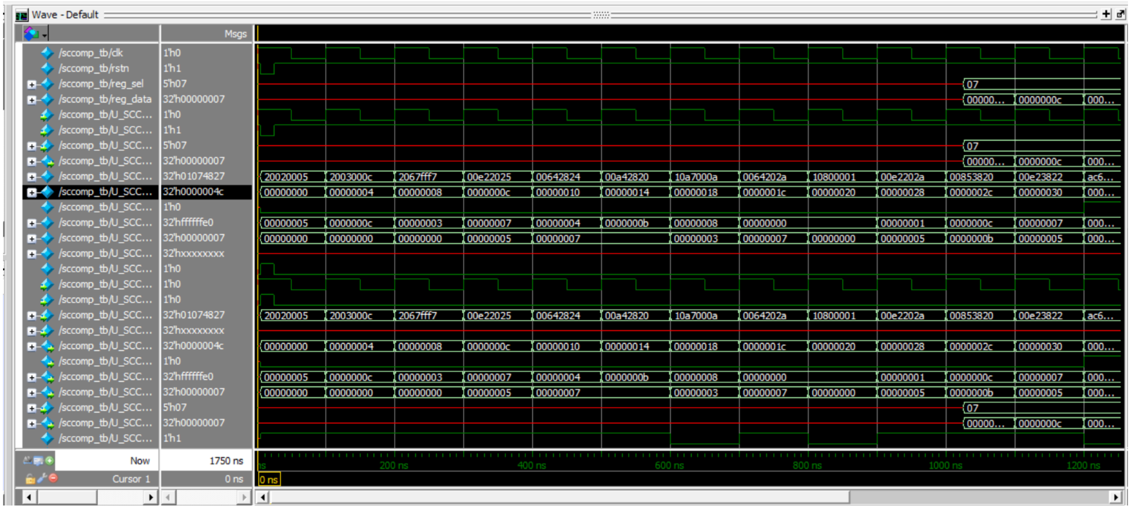
单周期CPU的仿真测试

1. Testbench

```
module sccomp_tb();

//reg_sel是寄存器组的测试选择信号
//reg_data是寄存器组的测试输出信号
reg  clk, rstn;
reg  [4:0] reg_sel;
wire [31:0] reg_data;

//将芯片+存储器实例化
sccomp U_SCCOMP(
   .clk(clk), .rstn(rstn), .reg_sel(reg_sel), .reg_data(reg_data)
);

initial begin

   //将预先准备好的指令文件CPUtest.dat加载到指令存储器中
   $readmemh( "CPUtest.dat" , U_SCCOMP.U_IM.ROM);
   //打印PC和指令，用于调试(打印到终端)
   $monitor("PC = 0x%8X, instr = 0x%8X", U_SCCOMP.PC, U_SCCOMP.instr);

   //初始化，等待5个时间周期让系统稳定在初始态
   clk = 1;
   rstn = 1;
   #5 ;

   //再复位，并等待20个时间周期让系统稳定在复位态
   rstn = 0;
   #20 ;

   //再释放复位，并等待1000个时间周期让系统稳定在工作态
   rstn = 1;
   #1000 ;
   reg_sel = 7;
end

//让时钟信号在每个时间周期的中间变换一次，以便于观察
always begin
 #(50) clk = ~clk;
end

endmodule
```

2. 仿真结果

> 1️⃣波形


Wave - Default
Msgs
/sccomp_tb/clk 1'h0
/sccomp_tb/rstn 1'h1
/sccomp_tb/reg_sel 5'h07
/sccomp_tb/reg_data 32'h00000007
/sccomp_tb/U_SCC... 1'h0
/sccomp_tb/U_SCC... 1'h1
/sccomp_tb/U_SCC... 5'h07
/sccomp_tb/U_SCC... 32'h00000007
/sccomp_tb/U_SCC... 32'h01074827
/sccomp_tb/U_SCC... 32'h0000004c
/sccomp_tb/U_SCC... 1'h0
/sccomp_tb/U_SCC... 32'hffffffe0
/sccomp_tb/U_SCC... 32'h00000007
/sccomp_tb/U_SCC... 32'hxxxxxxxx
/sccomp_tb/U_SCC... 1'h0
/sccomp_tb/U_SCC... 1'h0
/sccomp_tb/U_SCC... 1'h0
/sccomp_tb/U_SCC... 32'h01074827
/sccomp_tb/U_SCC... 32'hxxxxxxxx
/sccomp_tb/U_SCC... 32'h0000004c
/sccomp_tb/U_SCC... 1'h0
/sccomp_tb/U_SCC... 32'hffffffe0
/sccomp_tb/U_SCC... 32'h00000007
/sccomp_tb/U_SCC... 5'h07
/sccomp_tb/U_SCC... 32'h00000007
/sccomp_tb/U_SCC... 1'h1
Now 1750 ns
Cursor 1 0 ns
200 ns
400 ns
600 ns
800 ns
1000 ns
1200 ns

2 在控制台中不断打印出指令和PC

```
run
# PC = 0xxxxxxxxx, instr = 0xxxxxxxxx
# PC = 0x00000000, instr = 0x20020005
run
# r[ 2] = 0x00000005,
run
# PC = 0x00000004, instr = 0x2003000c
run
# r[ 3] = 0x0000000c,
run
# PC = 0x00000008, instr = 0x2067fff7
run
# r[ 7] = 0x00000003,
run
# PC = 0x0000000c, instr = 0x00e22025
run
# r[ 4] = 0x00000007,
run
# PC = 0x00000010, instr = 0x00642824
run
# r[ 5] = 0x00000004,
run
# PC = 0x00000014, instr = 0x00a42820
run
# r[ 5] = 0x0000000b,
run
# PC = 0x00000018, instr = 0x10a7000a
run
run
# PC = 0x0000001c, instr = 0x0064202a
run
# r[ 4] = 0x00000000,
run
# PC = 0x00000020, instr = 0x10800001
run
run
# PC = 0x00000028, instr = 0x00e2202a
run
# r[ 4] = 0x00000001,
run
# PC = 0x0000002c, instr = 0x00853820
run
# r[ 7] = 0x0000000c,
run
# PC = 0x00000030, instr = 0x00e23822
run
# r[ 7] = 0x00000007,
run
# PC = 0x00000034, instr = 0xac6700f4
run
# dmem[0x00000100] = 0x00000007,
run
# PC = 0x00000038, instr = 0x8c020100
run
# r[ 2] = 0x00000007,
run
# PC = 0x0000003c, instr = 0x0c000011
run
# r[31] = 0x00000040,
run
# PC = 0x00000044, instr = 0xac020104
run
# dmem[0x00000104] = 0x00000007,
run
# PC = 0x00000048, instr = 0x00074080
run
# r[ 8] = 0x0000001c,
run
# PC = 0x0000004c, instr = 0x01074827
```

3. 总结

> 实现了13条R型指令，9条i型指令，4条j型指令，共26条