计算机组成原理实验_CPU 综合设计实验

姓名:zml　学号：PB14011007　学院：计算机学院

一、实验目的：

深入学习 CPU 的设计，学会用流水线设计一个至少 16 条指令的 CPU，了解流水线、中断等的原理，巩固本学期的知识点。

二、实验内容：

实验检查提供基本的 16 条指令的测试例：

基本 16 条指令：

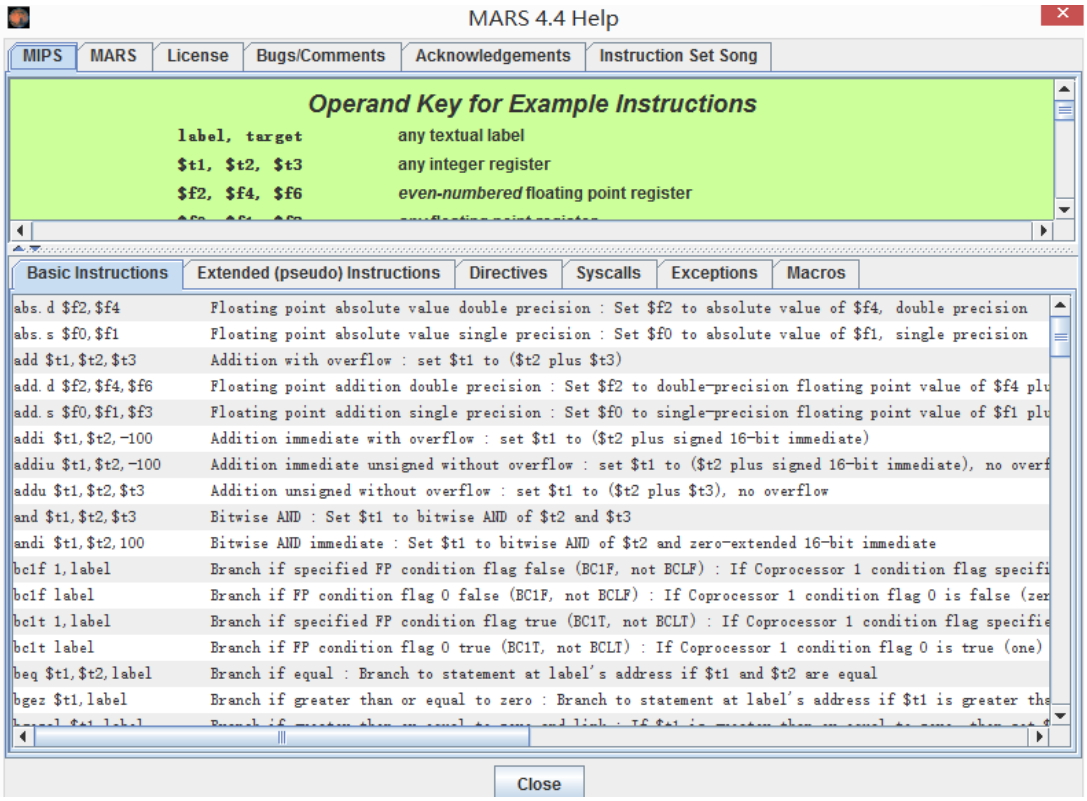
add addi addu　sub subu

and andi or nor xor

bgtz bne j jr

lw sw

额外增加指令需自行提供测试例，所有 MARS 能够仿真的指令均是可选指令范围见 help 部分：



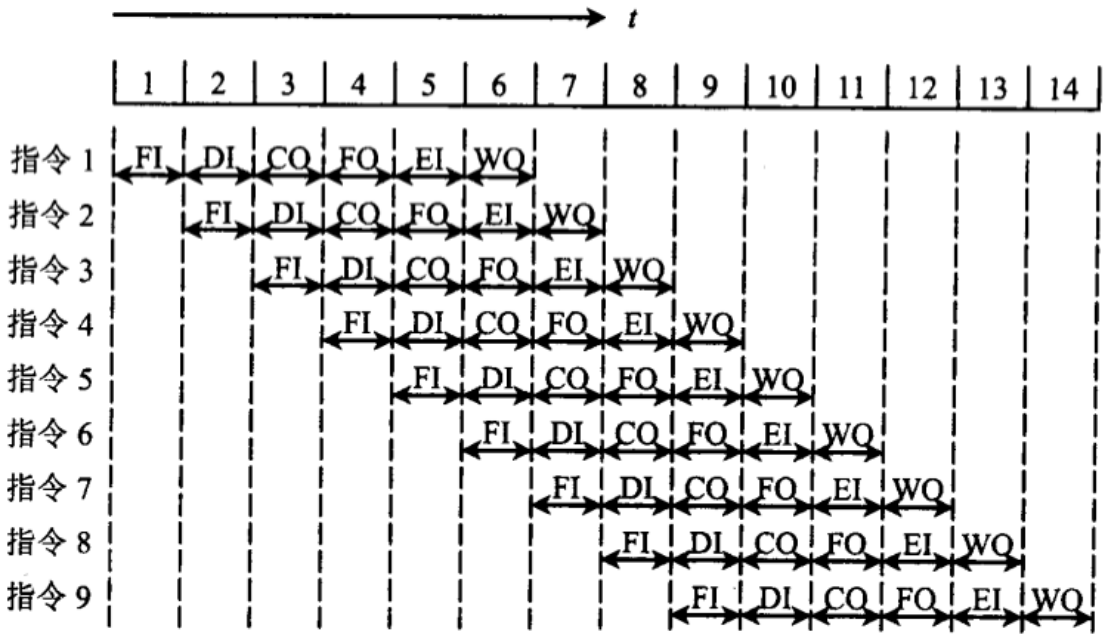
，指令译码以 MARS 为准。

检查需提供以下内容：汇编源代码+MARS 仿真结果+CPU 运行波形+结果等。

三、实验分析:

1.实验原理：

✓ 流水线技术：指令流水线是指将指令的整个执行过程用流水线进行分段处理，如下图六级流水时序。



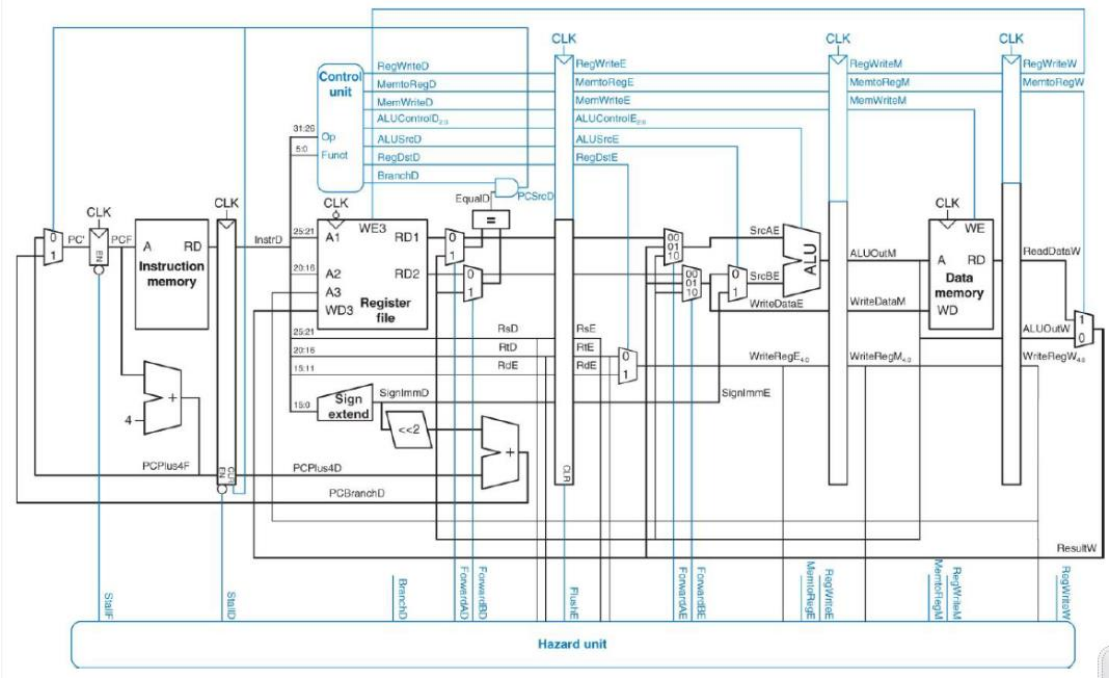
✓ 流水线的相关性处理：

（1）结构相关，即不同指令产生资源冲突，常见的是访存冲突。一种解决方案是在完成前一条指令对数据的存储器访问时，暂停（一个时钟周期）取后一条指令的操作（bubble）；另一种方式是采用哈佛结构，设置两个独立的存储器分别存放操作数和指令。

（2）数据相关，是指流水线中的各条指令因重叠操作，可能改变对操作数的读写访问顺序，从而导致了数据相关冲突。一种解决方案是通过 stall 来将相关指令延迟；另一种是通过指令调度。

（3）控制相关，由转移指令引起，可以通过分支预测或延迟分支解决。

2.数据通路：



✧ 其中，控制单元(Ctrl Unit)定义如下：

（1）MemToRegD：为 1 时，选择存储器数据；为 0 时，选择 ALU 输出的数据；

（2）BranchD：与 EqualD 共同形成 PCSrcD 控制信号；

（3）MemWriteD：为 1 时写入存储器。存储器地址由 ALU 的输出决定，写入数据为寄存器 rt 的内容；

（4）ALUControl[2:0]：ALU 控制码；

（5）ALUSrcD：ALU 操作数 B 的选择，为 1 时，选择扩展的立即数；为 0 时，选择寄存器数据；

（6）RegWriteD：为 1 时写入寄存器堆，目的寄存器号是由 REGDES 选出的 rt 或 rd，写入数据是由 MemToReg 选出的存储器数据或 ALU 的输出结果；

（7）RegDstD：目的地址，为 1 时，选择 rd；为 0 时，选择 rt。

✧ 冲突处理单元（Hazard unit）定义如下：

（1）StallF、StallD:用以处理控制相关和数据相关；

（2）ForwardAD[1:0]:处理控制相关，为 1 时选择 ALU 的输出结果，为 0 时选择寄存器输出数据，否则选择 ResultW；

（3）ForwardBD[1:0]:处理控制相关，为 1 时选择 ALU 的输出结果，为 0 时选择寄存器输出数据，否则选择 ResultW；

（4）FlushE:PCSrcD 为 1 时发生跳转数据清零，为 0 时也选择 0；

（5）ForwardAE[1:0]:处理结构相关，为 00 时选择 RD1_out，为 01 时选择 ResultW, 为 10 时选择 ALU 的输出结果；

（6）ForwardBE[1:0]:处理结构相关，为 00 时选择 RD2_out，为 01 时选择 ResultW，为 10 时选择 ALU 的输出结果。

✧ 总体而言，数据通路分为五个部分，分别是取指（IF）、译码（ID）、执行（EX）、

访存（MEM）、写回（WB），由四个寄存器文件（D_reg、E_reg、M_reg、W_reg）区分进行各控制信号的传递。

3.相关指令：

(一) Op

（1）add 000000 rs rt rd shamt=00000 funct=100000
符号加 rs+rt—>rd

（2）addi 001000 rs rt immediate
符号加立即数 rs+(signextend)immediate -> rt

（3）addu 000000 rs rt rd shamt=00000 funct=100001
无符号加 rs+rt—>rd

（4）sub 000000 rs rt rd shamt=00000 funct=100010
符号减 rs+rt—>rd

（5）subu 000000 rs rt rd shamt=00000 funct=100011
无符号减 rs+rt—>rd

（6）and 000000 rs rt rd shamt=00000 funct=100100
与 rs&rt—>rd

（7）andi 001100 rs rt immediate
与立即数 rs&immediate—>rd

（8）or 000000 rs rt rd shamt=00000 funct=100101
或 rs|rt—>rd

（9）nor 000000 rs rt rd shamt=00000 funct=100111
或非 rs nor rt—>rd

（10）xor 000000 rs rt rd shamt=00000 funct=100110
异或 ts^rt—>rd

（11）bgtz 000111 rs rt=00000 offset
rs 中的值大于 0 时跳转，跳转值为 offset

（12）bne 000101 rs rt offset
不等时转移 if(rs!=rt),then(PC<—PC+4+sign_extend(offset||0^2))

（13）j 000010 instr_index
(PC+4)[31:28],address,0,0 -> PC (26 位扩展为 32 位)

（14）jr 000000 rs 0000000000 00000 001000

跳转到寄存器　　rs—>PC

（15）lw　　100011　rs　　rt　immediate

加载字　　memory[rs + (signextend)immediate] -> rt

（16）sw　　101011　rs　rt　immediate

储存　　rt -> memory[rs + (signextend)immediate]

（17）扩展的其余指令参考提供的<Verilog-testbench 的写法>

包括 sllv(逻辑可变左移)、srav（算术可变右移）、srlv（逻辑可变右移）、slt（有符号小于置 1）、sltu（无符号小于置 1）、ori（或立即数）、xori（异或立即数）、lui（立即数加载至高位）、addiu（无符号加立即数）、beq（相等时转移）、bgez（大于等于 0 时转移）、bltz（小于 0 时转移）、blez（小于等于 0 时转移）、slti（有符号小于立即数置 1）、sltiu（无符号小于立即数置 1）、nop（空操作）。

(二) 上述指令分为以下几类：

（1）R_R 运算

ADD	加	0/20H	GPR[rd] = GPR[rs] + GPR[rt]
ADDU	无符号加	0/21H	GPR[rd] = GPR[rs] + GPR[rt]
SUB	减	0/22H	GPR[rd] = GPR[rs] - GPR[rt]
SUBU	无符号减	0/23H	GPR[rd] = GPR[rs] - GPR[rt]

SLT	小于置 1	0/2AH	GPR[rd] = (GPR[rs] < GPR[rt]) ? 1:0
SLTU	小于置 1 (无符号)	0/2BH	GPR[rd] = (GPR[rs] < GPR[rt]) ? 1:0

SLLV	逻辑可变左移	0/4H	GPR[rd] = {GPR[rt][31-v:0], v{0}}
SRLV	逻辑可变右移	0/6H	GPR[rd] = {v{0}, GPR[rt][31:v]}
SRAV	算术可变右移	0/7H	GPR[rd] = {v{GPR[rt][31]}, GPR[rt][31:v]}
AND	与	0/24H	GPR[rd] = GPR[rs] & GPR[rt]
OR	或	0/25H	GPR[rd] = GPR[rs] | GPR[rt]
XOR	异或	0/26H	GPR[rd] = GPR[rs] ^ GPR[rt]
NOR	或非	0/27H	GPR[rd] = ~(GPR[rs] | GPR[rt])

（2）R_I 运算

ADDI	加立即数	8H	GPR[rt] = GPR[rs] + SignExt(Imm)

ADDIU	加立即数(无符号)	9H	GPR[rt] = GPR[rs] + SignExt(Imm)
ANDI	与立即数	CH	GPR[rt] = GPR[rs] & ZeroExt(Imm)
ORI	或立即数	DH	GPR[rt] = GPR[rs] | ZeroExt(Imm)
XORI	异或立即数	EH	GPR[rt] = GPR[rs] ^ ZeroExt(Imm)
LUI	立即数加载至高位	FH	GPR[rt] = {imm, 16'b0}
SLTI	小于立即数置 1	AH	GPR[rt] = (GPR[rs] < SignExt(Imm)) ? 1 : 0
SLTIU	小于立即数置 1(无符号)	BH	GPR[rt] = (GPR[rs] < SignExt(Imm)) ? 1 : 0

（3）lw/sw

LW	加载字	23H	R[rt] = Mem[GPR[rs]+sign_ext(offset)]
SW	存储字	2BH	Mem[GPR[rs]+sign_ext(offset)] = R[rt]

（4）分支

BEQ	等于转移	4H	if (GPR[rs] == GPR[rt]) PC = PC + 4 + BranchAddr
BNE	不等转移	5H	if (GPR[rs] != GPR[rt]) PC = PC + 4 + BranchAddr
BLEZ	小于等于 0 转移	6H	if (GPR[rs] <= 0) PC = PC + 4 + BranchAddr
BGTZ	大于 0 转移	7H	if (GPR[rs] > 0) PC = PC + 4 + BranchAddr
BLTZ	小于 0 转移	特殊编码①	if (GPR[rs] <0) PC = PC + 4 + BranchAddr
BGEZ	大于等于 0 转移	特殊编码②	if (GPR[rs] >= 0) PC = PC + 4 + BranchAddr

（4）跳转

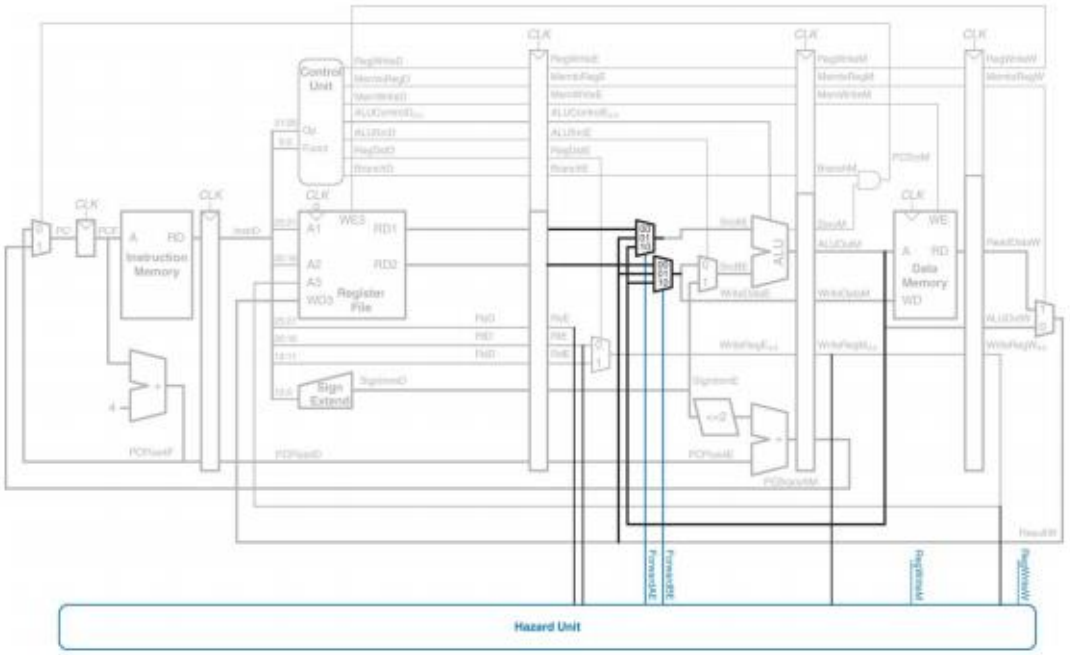
J	跳转	2H	PC = JumpAddr
JR	跳转寄存器	0/8H	PC = GPR[rs]

4.实验过程分析

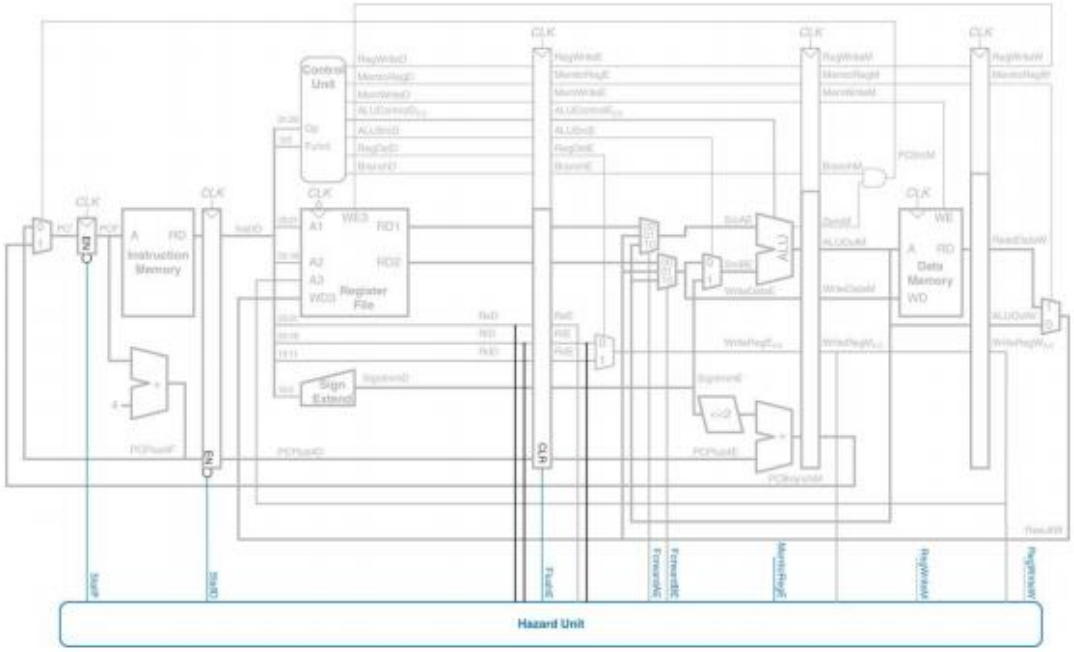
a.例化两个 IP 核，一个存放指令，一个存放数据，初始 coe 文件中的值同单周期 CPU 实验中的的指令和数据（求斐波那契数列，记作测试例 1），后期编写完成后选择其他测试案例检测（记作测试例 2）；

b.根据数据通路完成各模块的编写，其中冲突处理单元 HU 中——

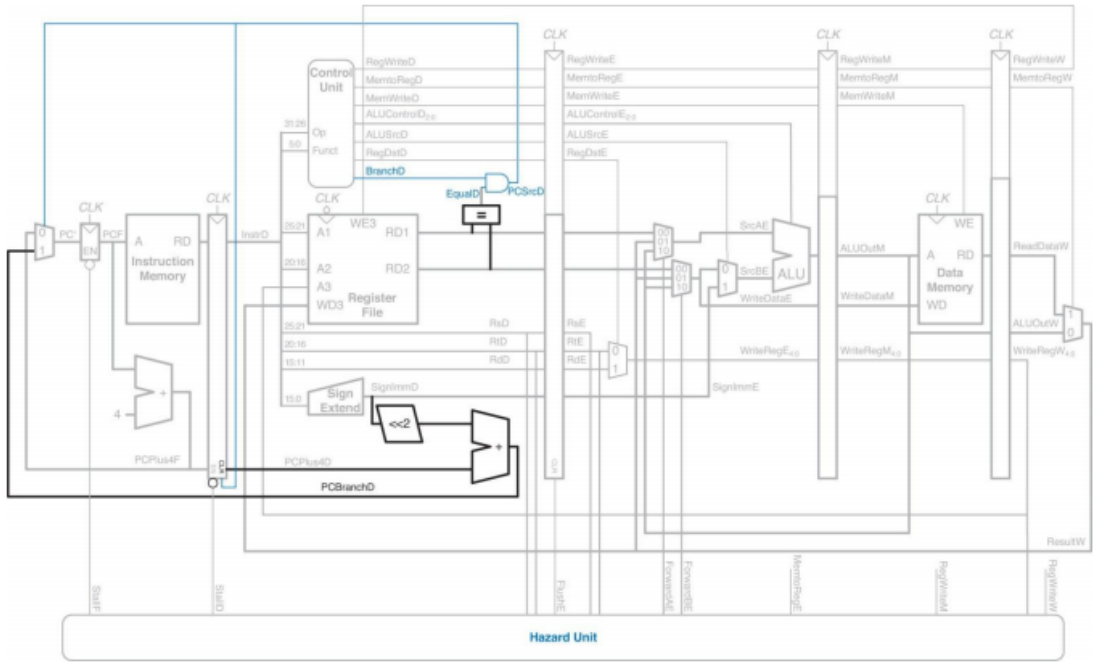
处理结构相关的控制信号如下：

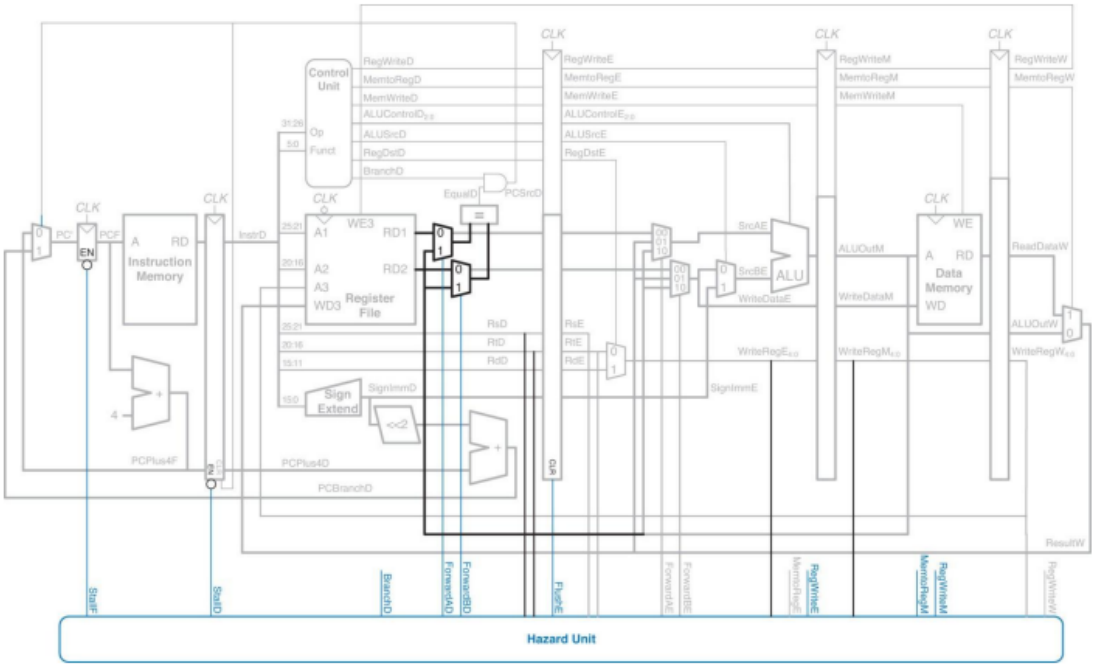


处理数据相关的控制信号如下：



处理控制相关（跳转指令）的信号如下：





其余各模块的设计说明见附件：CPU 代码设计报告

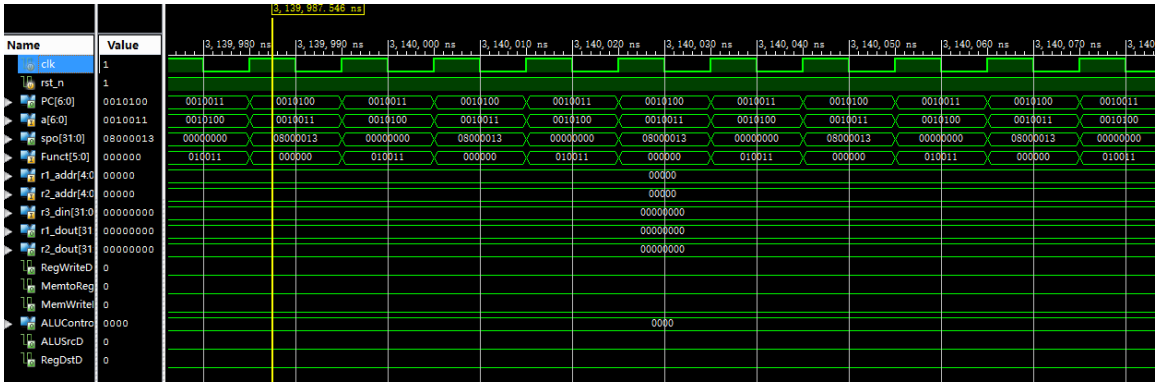
c.顶层代码进行连线（主要分五个阶段进行）

```
//*****************************************IF*****************************************//
  PCSrcD_MUX PCSrcD_Mux(PCSrcD,PCF+7'b0000001,PCBranchD,insD[6:0],r1_out[6:0],PCin);
  PC_Reg PC_Reg(clk,rst_n,PCin,StallF,PCF);
  ROM ROM(.a(PCF),.spo(ins));

//*****************************************ID*****************************************//
  D_Reg D_Reg(clk,ins,StallD,PCF+7'd1,PCSrcD,insD,PC_plus1D);
  CU CU(insD[31:26],insD[5:0],rst_n,RegWriteD,MemtoRegD,MemWriteD,ALUSrcD,RegDstD,ALUControlD);
  RegFile Reg(~clk,rst_n,insD[25:21],insD[20:16],WriteRegW,ResultW,RegWriteW,RD1,RD2);
  ForwardAD_MUX ForwardAD_Mux(ForwardAD,ALUOutM,RD1,ResultW,r1_out);
  ForwardBD_MUX ForwardBD_Mux(ForwardBD,ALUOutM,RD2,ResultW,r2_out);
  PCSrc_BranchD PCSrc_BranchD(rst_n,r1_out,r2_out,insD[31:26],insD[5:0],insD[20:16],StallF,PCSrcD,BranchD);
  PC_Plus PC_Plus(PC_plus1D,insD[6:0],PCBranchD);

//*****************************************EX*****************************************//
  E_Reg E_Reg(clk,FlushE,RegWriteD,MemtoRegD,MemWriteD,ALUSrcD,RegDstD,ALUControlD,RD1,RD2,insD[25:21],insD[
  RegDstE_MUX RegDstE_Mux(RtE,RdE,RegDstE,WriteRegE);
  ForwardAE_MUX ForwardAE_Mux(RD1_out,ALUOutM,ResultW,ForwardAE,SrcAE);
  ForwardBE_MUX ForwardBE_Mux(RD2_out,ALUOutM,ResultW,ForwardBE,RD2_outE);
  ALUSrcE_MUX ALUSrcE_Mux(RD2_outE,extend_out,ALUSrcE,SrcBE);
  ALU ALU(SrcAE,SrcBE,ALUControlE_modify,alu_out);

//*****************************************MM*****************************************//
  M_Reg M_Reg(clk,RegWriteE,MemtoRegE,MemWriteE,alu_out,RD2_outE,WriteRegE,RegWriteM,MemtoRegM,MemWriteM,ALU
  RAM Ram(ALUOutM[8:2],WriteDataM,MemWriteM,clk,RD);

//*****************************************WB*****************************************//
  W_Reg W_Reg(clk,RegWriteM,MemtoRegM,RD,ALUOutM,WriteRegM,RegWriteW,MemtoRegW,ReadDataW,ALUOutW,WriteRegW);
  MemtoRegW_MUX MemtoRegW_Mux(MemtoRegW,ReadDataW,ALUOutW,ResultW);

//Hazard unit
  HU HU(.rst_n(rst_n),.BranchD(BranchD),.RsD(insD[25:21]),.RtD(insD[20:16]),.RtE(RtE),.RsE(RsE),.PCSrcD(PCSr
```

d.进行仿真和单步跟踪比较结果

四、实验结果：

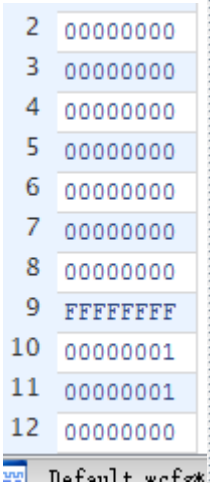
成功用 verilog HDL 编写出流水线 CPU，并编写仿真文件完成仿真：



其中采用测试例 1 得到正确结果：

	0	0	0	0	0	0	0
	0	0	0	0	0	0	0
	0	0	0	0	0	0	0
	0	0	0	0	0	0	0
	3	3	20	20295	12543	7752	4791
	1830	1131	699	432	267	165	102
	39	24	15	9	6	3	3

采用测试例 2 单步跟踪与 Mars 编译结果相同：



2	00000000
3	00000000
4	00000000
5	00000000
6	00000000
7	00000000
8	00000000
9	FFFFFFFF
10	00000001
11	00000001
12	00000000

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0xffffffff
$t2	10	0x00000001
$t3	11	0x00000001
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000

五、附录：

附件 1.CPU 代码设计报告
附件 2.仿真测试流程
附件 3.工程文件
附件 4.测试代码 2 例