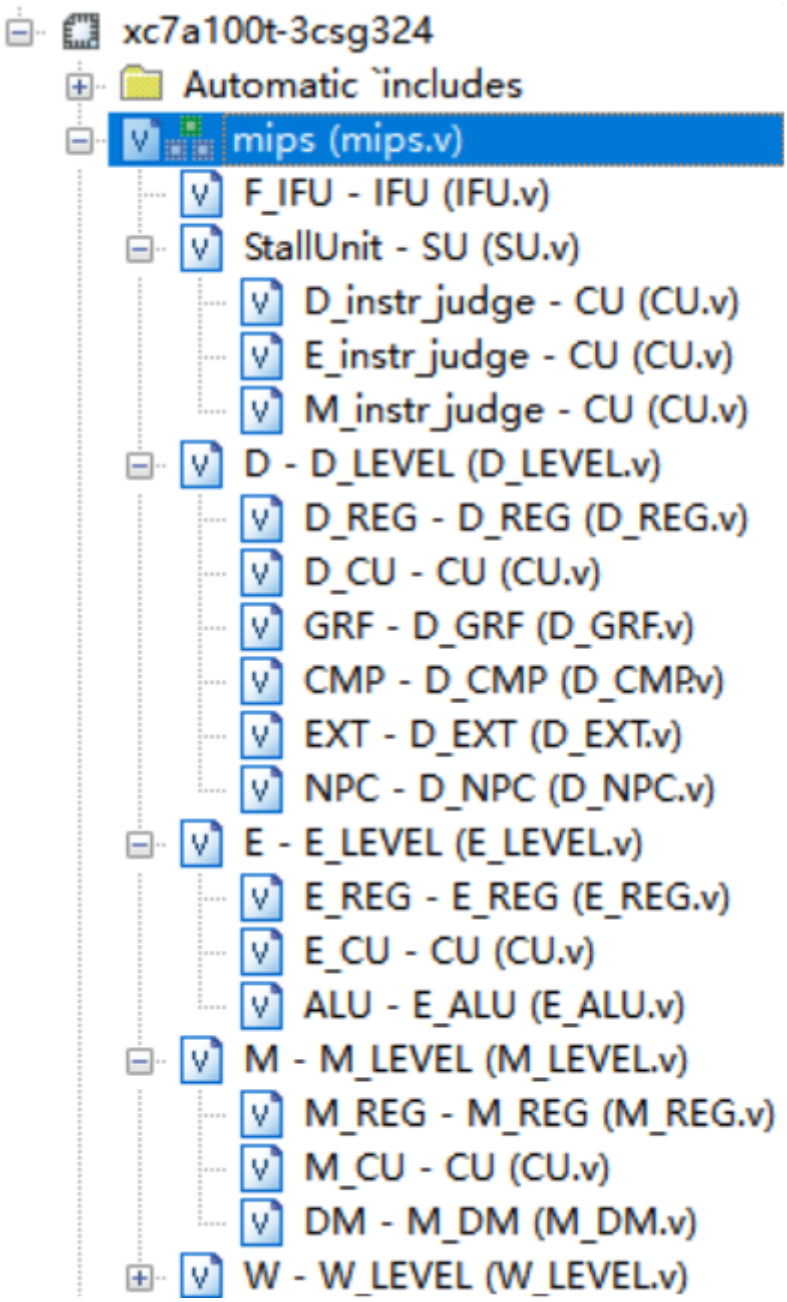
Verilog流水线CPU设计

一、CPU设计方案综述

（一）总体设计概述

本CPU为Logisim实现的32位5级流水线MIPS-CPU，支持指令集包含`{addu，subu，ori，lw，sw，beq，j，jal，jr，lui，nop，j，lh，lhu，lb，lbu，sh，sb}`。为实现相关功能，CPU主要包含IFC、NPC、NPC、IM、GRF、EXT、ALU、DM、CU、SU等模块。遵循形式化建模综合方法完成设计与实现。

（二）关键模块定义



1.暂停控制器-SU

```
module SU(
    input [31:0] D_instr,
    input [31:0] E_instr,
    input [31:0] M_instr,
    output stall
    );

    wire [1:0] Tuse_RS, Tuse_RT;
    wire [1:0] Tnew_E, Tnew_M;
    wire stall_rs_e, stall_rs_m, stall_rs;
    wire stall_rt_e, stall_rt_m, stall_rt;

    assign stall_rs_e = (Tuse_RS < Tnew_E) && D_instr[25:21] && (D_instr[25:21]
== E_RFA3) && E_RFWr;
    assign stall_rs_m = (Tuse_RS < Tnew_M) && D_instr[25:21] && (D_instr[25:21]
== M_RFA3) && M_RFWr;
    assign stall_rs = stall_rs_e | stall_rs_m ;

    assign stall_rt_e = (Tuse_RT < Tnew_E) && D_instr[20:16] && (D_instr[20:16]
== E_RFA3) && E_RFWr;
    assign stall_rt_m = (Tuse_RT < Tnew_M) && D_instr[20:16] && (D_instr[20:16]
== M_RFA3) && M_RFWr;
    assign stall_rt = stall_rt_e | stall_rt_m ;

    assign stall = stall_rs | stall_rt ;

// ============== D级 ======================//
    wire D_cal_r, D_cal_i, D_load, D_store, D_branch, D_j_reg;
    CU D_instr_judge (
    .op(D_instr[31:26]),
    .funct(D_instr[5:0]),
    .rt_op(D_instr[20:16]),
     //output
    .cal_r(D_cal_r),
    .cal_i(D_cal_i),
    .load(D_load),
    .store(D_store),
    .branch(D_branch),
    .j_reg(D_j_reg)
    );

    assign Tuse_RS = (D_cal_r | D_cal_i | D_load | D_store) ? 2'd1:
                         (D_branch | D_j_reg) ? 2'd0 : 2'd0 ;
    assign Tuse_RT = (D_store) ? 2'd2 :
                         (D_cal_r) ? 2'd1 :
                         (D_branch)? 2'd0 : 2'd0 ;

// ============== E级 ======================//
    wire E_cal_r, E_cal_i, E_load;
    wire E_RFWr;
    wire [1:0] E_GRFA3Sel;
    wire [4:0] E_RFA3;
    CU E_instr_judge (
    .op(E_instr[31:26]),
    .funct(E_instr[5:0]),
```

```
    .rt_op(E_instr[20:16]),
     //output
    .cal_r(E_cal_r),
    .cal_i(E_cal_i),
    .load(E_load),

    .RFWr(E_RFWr),
    .GRFA3Sel(E_GRFA3Sel)
    );

    assign E_RFA3 = (E_GRFA3Sel == `GRFA3_rt) ? E_instr[20:16] :
                        (E_GRFA3Sel == `GRFA3_rd) ? E_instr[15:11] :
                        (E_GRFA3Sel == `GRFA3_31) ? 5'd31 :
                          E_instr[20:16];
    assign Tnew_E = (E_load) ? 2'd2 :
                        (E_cal_r | E_cal_i) ? 2'd1 :
                           2'd0;

// ============== M级 ====================//
    wire M_load;
    wire M_RFWr;
    wire [1:0] M_GRFA3Sel;
    wire [4:0] M_RFA3;
    CU M_instr_judge (
    .op(M_instr[31:26]),
    .funct(M_instr[5:0]),
    .rt_op(M_instr[20:16]),
     //output
    .load(M_load),

    .RFWr(M_RFWr),
    .GRFA3Sel(M_GRFA3Sel)
    );

    assign M_RFA3 = (M_GRFA3Sel == `GRFA3_rt) ? M_instr[20:16] :
                        (M_GRFA3Sel == `GRFA3_rd) ? M_instr[15:11] :
                        (M_GRFA3Sel == `GRFA3_31) ? 5'd31 :
                          M_instr[20:16];
    assign Tnew_M = M_load ? 2'd1 : 2'd0 ;

endmodule
```

2.转发模块

分散在各级，列举典型例子:

E_LEVEL 发出转发数据:

```
    wire [3:0] ALUOp;
    wire [1:0] ALUBSel;
    wire [1:0] GRFA3Sel,GRFWDSel;//解析回写控制信号
    wire j_link; //解析当前指令类型，以判断是否以产生数据可以转发
    CU E_CU (
    .op(IR_out[31:26]),
    .funct(IR_out[5:0]),
    .rt_op(IR_out[20:16]),
     //output
```

```
    .ALUOp(ALUOp),
    .ALUBSel(ALUBSel),

     .RFWr(E_RFWr_out),
    .GRFA3Sel(GRFA3Sel),
    .GRFWDSel(GRFWDSel),
     //指令检测
     .j_link(j_link)
    );

    assign E_RFA3_out = (GRFA3Sel == `GRFA3_rt) ? IR_out[20:16] :
                        (GRFA3Sel == `GRFA3_rd) ? IR_out[15:11] :
                        (GRFA3Sel == `GRFA3_31) ? 5'd31 :
                          IR_out[20:16];
    assign E_RFWD_out = (GRFWDSel == `GRFWD_pc8) ? PC_out + 8 :
                            0 ;
    assign E_Forward_Ready_out = j_link ;
```

E_LEVEL 接收并选择转发数据

```
    wire [31:0] ALU_A, ALU_B;
    wire [31:0] FWD_RS, FWD_RT;
    assign FWD_RS = (IR_out[25:21] == 0) ? 0 :
                    (IR_out[25:21] == M_RFA3_in && M_RFWr_in &&
M_Forward_Ready_in) ? M_RFWD_in :
                    (IR_out[25:21] == W_RFA3_in && W_RFWr_in &&
W_Forward_Ready_in) ? W_RFWD_in :
                     V1_out ;
    assign FWD_RT = (IR_out[20:16] == 0) ? 0 :
                    (IR_out[20:16] == M_RFA3_in && M_RFWr_in &&
M_Forward_Ready_in) ? M_RFWD_in :
                    (IR_out[20:16] == W_RFA3_in && W_RFWr_in &&
W_Forward_Ready_in) ? W_RFWD_in :
                     V2_out0 ;
    assign ALU_A = FWD_RS ;
    assign ALU_B =  (ALUBSel == `ALUB_rd2) ? FWD_RT         :
                    (ALUBSel == `ALUB_imm) ? EXT_out        :
                    (ALUBSel == `ALUB_shamt) ? IR_out[10:6] : 0 ;

    assign V2_out = FWD_RT; //转发处理后正确的数据才能传到下一级
```

（三） 重要机制实现方法

1.跳转

这里确实容易出错， 因为寄存器的原因导致了PC和NPC的分离，若使用D级PC信号会导致NPC的滞后而产生类似指令回流错觉的错误。因而得用F级PC的信号做PC正常累加。其他的同单周期CPU设置NPCOp的方法。

2.流水线延迟槽

jal的改存PC+8进入寄存器

3.转发

E、M、W级都具有完整指令，将其解析出所有写寄存器的信息（不管是不是写入指令）一起转发到相应D、E、M共5处转发位点。设置转发判断相关条件已控制是否选择转发的信号

4.暂停

AT法分析Tnew和Tuse并比较，当Tuse < Tnew且当前指令是冲突指令（需要转发）时，需要暂停

T_use < T_new : 暂停

指令	T_use	
	rs([25:21])	rt([20:16]])
cal_r	1	1
cal_i	1	
load	1	
store	1	2
branch	0	0
j_reg	0	0
j_link		
j_addr		

D			E			M
指令	源寄存器	T_use	Tnew_E			Tnew_M
			cal_r (rd)	cal_i (rt)	load (rt)	load (rt)
			1	1	2	1
cal_r	rs/rt	1			stall	
cal_i	rs	1			stall	
load	rs	1			stall	
store	rs	1			stall	
store	rt	2				
branch	rs/rt	0	stall	stall	stall	stall
j_reg	rs	0	stall	stall	stall	stall

二、测试方案

（一）手动少量测试

转发路线的测试

```
ori $t0, $0, 8
ori $t1, $0, 9
addu $t2, $t0, $t1
# W_alu转E_rs ; M_alu转E_rt
# 交换$t0 $t1顺序 则可检测： W_alu转E_rt ; M_alu转E_rs


ori $t0, $0, 4
lui $t7, 0x1234
sw $t7, 4($t0)
lw $t8, 4($t0)
sw $t8, 8($t0)
# W_alu转M_rt ; W_dm转M_rt ; W_alu转E_rs


ori $t0, $0, 4
ori $t1, $0, 0x55
sw $t1, 4($t0)   # above tested already, assum it's right
lw $t5, 4($t0)
nop
addu $t6, $t5, $t5
# W_dm转E_rs ; W_dm转E_rt
```

```
ori $t3, $0, 3
ori $t4, $0, 4
ori $t5, $0, 4
sw $t5, 4($0)
lw $t3, 4($0)
nop
nop
beq $t3, $t4, label
nop
nop
addu $t0, $t3, $t4
label:
lui $t1, 0x1111
nop
# W_dm转D_rs ; 交换$t3 $t4位置 W_dm转D_rt
# 去掉lw后两个nop，检验了lw接beq的暂停正确


ori $t1, $0, 10
ori $t2, $0, 6
nop
nop
ori $t1, $0, 5
ori $t2, $0, 5
beq $t1, $t2, loop
nop
addu $t8, $t1, $t2
loop:
lui $t9, 10
# W_alu转D_rs ; M_alu转D_rt
# 交换beq中$t1 $t2后可测 : W_alu转D_rt ; M_alu转D_rs


ori $t0, $0, 8
jal loop
sw $ra, 4($t0)
lui $t1, 8
loop:
lw $t2, 4($t0)
# W_pc8转M_rt


ori $t0, $0, 8
ori $t3, $0, 0x3010
jal loop
nop
sw $ra, 4($t0)
lui $t1, 8
loop:
addu $t2, $ra, $ra
beq $ra, $t3, branch
nop
jr $ra
nop
branch:
lui $t3, 12
# W_pc8转E_rs ; W_pc8转E_rt
# W_pc8转D_rs; 交换$ra $t3后可测: W_pc8转D_rt
```

```
ori $s0, $0, 0x300c
ori $s3, $0, 0x3010
ori $s4, $0, 8
jal loop
nop
sw $ra, 4($s4)
addu $s0, $s0, $s4
lui $t1, 8
loop:
addu $t2, $t2, $ra
beq $ra, $s0, branch
nop
jr $ra
nop
branch:
lui $t3, 12
# beq的另一种情形，完整的jal和jr组合？

jal label
addu $31, $31, $31
ori $t2, $1, 0xff00
label:
addu $2, $31, $31
addu $3, $31, $31
addu $4, $31, $31
addu $5, $31, $31
ori $t3, $0, 20
# E_pc8 M_pc8转D_rs D_rt
```

转发检测		接受转发位点				
		M_rt	E_rs	E_rt	D_rs	D_rt
转发数据	W_dm	ok	ok	ok	ok	ok
	W_alu	ok	ok	ok	ok	ok
	W_pc8	ok	ok	ok	ok	ok
	M_alu	/	ok	ok	ok	ok
	M_pc8	/	ok	ok	ok	ok
	E _pc8	/	/	/	ok	ok

加载代码到ise的code.txt中

```
system("java -jar Mars_perfect.jar db nc mc CompactDataAtZero 8000 dump .text
HexText P5_LO_CPU/code.txt self_test.asm > msg_out_mars.txt");
```

在mips_tb.v相应模块中进行文件操作，导出文件后在进行比较

```
void fileCmp(){
    int cnt=0, flag = 1;
    string tmp_cpu,tmp_mars;

    ifstream cpufile("msg_out_cpu.txt");
```

```
    ifstream marfile("msg_out_mars.txt");
    ofstream cmpfile("msg_cmp.txt");

    while(getline(cpufile,tmp_cpu) && getline(marfile,tmp_mars)){
        cnt++;
        if(tmp_cpu != tmp_mars){
            flag = 0;
            cmpfile << "mismatch - line" << cnt << "\t: ";
            cmpfile << "[cpu] " << tmp_cpu << "\t[mars] " << tmp_mars << endl;
        }
    }
    if(flag) cmpfile << "no error";
}

int main(){
    fileCmp();
    return 0;
}
```

（二）自动批量测试

使用了讨论区中共三组测试数据

并搭配了自动批量测试文件

```
import os

mars_dir = 'D:\\0_Personal_File\\Grade2_01_Autumn\\CO\\P5\\Mars_perfect.jar'
hexcode_dir = 'D:\\0_Personal_File\\Grade2_01_Autumn\\CO\\P5\\code.txt'
mipscode_dir = ''
standard_outdir =
'D:\\0_Personal_File\\Grade2_01_Autumn\\CO\\P5\\ans_out_mars.txt'

walk =
os.walk('D:\\0_Personal_File\\Grade2_01_Autumn\\CO\\P5\\P5tester\\analysis\\work
\\P5_data')
fcmp = open('ans_cmp.txt','w')
epoch = 0
for path1, docu_list, file_list in walk:
    for file_name in file_list:
        if('.asm' in file_name):
            mipscode_dir = os.path.join(path1, file_name)

            print(epoch + 1,mipscode_dir)

            epoch = epoch + 1
            os.system('java -jar ' + mars_dir + ' ' + mipscode_dir + ' nc mc
CompactDataAtZero a dump .text HexText ' + hexcode_dir) #编译出十六进制文件
            os.system('java -jar ' + mars_dir + ' ' + mipscode_dir + ' db nc mc
CompactDataAtZero 8000 >' + standard_outdir)

            print('mars finished')

            #进行编译
            os.environ['XILINX'] = 'D:\\Xilinx\\14.7\\ISE_DS\\ISE'
```

```
            fuse_dir = 'D:\\Xilinx\\14.7\\ISE_DS\\ISE\\bin\\nt64\\fuse'
            prj_dir =
'D:\\0_Personal_File\\Grade2_01_Autumn\\CO\\P5\\P5_L0_CPU\\mips_tb_beh.prj'
            tcl_dir =
'D:\\0_Personal_File\\Grade2_01_Autumn\\CO\\P5\\P5_L0_CPU\\conf.tcl'

            print('ise file prepared')

            #使用编译产生的可执行文件，输出CPU的答案
            os.system(fuse_dir + ' -nodebug -prj ' + prj_dir + ' -o testX.exe
mips_tb > log.txt')
            os.system('testX.exe -nolog -tclbatch '+ tcl_dir + '>
ans_out_cpu.txt')

            print('ise finished')

            #比对两个文件
            temp = open('ans_out_cpu.txt', 'r')
            useful = temp.readlines()[8:]
            newtemp = open('ans_out_cpu_new.txt', 'w')
            newtemp.writelines(useful)
            temp.close()
            newtemp.close()

            temp = open('ans_out_mars.txt', 'r')
            useful = temp.readlines()
            while useful[0][0:9] < '@00003000' :
                del useful[0]
            newtemp = open('ans_out_mars_new.txt', 'w')
            newtemp.writelines(useful)
            temp.close()
            newtemp.close()

            file1 = open('ans_out_cpu_new.txt')
            l1 = file1.readlines()
            file2 = open('ans_out_mars_new.txt')
            l2 = file2.readlines()

            new_l1 = []
            new_l2 = []

            for i in range(len(l1)):
                if(l1[i][0] == '@'):
                    new_l1.append(l1[i])
            for i in range(len(l2)):
                if(l2[i][0] == '@'):
                    new_l2.append(l2[i])

            same = True
            if(len(new_l1) != len(new_l2)):
                same = False
            else:
                for i in range(len(new_l1)):
                    if(new_l1[i].strip() != new_l2[i].strip()):
                        print(i,'\n', new_l1[i], new_l2[i])
                        same = False
            if(same == False):
```

```
                print('Failure in: ', file_name, '  epoch: {}, result =
{}'.format(epoch, same))
                fcmp.write('Failure in: ' + file_name + '  epoch: {}, result =
{}\n'.format(epoch, same))
                break

            print('epoch: {}, result = {}'.format(epoch, same))
            fcmp.write('epoch: {}, result = {}\n'.format(epoch, same))

fcmp.close()
```

三、 思考题

（一）流水线冒险

1.在采用本节所述的控制冒险处理方式下，PC 的值应当如何被更新？请从数据通路和控制信号两方面进行说明。

数据通路： F 级和 D 级相连，由 F 级提供当前 PC 寄存器值，由 D 级提供下一个 PC 值所需的其他值，并由 NPC 输出下一个 PC 值。注意由于寄存器的分割，F和D级的PC不同，应遵从F级PC设置相应NPC

控制信号：D级控制器解码出NPCOp，控制NPC输出

2.对于 jal 等需要将指令地址写入寄存器的指令，为什么需要回写 PC+8？

默认支持延迟槽。

（二）数据冒险的分析

1.为什么所有的供给者都是存储了上一级传来的各种数据的流水级寄存器，而不是由 ALU 或者 DM 等部件来提供数据?

组合逻辑的运算延迟各不相同，CPU流水线各级延迟不均衡，将导致流水线性能下降，同时也不利于计算设置流水线时钟频率。

（三）AT 法处理流水线数据冒险

1.“转发（旁路）机制的构造”中的 Thinking 1-4;

Thinking 1：如果不采用已经转发过的数据，而采用上一级中的原始数据，会出现怎样的问题？试列举指令序列说明这个问题。

```
lw $4, 0($0)
sw $4, 0x2000($0)
sw $4, 0x2004($0)
sw $4, 0x2008($0)
sw $4, 0x200c($0)
```

被转发的数据都是将要写入但还没写入GRF的数据。因此是对错误的`GPR[rs]` `GPR[rt]`值进行修正，这两个寄存器值的数据路径分别是

- `GPR[rs]`：D_GRF_RD1 -> E_reg_V1
- `GPR[rt]`：D_GRF_RD2 -> E_reg_V2 -> M_reg_V2

当某指令到达某一级，必须使用指定寄存器值时，该级路径上的内容必须正确(接收转发)，这也是T_use计算截止的地方。若能达成上述目标（T_use >= T_new），则T_use截止处前的内容无需关心，但这不代表此前转发的值没用。若不将转发后的数据流，当转发源消失后，便得不到正确信息

Thinking 2：我们为什么要对 GPR 采用内部转发机制？如果不采用内部转发机制，我们要怎样才能解决这种情况下的转发需求呢？

RFA3和A1或A2相同且不为0时，直接选取RFWD的内容作为RD1或RD2的内容输出。不内部转发的话，只好从W级转发数据到D级的RD端，那样子挺蠢。

Thinking 3：为什么 0 号寄存器需要特殊处理？

对0号寄存器的写入无效，若不特殊处理转发将写入0号寄存器的数据则会导致错误

Thinking 4：什么是“最新产生的数据”？

离转发位点最近流水级处的数据。

2.在 AT 方法讨论转发条件的时候，只提到了“供给者需求者的A相同，且不为 0”，但在 CPU 写入 GRF 的时候，是有一个 we 信号来控制是否要写入的。为何在 AT 方法中不需要特判 we 呢？为了用且仅用 A 和 T 完成转发，在翻译出 A 的时候，要结合 we 做什么操作呢?

只有要写寄存器的指令才有 Tnew，需要寄存器值的指令才有 Tuse，因而不必特判 we 信号来确定；判断当前是否需要阻塞。

（四）在线测试相关说明

在本实验中你遇到了哪些不同指令类型组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的?

```
                "cal_rr <~~ cal_rr": 86.66666666666666,
                "cal_rr <~~ cal_ri": 100.0,
                "cal_rr <~~ load": 90.0,
                "cal_rr <~~ lui": 86.66666666666666,
                "cal_rr <~~ jal": 93.33333333333334,
                "cal_ri <~~ cal_rr": 73.33333333333333,
                "cal_ri <~~ cal_ri": 86.66666666666666,
                "cal_ri <~~ load": 80.0,
                "cal_ri <~~ lui": 73.33333333333333,
                "cal_ri <~~ jal": 86.66666666666666,
                "br_r2 <~~ cal_rr": 90.0,
                "br_r2 <~~ cal_ri": 100.0,
                "br_r2 <~~ load": 100.0,
                "br_r2 <~~ lui": 100.0,
                "br_r2 <~~ jal": 100.0,
                "load <~~ cal_rr": 86.66666666666666,
                "load <~~ cal_ri": 100.0,
                "load <~~ load": 100.0,
                "load <~~ lui": 73.33333333333333,
                "load <~~ jal": 73.33333333333333,
                "store <~~ cal_rr": 93.33333333333334,
                "store <~~ cal_ri": 100.0,
                "store <~~ load": 100.0,
                "store <~~ lui": 73.33333333333333,
                "store <~~ jal": 100.0,
                "jr <~~ cal_rr": 80.0,
                "jr <~~ cal_ri": 100.0,
```

```
                "jr <~~ load": 100.0,
                "jr <~~ jal": 100.0
```

如果你是手动构造的样例，请说明构造策略，说明你的测试程序如何保证**覆盖**了所有需要测试的情况；如果你是**完全随机**生成的测试样例，请思考完全随机的测试程序有何不足之处；如果你在生成测试样例时采用了**特殊的策略，**比如构造连续数据冒险序列，请你描述一下你使用的策略如何**结合了随机性**达到强测的效果。

此思考题请同学们结合自己测试 CPU 使用的具体手段，按照自己的实际情况进行回答。