

（深圳）

# 课程报告

开课学期： 2022夏季

课程名称： 计算机设计与实践

项目名称： 基于miniRV-1的SoC设计

项目类型： 综合设计型

课程学时： 56 地点： T2210

学生班级： 4班

学生学号： 20011xxxx

学生姓名： 代码不自由

评阅教师：

报告成绩：

实验与创新实践教育中心制

2022年7月

注：本设计报告中各个部分如果页数不够，请同学们自行扩页。原则上一定要把报告写详细，能说明设计的成果、特色和过程。报告应该详细叙述整体设计，以及设计中的每个模块。设计报告将是评定每个人成绩的重要组成部分（**设计内容及报告写作**都作为评分依据）。

|  |
| --- |
| 设计的功能描述（含所有实现的指令描述，以及单周期/流水线CPU频率） |
| 实现的18条指令如下：  【注释：(r)表示寄存器r的值，Mem[addr]表示地址为addr的存储单元的值，sext(a)表示a的符号扩展】   1. add rd, rs1, rs2：   加(rd)🡨(rs1)+(rs2)   1. sub rd, rs1, rs2:   减(rd)🡨(rs1)-(rs2)   1. and rd, rd, rs1,rs2:   与(rd)🡨(rs1)&(rs2)   1. or rd, rs1, rs2:   或(rd)🡨(rs1)|(rs2)   1. xor rd, rs1, rs2:   异或(rd)🡨(rs1)^(rs2)   1. sll rd, rs1, rs2:   左移(rd)🡨(rs1)<<(rs2)   1. srl rd, rs1, rs2:   逻辑右移(rd)🡨(rs1)>>(rs2)   1. sra rd, rs1, rs2:   算术右移(rd)🡨($signed(rs1))>>(rs2)   1. addi rd, rs1, imm:   加立即数(rd)🡨(rs1)+sext(imm)   1. andi rd, rs1, imm:   与立即数(rd)🡨(rs1)&sext(imm)   1. ori rd, rs1, imm:   或立即数(rd)🡨(rs1)|sext(imm)   1. xori rd, rs1, imm:   异或立即数(rd)🡨(rs1)^sext(imm)   1. slli rd, rs1, shamt:   左移立即数位(rd)🡨(rs1)<<shamt   1. srli rd, rs1, shamt:   逻辑右移立即数位(rd)🡨(rs1)>>shamt   1. srai rd, rs1, shamt:   算数右移立即数(rd)🡨 ($signed(rs1))>>shamt   1. lw rd, offset(rs1):   取数(rd)🡨sext(Mem[(rs1)+sext(offset)][31:0]   1. jalr rd, offset(rs1):   跳转t🡨(pc)+4;(pc)🡨((rs1)+sext(offset))&~1;(rd)🡨t（将原来的pc+4的值写入寄存器rd，将pc设置为(rs1)+sext(offset)，把计算出的地址的最低位设为0）   1. sw rs2, offset(rs1):   存数Mem[(rs1)+sext(offset)]🡨(rs2)[31:0]   1. beq rs1, rs2, offset:   相等则跳转if((rs1)==(rs2))(pc)🡨(pc)+sext(offset)   1. bne rs1, rs2, offset:   不等则跳转if((rs1)≠(rs2))(pc)🡨(pc)+sext(offset)   1. blt rs1, rs2, offset:   小于则跳转if((rs1)<(rs2))(pc)🡨(pc)+sext(offset)（有符号数）   1. bge rs1, rs2, offset:   大于等于则跳转if((rs1)≥(rs2))(pc)🡨(pc)+sext(offset)（有符号数）   1. lui rd, imm:   取长立即数(rd)🡨sext(imm[31:12]<<12)   1. jal rd, offset:   跳转(rd)🡨(pc)+4; (pc)🡨(pc)+sext(offset)  按照指导书,单周期时钟频率使用25mhz，流水线频率使用50mhz皆成功。单周期未做更高频率的尝试，流水线cpu频率成功上板最高为180mhz。 |
| 设计的主要特色（除基本要求以外的设计） |
| 流水线CPU设计中，对于不涉及访存的数据冒险，RAW冒险的三种情形均采用前递的方式解决，提高执行效率；对于涉及访存的数据冒险，采用先停顿后续指令一个时钟周期，再前递的方式解决；对于控制冒险，采用静态预测的方式解决，预测分支指令总是不跳转，若在分支指令到达执行阶段时发现预测错误，则将后续已传入错误指令的流水级清空，以在下一个时钟周期传入正确的指令。 |
| 资源使用、功耗数据截图（Post Implementation；含单周期、流水线2个截图） |
| 以下是示例，请贴自己的图。  单周期25mhz    流水线50mhz    流水线100mhz    流水线180mhz |

1 单周期CPU设计与实现

1.1 单周期CPU整体框图

|  |
| --- |
| 要求：无需画出模块内的具体逻辑，但要标出模块的接口信号名、模块之间信号线的信号名和位宽，以及说明每个模块的功能含义。 |
| 各个模块功能：  pc：更新pc  指令IM：指令寄存器  寄存器堆regfile：从此获取32个寄存器存储内容  控制单元control：给出控制信号  比较器BranchComp：获取相应比较结果  计算器ALU：获取相应计算结果  数据寄存器DM：根据所给地址获得内存内容  MUX\_WB：选择写回数据寄存器的内容 |

1.2 单周期CPU模块详细设计

|  |
| --- |
| 要求：画出各个模块的详细设计图，包含内部的子模块，以及关键性逻辑；标出子模块接口信号名、各信号线的信号名和位宽，并有详细的解释说明。 |
| next\_pc的赋值采用组合逻辑，当pcSel为0时表示跳转，被赋为计算出来的地址from\_alu；当pcSel为1时表示不跳转，被赋为pc4（恒为pc+4）。复位信号rst无效一个时钟周期后的时钟上升沿信号rst\_p无效。current\_pc的赋值采用时序逻辑，在每个时钟上升沿，当rst\_p有效时，被赋值为0；无效时，被赋值为next\_pc。以实现pc的更新。    控制信号的赋值均采用组合逻辑。  pcSel：根据opcode判断指令类型，当指令为R型指令、I型指令、LW指令、S型指令、U型指令时，指令不跳转，pcSel设为1；当指令为B型指令时，由比较器结果判断指令跳转与否，pcSel设为brAns；当指令为J型指令或jalr指令，指令跳转，pcSel设为0；其余情况默认指令不跳转，pcSel设为1。  regWEn：当指令为S型指令或B型指令时，不写寄存器，regWEn设为0，否则设为1。  immSel：根据opcode判断指令类型选择如何对立即数进行扩展。当指令为I型指令、LW指令、jalr指令时，immSel设为`I\_SEXT；当指令为S型指令时，immSel设为`S\_SEXT ；当指令为B型指令时，immSel设为`B\_SEXT；当指令为U型指令时，immSel设为`U\_SEXT；当指令为J型指令时，immSel设为`J\_SEXT；其余情况默认immSel设为3’b111。  brOp：当指令类型为B型指令时，比较器做何比较由funct3决定，直接将brOp设为funct3；其余情况默认brOp设为3’b111。  aSel：根据指令类型判断ALU第一个操作数。当指令类型为B型指令或J型指令时， aSel设为0；否则aSel设为1。  bSel：根据指令类型判断ALU第二个操作数。当指令类型为R型指令时， bSel设为0；否则bSel设为1。  aluSel：当指令类型为R型指令时，根据funct3和funct7判断计算器进行的操作类型，分别将aluSel设为`SUB、`ADD、`AND、`OR、`XOR、`SLL、`SRL、`SRA，其它情况默认为`NOOP；当指令类型为I型指令时，根据funct3和funct7判断计算器进行的操作类型，分别将aluSel设为`ADD、`AND、`OR、`XOR、`SLL、`SRL、`SRA，其它情况默认为`NOOP；当指令为LW指令、jalr指令、S型指令、B型指令、J型指令时，aluSel设为`ADD；当指令为U型指令时，aluSel设为`LUI；其余情况默认aluSel设为`NOOP。  memRW：根据指令类型判断读写内存。当指令类型为S型指令时需要写内存，memRW设为1；其余情况默认不写内存，设为0。  wbSel：根据指令类型选择写回寄存器的内容。当指令类型为J型指令、jalr指令时， wbSel设为`PC4；当指令为LW指令时， wbSel设为`FROM\_DM；其余情况默认wbSel设为`FROM\_ALU。    读寄存器采用时序逻辑，每个时钟下降沿来临时，若rR1不是0号寄存器，将其中存储的内容赋给rD1，否则将rD1设为0；若rR2不是0号寄存器，将其中存储的内容赋给rD2，否则将rD2设为0。  写寄存器采用时序逻辑，每个时钟上升沿来临时，0号寄存器reg\_array[0]恒设为0，复位信号有效时，所有寄存器设为0。不复位时，若写寄存器堆信号有效，即wE为1时，且目标寄存器不是0号寄存器，则将reg\_array[wR]设为wD。    根据控制信号对25位的数ins进行相应立即数扩展为32位的sext\_imm。当sext\_op为`I\_SEXT时，进行I型立即数扩展；当sext\_op为`S\_SEXT 时，进行S型立即数扩展；当sext\_op为`B\_SEXT，进行B型立即数扩展；当sext\_op为`U\_SEXT 时，进行U型立即数扩展；当sext\_op为`J\_SEXT 时，进行J型立即数扩展；其余情况默认设置sext\_imm为0。    brOp为`BEQ时，data1==data2则将brAns设为0，否则为1；为`BNE时，与`BEQ 相反；为`BLT时，data1<data2则将brAns设为0，否则为1；为`BGE时，与`BLT相反。    运算器采用组合逻辑。  aSel为0时，ALU第一个操作数a选择pc，为1时选择reg1存储的数。bSel为0时，ALU第二个操作数b选择reg2存储的数，为1时选择扩展后的立即数。当aluSel为`ADD时，ALU对两操作数进行加法操作（result=a+b）；当为`SUB时，做减法操作（result=a-b）；当为`AND时，做与操作（result=a&b）；当为`OR时，做或操作（result=a|b）；当为`XOR时，做异或操作（result=a^b）；当为`SLL时，做左移操作（result=a<<b[4:0]）；当为`SRL时，做逻辑右移操作（result=a>>b[4:0]）；当为`SRA时，做算数右移操作（result=($signed(a))>>>b[4:0]）；当为`LUI时，直接将b赋值给result（result=b）；其他情况默认将result直接赋值为0。    寄存器堆写回值dataW的赋值采用组合逻辑。当wbSel=`PC4时，选择pc+4；当wbSel=`FROM\_ALU时，选择ALU的计算结果；当wbSel=`FROM\_DM时，选择从DM中取出的结果。    数码管显示逻辑。Top模块获取数码管所需显示的数字data和进行时钟分频得到时钟信号clk\_1khz，传入shumaguan模块，由shumaguan模块计算出每个分频后时钟周期显示的数字num和数码管使能信号。 |

1.3 单周期CPU仿真及结果分析

|  |
| --- |
| 要求：包含逻辑运算、访存、分支跳转三类指令的仿真截图以及波形分析；每类指令的截图和分析中，至少包含1条具体指令；截图需包含信号名和关键信号。 |
| 4: 02500413 addi x8,x0,37  在时钟上升沿获得当前pc后，从IM取出指令instruction，分析指令。从寄存器堆中取出寄存器rR1，即0号寄存器的内容作为ALU的第一个操作数，即a=0；同时由sext\_op=0，判断将指令后25位做I型立即数扩展，得到32位的立即数sext\_imm作为ALU的第二个操作数，即b=0x25。两操作数在ALU中做加法操作得到结果，即from\_alu=0x25，并将该结果写回8号寄存器。    b00: 0000a703 lw x14,0(x1)  在时钟上升沿获得当前pc后，从IM取出指令instruction，分析指令。从寄存器堆中取出寄存器rR1，即1号寄存器的内容作为ALU的第一个操作数，即a=0x4000；同时由sext\_op=0，判断将指令后25位做I型立即数扩展，得到32位的立即数sext\_imm作为ALU的第二个操作数，即b=0。两操作数在ALU中做加法操作得到结果，即from\_alu=0x4000。该结果作为内存地址，取出该地址中存储的内容0xff00ff写入14号寄存器。    2a4: fc7716e3 bne x14,x7,270 <fail>  在时钟上升沿获得当前pc后，从IM取出指令instruction，分析指令。从寄存器堆中取出寄存器rR1，即14号寄存器的内容作为比较器的第一个操作数，即data1=0；从寄存器堆中取出寄存器rR2，即7号寄存器的内容作为比较器的第一个操作数，即data2=0；由brOp=1，判断两操作数是否相等，得判断结果相等，即brAns=1 ，说明指令顺序执行，不跳转，所以将pc4作为下条pc，即next\_pc=0x2a8，而非ALU得到的跳转地址0x270。 |

2 流水线CPU设计与实现

2.1 流水线的划分

|  |
| --- |
| 要求：画出流水线如何划分，说明每个流水级具备什么功能、需要完成哪些操作。 |
| 经典五级流水结构：  》取指 阶段（Instruction Fetch，IF）：是指将指令从指令存储器中读取出来的过程。  》译码 阶段（Instruction Decode，ID）：是指将取指阶段取出的指令进行翻译的过程。译码阶段将得到指令的操作码、功能码、操作数寄存器号等信息，然后从寄存器堆（Register File）取出操作数，同时产生指令执行所需的控制信号。  》执行 阶段（instruction EXecute，EX）：是指根据指令的操作码和功能码，对指令操作数进行运算的过程。如果指令是一条加法运算指令，则对操作数进行加法操作；如果是减法运算指令，则进行减法操作。CPU中的算术逻辑单元（Arithmetic Logical Unit，ALU）是实施具体运算的硬件功能单元，是执行阶段的核心部件。  》访存 阶段（MEMory access，MEM）：是指访存指令从存储器读出数据，或将数据写入存储器的过程。  》写回 阶段（Write Back，WB）：是指将指令执行结果写回到目标寄存器的过程。运算类指令的执行结果是执行阶段的输出，而访存指令的执行结果则是访存阶段从存储器读取出的数据。 |

2.2 流水线CPU整体框图

|  |
| --- |
| 要求：无需画出模块内的具体逻辑，但要标出模块的接口信号名、模块之间信号线的信号名和位宽，以及说明每个模块的功能含义。 |
| 各个模块功能：  pc：更新pc  指令IM：指令寄存器  寄存器堆regfile：从此获取32个寄存器存储内容  控制单元control：给出控制信号  比较器BranchComp：获取相应比较结果  计算器ALU：获取相应计算结果  数据寄存器DM：根据所给地址获得内存内容  MUX\_WB：选择写回数据寄存器的内容  IF/ID流水级、ID/EX流水级、EX/MEM流水级、MEM/WB流水级：4个用于暂存指令执行的中间结果的流水线寄存器  停顿判断逻辑：判断载入使用型数据冒险，并通过暂停和前递解决冒险  前递判断逻辑：判断不涉及LW指令的数据冒险，并通过前递解决冒险 |

2.3 流水线CPU模块详细设计

|  |
| --- |
| 要求：画出各个模块的详细设计图，包含内部的子模块，以及关键性逻辑；标出子模块接口信号名、各信号线的信号名和位宽，并有详细的解释说明；此外，必须结合模块图，详细说明数据冒险、控制冒险的解决方法。 |
| 当EX阶段正在执行取数指令(opcode\_2==`LW\_TYPE)，EX阶段的写回寄存器与ID阶段要读取的寄存器reg1/reg2相同(wreg\_2==reg1/reg2且rr1\_1/rr2\_1，其中rr1\_1/rr2\_1表示ID阶段要读取寄存器reg1/reg2)时，说明此时EX阶段和ID阶段的两条指令发生载入-使用型数据冒险，采用组合逻辑，赋stop1/stop2=1，而停顿信号stop=stop1|stop2，得知此时得停顿。pre11和pre21的赋值采用时序逻辑，复位信号无效时，在每个时钟上升沿将pre11<=stop1、pre21<=stop2。停顿一个时钟周期后，利用pre11或pre21的改变将stop变回0，以实现只停顿一个时钟周期。同时采用组合逻辑，赋停顿后续前递信号pre1\pre2为pre11\pre21，或着当MEM阶段正在执行取数指令(opcode\_3==`LW\_TYPE)，MEM阶段的写回寄存器与ID阶段要读取的寄存器reg1/reg2相同(wreg\_3==reg1/reg2且rr1\_1/rr2\_1，其中rr1\_1/rr2\_1表示ID阶段要读取寄存器reg1/reg2)时，将pre1\pre2赋为1。停顿操作在部分部件设计中说明。  代码如下：      当ID阶段正在读寄存器reg1/reg2（即rr1\_1/rr2\_1）时，若EX阶段的指令要写寄存器wreg\_2（即regWEn\_2），同时两寄存器是同一个（即reg1== wreg\_2/reg2== wreg\_2），则检测到RAW情形A，设rs1\_id\_ex\_hazard=1/ rs2\_id\_ex\_hazard=1；若MEM阶段的指令要写寄存器wreg\_3（即regWEn\_3），同时两寄存器是同一个（即reg1== wreg\_3/reg2== wreg\_3），则检测到RAW情形B，设rs1\_id\_mem\_hazard=1/ rs2\_id\_mem\_hazard=1；若WB阶段的指令要写寄存器wreg\_4（即regWEn\_4），同时两寄存器是同一个（即reg1== wreg\_4/reg2== wreg\_4），则检测到RAW情形C，设rs1\_id\_wb\_hazard=1/ rs2\_id\_wb\_hazard=1。都需要前递。  前递的具体操作：  pre1=1：将MEM阶段从DM中取来的数(from\_dm\_3)赋给data1\_1  pre2=1：将MEM阶段从DM中取来的数(from\_dm\_3)赋给data2\_1  rs1\_id\_ex\_hazard=1：将EX阶段ALU计算出的结果(from\_alu\_2)赋给data1\_1  rs2\_id\_ex\_hazard=1：将EX阶段ALU计算出的结果(from\_alu\_2)赋给data2\_1  rs1\_id\_mem\_hazard=1：将MEM阶段ALU计算出的结果(from\_alu\_3)赋给data1\_1  rs2\_id\_mem\_hazard=1：将MEM阶段ALU计算出的结果(from\_alu\_3)赋给data2\_1  rs1\_id\_wb\_hazard=1：将WB阶段ALU计算出的结果(from\_alu\_4)赋给data1\_1  rs2\_id\_wb\_hazard=1：将WB阶段ALU计算出的结果(from\_alu\_4)赋给data2\_1    next\_pc的赋值采用组合逻辑，当EX阶段传来的pcSel\_2为0时表示跳转，被赋为EX阶段计算出来的地址from\_alu\_2；当pcSel\_2为1时表示不跳转，被赋为pc4（恒为pc+4）。复位信号rst无效一个时钟周期后的时钟上升沿信号rst\_p无效。current\_pc的赋值采用时序逻辑，在每个时钟上升沿，当rst\_p有效时，被赋值为0；其它当停顿检测逻辑传来的信号有效，即stop=1时，保持；其他情况默认赋值为next\_pc。以实现pc的更新。    控制信号的赋值均采用组合逻辑。  》regWEn：当指令为S型指令或B型指令时，不写寄存器，regWEn设为0，否则设为1。  》immSel：根据opcode判断指令类型选择如何对立即数进行扩展。当指令为I型指令、LW指令、jalr指令时，immSel设为`I\_SEXT；当指令为S型指令时，immSel设为`S\_SEXT ；当指令为B型指令时，immSel设为`B\_SEXT；当指令为U型指令时，immSel设为`U\_SEXT；当指令为J型指令时，immSel设为`J\_SEXT；其余情况默认immSel设为3’b111。  》brOp：当指令类型为B型指令时，比较器做何比较由funct3决定，直接将brOp设为funct3；其余情况默认brOp设为3’b111。  》aSel：根据指令类型判断ALU第一个操作数。当指令类型为B型指令或J型指令时， aSel设为0；否则aSel设为1。  》bSel：根据指令类型判断ALU第二个操作数。当指令类型为R型指令时， bSel设为0；否则bSel设为1。  》aluSel：当指令类型为R型指令时，根据funct3和funct7判断计算器进行的操作类型，分别将aluSel设为`SUB、`ADD、`AND、`OR、`XOR、`SLL、`SRL、`SRA，其它情况默认为`NOOP；当指令类型为I型指令时，根据funct3和funct7判断计算器进行的操作类型，分别将aluSel设为`ADD、`AND、`OR、`XOR、`SLL、`SRL、`SRA，其它情况默认为`NOOP；当指令为LW指令、jalr指令、S型指令、B型指令、J型指令时，aluSel设为`ADD；当指令为U型指令时，aluSel设为`LUI；其余情况默认aluSel设为`NOOP。  》memRW：根据指令类型判断读写内存。当指令类型为S型指令时需要写内存，memRW设为1；其余情况默认不写内存，设为0。  》wbSel：根据指令类型选择写回寄存器的内容。当指令类型为J型指令、jalr指令时， wbSel设为`PC4；当指令为LW指令时， wbSel设为`FROM\_DM；其余情况默认wbSel设为`FROM\_ALU。  》rr1：根据指令类型判断是否需要读取寄存器reg1中的内容。当指令为R型指令、I型指令、LW型指令、jalr指令、S型指令、B型指令时，rr1设为1；当指令为U型指令、J型指令时，rr1设为0；其他情况默认设为0。  》rr2：根据指令类型判断是否需要读取寄存器reg2中的内容。当指令为R型指令、S型指令、B型指令时，rr2设为1；当指令为I型指令、LW型指令、jalr指令、U型指令、J型指令时，rr2设为0；其他情况默认设为0。    所有输出信号的更新均采用时序逻辑，在时钟上升沿来临时：  当复位信号有效，即rst=1时，或要清空流水级，即flush（EX阶段得到的pcSel）=0时，要将相应的寄存器清零，useful\_1设为’b0；pc\_1设为32'b0；reg1设为5'b0；reg2设为5'b0；wreg\_1设为5'b0；imm设为25'b0；opcode\_1设为7'b0；regWEn\_1设为1；immSel\_1设为'b111；brOp\_1设为'b111；aSel\_1设为1；bSel\_1设为1；aluSel\_1设为`NOOP；memRW\_1设为0；wbSel\_1设为`FROM\_ALU；rr1\_1设为0；rr2\_1设为0。  当来自停顿判断逻辑的停顿信号有效，即stop=1时，所有输出信号保持。  其它情况默认，useful\_1设为’b1；pc\_1设为pc；reg1设为instruction[19:15]；reg2设为instruction[24:20]；wreg\_1设为instruction[11:7]；imm设为instruction[31:7]；opcode\_1设为instruction[6:0]；regWEn\_1设为regWEn；immSel\_1设为immSel；brOp\_1设为brOp；aSel\_1设为aSel；bSel\_1设为bSel；aluSel\_1设为aluSel；memRW\_1设为memRW；wbSel\_1设为wbSel；rr1\_1设为rr1；rr2\_1设为rr2。    读寄存器采用组合逻辑，若rR1不是0号寄存器，将其中存储的内容赋给rD1，否则将rD1设为0；若rR2不是0号寄存器，将其中存储的内容赋给rD2，否则将rD2设为0。  写寄存器采用时序逻辑，每个时钟上升沿来临时，0号寄存器reg\_array[0]恒设为0，复位信号有效时，所有寄存器设为0。不复位时，若写寄存器堆信号有效，即wE为1时，且目标寄存器不是0号寄存器，则将reg\_array[wR]设为wD。    根据控制信号对25位的数imm进行相应立即数扩展为32位的sext\_imm\_1。当immSel\_1为`I\_SEXT时，进行I型立即数扩展；当immSel\_1为`S\_SEXT 时，进行S型立即数扩展；当immSel\_1为`B\_SEXT，进行B型立即数扩展；当immSel\_1为`U\_SEXT 时，进行U型立即数扩展；当immSel\_1为`J\_SEXT 时，进行J型立即数扩展；其余情况默认设置sext\_imm\_1为0。（与单周期设计相同）    所有输出信号的更新均采用时序逻辑，在时钟上升沿来临时：  当复位信号有效，即rst=1时，或要清空流水级，即flush（EX阶段得到的pcSel）=0时，要将相应的寄存器清零，useful\_2设为’b0；pc\_2设为32'b0； wreg\_2设为5'b0；sext\_imm\_2设为32'b0；data1\_2设为32'b0；data2\_2设为32'b0；opcode\_2设为7'b0；regWEn\_2设为0；brOp\_2设为'b111；aSel\_2设为1；bSel\_2设为1；aluSel\_2设为`NOOP；memRW\_2设为0；wbSel\_2设为`FROM\_ALU。  当来自停顿判断逻辑的停顿信号有效，即stop=1时，useful\_2设为’b0；pc\_2设为pc\_1； wreg\_2设为wreg\_1；sext\_imm\_2设为sext\_imm\_1；data1\_2设为data1\_1；data2\_2设为data2\_1；opcode\_2设为opcode\_1；regWEn\_2设为0；brOp\_2设为brOp\_1；aSel\_2设为aSel\_1；bSel\_2设为bSel\_1；aluSel\_2设为aluSel\_1；memRW\_2设为0；wbSel\_2设为wbSel\_1。  其余情况默认将useful\_2设为useful\_1；pc\_2设为pc\_1； wreg\_2设为wreg\_1；sext\_imm\_2设为sext\_imm\_1；data1\_2设为data1\_1；data2\_2设为data2\_1；opcode\_2设为opcode\_1；regWEn\_2设为regWEn\_1；brOp\_2设为brOp\_1；aSel\_2设为aSel\_1；bSel\_2设为bSel\_1；aluSel\_2设为aluSel\_1；memRW\_2设为memRW\_1；wbSel\_2设为wbSel\_1。    brOp\_2为`BEQ时，data1\_2==data2\_2则将brAns设为0，否则为1；为`BNE时，与`BEQ 相反；为`BLT时，data1\_2<data2\_2则将brAns设为0，否则为1；为`BGE时，与`BLT相反。  同时根据EX阶段所执行的指令类型，即opcode\_2判断pcSel，当指令为R型指令、I型指令、LW指令、S型指令、U型指令时，指令不跳转，pcSel\_2设为1；当指令为B型指令时，由比较结果判断指令跳转与否，pcSel\_2设为brAns；当指令为J型指令或jalr指令，指令跳转，pcSel\_2设为0；其余情况默认指令不跳转，pcSel\_2设为1。    运算器采用组合逻辑。  aSel为0时，ALU第一个操作数a选择pc，为1时选择reg1存储的数。bSel为0时，ALU第二个操作数b选择reg2存储的数，为1时选择扩展后的立即数。当aluSel为`ADD时，ALU对两操作数进行加法操作（result=a+b）；当为`SUB时，做减法操作（result=a-b）；当为`AND时，做与操作（result=a&b）；当为`OR时，做或操作（result=a|b）；当为`XOR时，做异或操作（result=a^b）；当为`SLL时，做左移操作（result=a<<b[4:0]）；当为`SRL时，做逻辑右移操作（result=a>>b[4:0]）；当为`SRA时，做算数右移操作（result=($signed(a))>>>b[4:0]）；当为`LUI时，直接将b赋值给result（result=b）；其他情况默认将result直接赋值为0。（与单周期设计相同）    所有输出信号的更新均采用时序逻辑，在时钟上升沿来临时：  当复位信号有效，即rst=1时，useful\_3设为’b0；pc\_3设为32'b0；from\_alu\_3设为32’b0；wreg\_3设为5'b0；data2\_3设为32'b0；opcode\_3设为7'b0；regWEn\_3设为0；memRW\_3设为0；wbSel\_3设为`FROM\_ALU。  其余情况默认将useful\_3设为useful\_2；pc\_3设为pc\_2；from\_alu\_3设为from\_alu\_2；wreg\_3设为wreg\_2；data2\_3设为data2\_2；opcode\_3设为opcode\_2；regWEn\_3设为regWEn\_2；memRW\_3设为memRW\_2；wbSel\_3设为wbSel\_2。    所有输出信号的更新均采用时序逻辑，在时钟上升沿来临时：  当复位信号有效，即rst=1时，useful\_4设为’b0；pc\_4设为32'b0；from\_alu\_4设为32’b0；from\_dm\_4设为32’b0；wreg\_4设为5'b0；regWEn\_4设为0； wbSel\_4设为`FROM\_ALU。  其余情况默认将useful\_4设为useful\_3；pc\_4设为pc\_3；from\_alu\_4设为from\_alu\_3；from\_dm\_4设为from\_dm\_3；wreg\_4设为wreg\_3；regWEn\_4设为regWEn\_3；wbSel\_4设为wbSel\_3。    寄存器堆写回值dataW的赋值采用组合逻辑。当wbSel=`PC4时，选择pc+4；当wbSel=`FROM\_ALU时，选择ALU的计算结果；当wbSel=`FROM\_DM时，选择从DM中取出的结果。（与单周期设计相同）    数码管显示逻辑。Top模块获取数码管所需显示的数字data和进行时钟分频得到时钟信号clk\_1khz，传入shumaguan模块，由shumaguan模块计算出每个分频后时钟周期显示的数字num和数码管使能信号。（与单周期设计相同） |

2.4 流水线CPU仿真及结果分析

|  |
| --- |
| 要求：包含控制冒险和数据冒险三种情形的仿真截图，以及波形分析。若仅实现了理想流水，则此处贴上理想流水的仿真截图及详细的波形分析。 |
| 控制冒险：  0: 0040006f jal x0,4 <reset\_vector>  4: 02500413 addi x8,x0,37  我采用静态预测解决控制冒险，总是预测跳转。图中当第一条指令到达EX阶段(pc\_2=0x0)时，得知下条指令该跳转(pcSel=0)，跳转目的地址为from\_alu\_2=0x4，预测错误，一个时钟周期后，pc=0x4，清空IF/ID流水级(pc\_1=0)、ID/EX流水级(pc\_2=0)，ID、EX阶段的指令设为无效(useful\_1=0、useful\_2=0)。在经过四个时钟周期后，第二条指令执行完成，又一个时钟周期后将其计算结果0x25写入寄存器reg8。    RAW情形A：  4: 02500413 addi x8,x0,37  8: 01841413 slli x8,x8,0x18  我采用前递解决该数据冒险。当第一条指令进入EX阶段、第二条指令在ID阶段时，检测到RAW情形A(rs1\_id\_ex\_hazard=1)，于是采用组合逻辑直接将EX阶段ALU计算结果from\_alu\_2前递给data1\_1，在下一个时钟周期直接作为ALU的第一个操作数，以保证第二条指令计算结果的正确。故又过三个时钟周期后将两条指令正确执行所得结果0x25000000写回寄存器reg8。    8: 01841413 slli x8,x8,0x18  c: fffff0b7 lui x1,0xfffff  10: 0080a023 sw x8,0(x1) # fffff000 <\_end+0xffffaf90>  RAW情形B：  我采用前递解决该数据冒险。当第一条指令进入MEM阶段、第三条指令在ID阶段时，检测到RAW情形B(rs2\_id\_mem\_hazard=1) ，于是采用组合逻辑直接将MEM阶段ALU计算结果from\_alu\_3前递给data2\_1，在下一个时钟周期直接作为ALU的第二个操作数，以保证第三条指令计算结果的正确。故又过两个时钟周期后将要写入内存中的正确内容data2\_3=0x25000000写入内存地址from\_alu\_3=0xfffff000。    298: 00208733 add x14,x1,x2  29c: 00000393 addi x7,x0,0  2a0: 06600193 addi x3,x0,102  2a4: fc7716e3 bne x14,x7,270 <fail>  RAW情形C：  我采用前递解决该数据冒险。当第一条指令进入WB阶段、第四条指令在ID阶段时，检测到RAW情形C(rs1\_id\_wb\_hazard=1) ，于是采用组合逻辑直接将WB阶段ALU计算结果from\_alu\_4前递给data1\_1，在下一个时钟周期直接作为BranchComp的第一个操作数，以保证第四条指令比较结果的正确。故一个时钟周期后，BranchComp中两操作数data1=data2，指令不跳转(pcSel=1)。 |

3 设计过程中遇到的问题及解决方法

|  |
| --- |
| 要求：包括设计过程中遇到的有价值的错误，或测试过程中遇到的有价值的问题。所谓有价值，指的是解决该错误或问题后，能够学到新的知识和技巧，或加深对已有知识的理解和运用。 |
| 1. 在比较器BranchComp中进行有符号数的比较时，直接用“>”、“≤”判断时，默认进行的是无符号数的判断，改正后应该考虑分为两操作数同号和异号两种情况。 2. 对于时序逻辑电路，输出信号相比时钟有效边沿存在一定的输出延时，在单周期CPU设计时，若寄存器堆的读取采用组合逻辑，可能读取到pc还未来得及更新时的错误值，上板验证后发现结果也确实不正确，且存在电路不稳定的现象，改正后采取在时钟的上升沿更新pc，在时钟的下降沿读取寄存器堆RF。而在流水线CPU设计时，各个阶段总在执行不同的指令，为保证指令获得正确的执行结果，寄存器堆的读取操作应改成组合逻辑。 3. 运算器中对操作数a进行移位操作时，最多移动31位，故仅取操作数b的低5位作为移位位数shamt。考虑到算数右移有符号数的情况，代码应写成“result=($signed(a))>>>b[4:0]”。 4. 对于获取外设显示数据，相当于S型指令，设计单周期cpu时，采用组合逻辑，而设计流水线cpu时，应采用时序逻辑。而读取拨码开关，相当于LW指令，单周期cpu和流水线cpu均采用组合逻辑。 5. 复位信号rst为特殊信号，在if-else等条件判断语句中应单独列为一种情形，与其它条件分开，比如我原先写的if(rst | !flush)…应写成if(rst)…else if(!flush)… 6. 针对数码管显示全0但仿真波形无误，我所遇到的原因和解决方法：单周期时时钟分频有误，应根据时钟IP核传入的clk时钟频率25MHZ，经分频后数码管刷新频率2ms刷新一次来进行分频。流水线时原采用组合逻辑获取数码管要显示的数字，导致时许有误，应采用和CPU相同时钟的时序逻辑，在每个clk\_i的上升沿写数码管和led。 7. 在流水线CPU设计中，含有多个时序部件，尽量都采用同一个时钟沿。我原先流水级采用下降沿，跟新pc、读寄存器堆、写内存等采用上升沿，仿真比较难看，但也算成功反应理想情况下结果正确，但板子可能就会反应不过来，这也可能是我数码管显示全零的原因之一。 8. 流水线CPU设计过程中，采用无脑停顿虽说比前递好想，但前者会降低cpu整体效率，而且实际上后者的代码并不比前者的复杂。 |

4 总结

|  |
| --- |
| 要求：谈谈学完本课程后的个人收获以及对本课程的建议和意见。请在认真总结和思考后填写总结。 |
| 因为我有一位比我大一级的计科学长给我做了一个好榜样，他去年取得了这门课程的满分，同时我对这门课程兴趣浓厚，所以在考完试后我就开始认真着手于这门课程。  伊始到完成单周期计算器的下板，我花了近一周的时间，主要是刚开始我花了几天时间把指导书上提及的MobaXterm、VS Code、虚拟机等下载到自己的电脑里，熟悉软件、使用远程平台等（我还是这类经验太欠缺，以前配置个环境、移动个软件目录都要花很长时间，捂脸.jpg）。单周期cpu主体的设计还是比较简单，但因为之前数逻知识的遗忘，下板的时候出现了很多问题，甚至因为降频失败导致数码管显示全零，这个问题还想了我很久。后面四五天的时间里设计流水线CPU，我最大的困难和犯的错误出现在设计流水级等时序部件的时许上，还常常疑惑我这trace和仿真都没毛病哇，为啥下板就不行？现在我明白，不能总在理想情况下思考问题，也得为板子想想，捂脸.jpg。  我自觉对于《数字逻辑设计》和《计算机组成原理》的掌握并不全面深刻，还有很多遗忘，脑子也不灵活，所以算上我们班上课的半天时间，我几乎是把电脑搬到机房第一排放一整天。我见过的T2210上课的老师们都很好，不论是不是自己班里的学生，都很热心地帮我解决搓cpu过程中各式奇奇怪怪的问题，还老开玩笑说我又来白嫖老师hh，谢谢老师和同学们~  流水线180mhz（比我学长高10mhz，嘿嘿）上板成功的时候，我在心里满满的充实感和成就感，通过这几天的学习，我也对《数字逻辑设计》和《计算机组成原理》的知识有了进一步的熟悉和掌握，收获满满。  我还有一点小小的建议是对指导书的修改，对于“单周期CPU时序”这一块，我采用“（3）将RF的读逻辑更改为组合逻辑”，结果失败。还有建议下板测试trace的.xci文件和IP核的修改、更新、使用说明能再详细点，我之前在到vscode测试trace文件、跑仿真、下板trace测试、下板计算器反复折腾，脑子都混乱了，捂脸.jpg。 |