

（深圳）

# 实验报告

开课学期： 2021夏季

课程名称： 计算机设计与实践

实验名称： CPU设计

实验性质： 综合设计型

实验学时： 52 地点： T612

学生班级： 7班

学生学号： 190110713

学生姓名： 陈思超

评阅教师：

报告成绩：

实验与创新实践教育中心制

2021年5月

注：本设计报告中各个部分如果页数不够，请大家自行扩页，原则是一定要把报告写详细，能说明设计的成果和特色。报告中应该叙述设计中的每个模块。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） |
| 单周期CPU：  实现的指令：add,sub,and,or,xor,sll,srl,sra,addi,andi,ori,xori,alli,aril,aria,lw,jalr,sw,beq,bne,blt,bge,lui,jal  CPU频率：  流水线CPU：24MHz  实现的指令：add,sub,and,or,xor,sll,srl,sra,addi,andi,ori,xori,alli,aril,aria,lw,jalr,sw,beq,bne,blt,bge,lui,jal  CPU频率：50MHz |
| 设计的主要特色（除基本要求以外的设计） |
| 1. 流水线CPU实现了数据前递的功能，用于解决普通的一级和二级数据冒险，以及辅助解决load\_use型数据冒险 2. 流水线CPU实现了静态分支预测的功能，每当有跳转指令出现的时候总是预测不跳转 3. 流水线CPU实现了清除指令的功能，主要用于在静态分支预测失败的时候将IF和ID阶段正在执行的两条指令清除 4. 流水线CPU实现了流水线暂停的功能，主要用于和前递配合解决load\_use型数据冒险 |
| 资源使用情况、功耗数据截图（实现后） |
| 单周期CPU:    流水线CPU： |

1 单周期CPU设计与实现

1.1 单周期CPU整体框图

|  |
| --- |
| （要求：无需画出模块内的逻辑，但要标出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） |
| （红色的线是控制单元cu输出的控制信号）  pc：  pc是时序逻辑部件，是一个32位的指令地址寄存器，存储当前正在执行的指令的地址。  npc：  npc是一个组合逻辑部件，主要用于计算下一条指令的地址，并且将计算的结果连到pc上去  imem：  imem是一个组合逻辑部件，在哈佛架构中是指令存储器，是一个用于存放32位指令的64KB的ROM，它支持数据的异步读写，也就是只要在输入端一给出地址，就可以立马在输出端得到相应的指令  cu：  cu是一个组合逻辑部件，是控制单元，cu接受并解析整条指令的格式，然后根据指令的类型来输出控制信号，控制各个部件的行为  reg\_file：  reg\_file是一个包含了32个32位寄存器的寄存器堆，它是一个读的时候是组合逻辑，写的时候是时序逻辑的部件  sext：  sext是一个组合逻辑部件，是立即数符号扩展部件，他接受整条指令，并且根据cu传过来的控制信号sext\_op来判断要如何进行符号扩展  ALU：  ALU是一个组合逻辑部件，是计算单元，ALU接受两个操作数，然后根据cu传给它的alu\_op控制信号来判断要对这两个操作数执行什么运算，可以执行的运算有add，sub，or，and，xor，sll，srl，sra，仅仅输出第二个操作数等等。  Branch\_cmp：  Branch\_cmp是一个组合逻辑部件，是分支判断单元，在执行分支跳转指令的时候cu传给branch\_cmp的branch控制信号才会有效，此时它才会正常工作。在执行分支跳转指令的时候，cu会控制ALU让他执行sub运算，Branch\_cmp接受两个操作数相减得到的结果，根据这个结果判断第一个操作数到底是大于，还是小于，还是等于第二个操作数，并且相应地把输出信号GT，LT，EQ给置位，发送给cu  dmem：  dmem是一个时序逻辑部件，在哈佛架构中是数据存储器，是一个存储32位数据的64KB的RAM。dmem读写数据的时候需要等待时钟沿的到来，而这里的时钟沿应该是下降沿 |
|  |

1.2 单周期CPU模块详细设计

|  |
| --- |
| （要求：各个模块的详细设计图，要包含内部的子模块，以及关键性逻辑，标出信号名和位宽，并有详细说明） |
| ALU：    ALU是运算单元，它的作用主要有三：执行加减法运算，执行逻辑运算，执行移位运算。ALU执行运算功能的部件主要有三个，一个是执行补码加法运算的部件，一个是执行逻辑运算的部件，还有一个是执行移位运算的部件。同时在执行加法部件之前还有一个补码取反的部件get\_opposite，这个部件会把第二个操作数B取反，当执行减法指令的时候这个部件才会起作用，这样就可以只使用一个加法器就能够执行加减两种运算了。三个部件都会执行运算，并且输出结果，而最后会有一个MUX，这个MUX会根据alu\_op输出想要的结果  Sext：  Sext是立即数扩展单元，sext会根据指令的格式提取指令中的立即数，并且把立即数进行符号扩展  Reg File：  Reg File是寄存器堆，里面有32个32位的寄存器，这些寄存器除了x0是只读并且值只能是0之外，其他的寄存器都是可读可写的。所有的寄存器都是读时逻辑，写时时序 |

1.3 单周期CPU仿真及结果分析

|  |
| --- |
| （要求：包含逻辑运算指令、访存指令、跳转指令的仿真截图，以及结果分析） |
| 逻辑运算指令：  add指令：  执行的指令如下：  addi x1,x0,23  addi x2,x0,31  add x3,x1,x2  前两条指令分别把x1寄存器和x2寄存器赋值为23和31，第3条add指令把x1和x2的值加起来，存入x3中：    可以看到：这里的x3最后的值确实是正确的，是我们所期待的23+31=54。  这条add指令首先会在if模块取值，把取出来的指令inst送到id模块：  可以看到，此时的02081b3就是add x3,x1,x2的机器码的16进制形式。  id模块得到了inst之后，会把inst送入到控制单元cu中：    Cu是一个组合逻辑部件，当它接受了inst的输入之后，会对inst的指令格式进行解析，把相应的控制信号进行赋值。在执行add x3,x1,x2的时候，cu发现这是一条R型指令add，于是把rf\_we置为有效，表示寄存器堆写使能有效； 把wd\_sel设置为0，表示写入寄存器堆的数据是来自alu的计算结果；把alub\_sel设置为0，表示此时的alu的第二个操作数是来自第二个源寄存器；把alu\_op置为0，表示让alu执行add运算。  这些信号分别会连接到相应的部件上，控制这些部件的行为，比如alu：    可以看到，执行add x3,x1,x2的时候alu\_op是0，第一个操作数A是x1寄存器的值23，第二个操作数B是x2寄存器的值31，那么最终得到的结果C就是54  由于add指令是R型指令，没有访存阶段，所以dram\_we是无效的，mem模块并不工作。而最后到了写回的时候：    Rf\_we写使能信号有效，同时wR是3，wD是alu的计算结果，那么当时钟的上升沿到来的时候alu的计算结果54就会写入到x3寄存器中去。  至此，一条add指令执行完毕。  访存指令：  lw指令：  执行的指令如下：  lui x1,0x2  addi x2,x0,12  sw x2,4(x1)  lw x3,4(x1)  首先前三条指令把12这个数值放入了数据存储器中的0x2004中，然后第4句lw指令把数据存储器中的0x2004中的数据读出来存入x3寄存器中，所以最后x3的值是12：    在执行lw指令的时候，首先需要在if模块中进行取值：    取出的指令inst是0040a183，这是指令lw x3,4(x1)的机器码的16进制形式。接着，这个信号会被送入id模块，id模块的cu会对这个信号进行解析：    cu会发现这是一条I型指令lw，那么就会把rf\_we置为有效，表示写回寄存器堆有效；把wd\_sel置为10，表示此时写回寄存器堆的数据是从数据存储器data\_mem中读取出来的；sext\_op是1，表示此时需要对I型指令进行立即数的有符号扩展；alu\_op为0，表示此时alu需要执行add操作；alub\_sel是1，表示此时的alu的第二个操作数是符号扩展之后的立即数。  这些信号会连接到相应的部件中，控制它们的后续行为，比如sext：    此时sext\_op是10，表示对I型指令进行立即数扩展，因此它会把输入inst按照I型指令的格式把立即数的部分提取出来，并进行符号扩展，最终4进行符号扩展之后还是4，并将ext=4输出。这个ext会被连接到alu：  由于alub\_sel为1，因此alu的第二个操作数是来自sext的输出ext=4，而第一个操作数是x1的值0x2000，又因为alu\_op是0，因此alu执行add操作，把两个值相加就会得到最终的结果C=0x2004。这个结果会连接到data\_mem数据存储器作为地址输入：    可以看到，此时data\_mem的输入地址adr是0x2004，由于数据存储器是异步读取，因此一旦输入地址有效的时候，数据就会出现在输出端口rd上，这个rd会连接到寄存器堆中：    此时的wR是3，wD是刚刚数据存储器读取出来的值12，而且写寄存器堆使能rf\_we也是有效的，因此一旦时钟的上升沿到来，就会把12写入x3寄存器中。  至此，一条lw指令执行完毕  跳转指令：  beq指令：  执行的指令如下：  addi x1,x0,23  add x2,x1,x0  addi x3,x0,24  beq x1,x2,A  addi x4,x0,4  A:  addi x5,x0,5  此时x1和x2是相同的，因此第4条beq指令会跳转，也就意味着执行结束之后的x4的值是0，x5的值是5：    执行beq指令的时候首先也是取值，然后译码：    此时cu发现这是一条B型指令beq，那么就会把need\_branch置为1，表示此时是跳转指令。同时alu\_op会被置为1，表示alu需要执行sub：    Alu执行sub操作之后会得到结果C是0，那么这个0会被传入branch\_cmp进行判断：  Branch\_cmp发现sub得到的结果是0的时候，就会把输出branch置为0，表示此时比较的两个寄存器的值是一样的。那么这个branch会输出到cu中，cu得到了这个branch之后，发现此时的need\_branch是1，同时又发现此时的branch表示两个寄存器的值相同，再结合inst的格式发现这是一个beq指令，于是就知道下一条指令需要跳转，于是把npc\_op置为1，表示下一条指令不再是顺序执行了，而是需要进行跳转的：    那么此时的npc输出的npc就不会是0x10，而是0x14。  至此，一条beq指令执行结束。 |
|  |

2 流水线CPU设计与实现

2.1 流水线的划分

|  |
| --- |
| （要求：画出流水线的划分，并标明每个阶段CPU完成的功能） |
| 流水线主要划分为五个阶段：IF（取指），ID（译码），EX（执行），MEM（访存），WB（写回）。    IF（取指）阶段：  取指阶段做的主要工作是在时钟上升沿的时候根据pc的值作为地址去指令存储器inst\_mem中去取出相应的指令，送入ID阶段  ID（译码）阶段：  译码阶段做的主要工作有三：一，控制单元cu解析IF阶段传递过来的指令，然后赋值各个控制信号，把传给后面的各个阶段；二，立即数扩展单元sext获取IF阶段传递过来的指令，把指令中的立即数扩展成32位ext，然后传递给EX阶段；三：解析IF阶段传递过来的指令，然后从里面解析出rR1和rR2，根据这两个值从寄存器堆Reg File取出两个源寄存器的值传给EX阶段  EX（执行）阶段：  执行阶段做的主要工作是根据ID阶段传递过来的rD1和rD2还有ext，然后根据传递过来的控制信号，控制ALU来对相应的操作数进行相应的运算  MEM（访存）阶段：  访存阶段做的主要工作是根据EX阶段传过来的数据和ID阶段穿过来的控制信号来判断是否需要进行访存，根据什么地址进行访存，是写入还是读出，读出到哪一个寄存器，写入的寄存器是哪一个等等  WB（写回）阶段：  写回阶段进行的主要工作是根据ID阶段传过来的控制信号判断是否需要写回，根据EX阶段传过来的寄存器号和需要写回的数据来判断应该写回到哪里，写回是写回到Reg File |

2.2 流水线CPU整体框图

|  |
| --- |
| （要求：无需画出模块内的逻辑，但要标出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） |
| IF：取指模块，从指令寄存器中取出指令inst，放入IF/ID流水线寄存器  ID：译码模块，根据IF/ID流水线寄存器中取出的指令进行解析，同时根据指令中的寄存器编号在寄存器堆中寻找相应的寄存器并取指，同时进行符号扩展，然后把所有这些信号送入ID/EX流水线寄存器  EX：执行模块，根据ID/EX流水线寄存器传过来的寄存器值和立即数，根据alu\_op进行相应的计算，把计算结果放入EX/MEM流水线寄存器  MEM：访存模块，根据EX/MEM流水线寄存器传过来的数据和控制信号进行访存，并且把数据值传入到MEM/WB流水线寄存器  IF/ID,ID/EX,EX/MEM,MEM/WB：流水线寄存器，用于存储各个流水线阶段运行的值  HU：控制冒险检测单元，控制冒险单元总是会进行静态分支预测，每一次都预测跳转，它会根据控制信号pc\_sel和npc\_op的取指来判断静态分支预测是否成功，如果失败了，就将输出信号flush赋值为1，将此时IF和ID阶段正在执行的指令给清空  FU：前递单元，通过ID，EX，MEM阶段传过来的数据以及控制信号，判断是否需要进行数据前递  SU：停顿单元，如果出现了load use型数据冒险，那么就必须要使用停顿，停顿单元接受FU输出的load use信号，如果该信号为1，那么就说明需要停顿 |

2.3 流水线CPU模块详细设计

|  |
| --- |
| （要求：各个模块的详细设计图，要包含内部的子模块，以及关键性逻辑，标出信号名和位宽，并有详细说明；数据冒险与控制冒险的解决方法必须要详细说明） |
| Forwarding\_unit：  forwarding\_unit是前递检测单元，它接受当前ID阶段执行的指令的rR1和rR2，同时也接受EX和MEM阶段执行的指令的wR和rf\_we。这个前递检测单元会进行判断，如果EX和MEM阶段的rf\_we有效的话，就把ID阶段执行的指令的rR1和rR2和EX和MEM阶段执行的指令的wR进行比较，如果相同的话，那么就说明出现了ID/EX冒险或者EX/MEM冒险，把rD1\_sel和rD2\_sel置成相应的值。如果ID/EX冒险和EX/MEM冒险同时出现了的话，那么就选择执行EX/MEM冒险。因此上述的代码使用伪代码来表述就是：  同时，这个forwarding\_unit也会进行load\_use型数据冒险的判断。Forwarding\_unit会接受ex阶段执行的执行的opcode，如果此时ex阶段正在执行的指令的rf\_we是有效，同时wR和rR1或者rR2相同，而且此时如果opcode显示该指令是lw，那么就说明此时发生了load use型数据冒险，那么此时forwarding \_unit就会把load use型数据冒险信号赋值并且输出到stop\_pipelined\_unit，同时把rD1\_sel或者rD2\_sel赋值成ID/EX冒险。 |
| Stop\_pipelined\_unit:  Stop\_pipelined\_unit用于暂停流水线，该部件主要是在load use型数据冒险发生的时候和前递一起解决冒险的。它接受forwarding\_unit输出的load use信号，如果这个信号有效，那么stop\_pipelined\_unit就会把输出信号stop置为有效，这个信号会传递给IF模块，让IF里面的pc的值不变，同时传递给ID/EX流水线寄存器，不让流水线寄存器的值改变，以此来达到暂停流水线一次的目的  hazard\_detection\_unit:  hazard\_detection\_unit是控制冒险检测单元，由于我做的流水线CPU实现了静态分支预测，每一次都是预测分支不跳转，因此我们需要hazard\_detection\_unit来检测预测是否成功，这个hazard\_detection\_unit会接受两个控制信号pc\_sel和npc\_op，如果这两个控制信号有任意一个不是0的话，那么就说明此时需要执行跳转，那么hazard\_detection\_unit就会输出flush到IF/ID和ID/EX流水线寄存器，把IF和ID阶段正在执行的两条指令给清空  流水线寄存器：  4个流水线寄存器IF/ID,ID/EX,EX/MEM,MEM/WB都是时序逻辑部件，而且都是下降沿触发，当下降沿来临的时候，流水线寄存器就会把输入端的值存储到内部的寄存器中，并且进行输出。同时部分流水线寄存器还会连接flush控制信号和stop控制信号，如果flush或者stop信号有效，那么流水线寄存器就会把控制信号的所有使能信号置为无效，同时把流水线寄存器中的指令置为add x0,x0,x0，这就就能够完成流水线的停顿和指令的清除 |

2.4 流水线CPU仿真及结果分析

|  |
| --- |
| （要求：包含数据冒险、控制冒险的仿真截图，以及结果分析） |
| 数据冒险：  执行的指令如下：  addi x1,x0,24  addi x2,x0,1  sub x3,x1,x2  可以看到，上述的代码中的第1行的目的寄存器是x1，而第3行的第一个源寄存器就是x1，因此会发生mem/wb数据冒险，而第2行的目的寄存器是x2，第3行的第二个源寄存器是x2，会发生ex/mem寄存器。而如果没有对数据冒险进行处理的话，由于复位的时候我们把所有的寄存器的值都设置为0，那么结果就会是x3的值是0。而我们处理了数据冒险之后，结果就是：    可以看到：x3的值是正确的，也就意味着我们对数据冒险的处理是成功的。  处理数据冒险的核心部件是位于id阶段的前递单元forwarding\_unit，它的仿真信号如下：    Forwardind\_unit接受id阶段正在执行的指令的rR1和rR2，id/ex流水线寄存器和ex/mem流水线寄存器输出的rf\_we（也就是寄存器堆的写使能信号）和wR（也就是要写回的目的寄存器编号）作为输入，输出rD1\_sel和rD2\_sel信号来判断此时id阶段正在执行的指令的rR1和rR2有没有发生数据冒险，如果发生了，那么发生的是什么数据冒险。这两个输出会接到id/ex流水线寄存器的rR1和rR2的输入的多路选择器mux中作为选择信号。此时执行前两条指令的时候，由于没有发生数据冒险，因此rR1\_sel和rR2\_sel都是0，表示没有发生数据冒险，而到了第3条指令的时候，由于发生了两个数据冒险，因此rR1\_sel的值是10，表示发生了mem/wb数据冒险，而rR2\_sel的值是01，表示发生了ex/mem数据冒险，那么此时这两个输出就会传到id/ex流水线寄存器的rR1和rR2的输入的多路选择器mux中作为选择信号：    可以看到，rD1\_mux和rD2\_mux的选择信号都被正确赋值了，所以id/ex流水线寄存器的输入id\_rD1和id\_rD2的值也都分别变成了id/ex流水线寄存器的wD（就是写入目的寄存器的值）和ex/mem流水线寄存器的wD了，至此，前递很好地解决了ex/mem和mem/wb数据冒险  Load use型数据冒险：  执行的指令如下：  addi x1,x0,12  sw x1,0(x0)  lw x2,0(x0)  add x3,x2,x0  此时第3句指令是lw指令，目的寄存器是x2，而下一句第4句指令就用到了x2作为源寄存器，此时就会发生load use型的数据冒险，如果不进行处理的话，由于数据存储器的初始值都是0，那么最终的x3的值会是0，但是我们此时进行了load use型数据冒险的处理之后x3的值就是：    处理load use型的数据冒险需要用到前递单元forwarding\_unit和流水线暂停单元stop\_pipelined\_unit的配合。我们先看forwarding\_unit的仿真波形：    可以看到，forwarding\_unit通过输入opcode\_from\_ex（就是id/ex流水线寄存器输出的opcode）得到此时ex中执行的是一条lw指令，同时根据输入wR\_from\_ex得到了这条lw指令的目的寄存器的编号是x2，同时输入rf\_we\_from\_ex又是有效的，而且发现此时正在id阶段执行的指令（也就是地4条指令）的第1个源寄存器rR1是x2，那么此时就会发生数据冒险，此时的forwarding\_unit的输出load\_use\_hazard就是1，这个信号会连接到stop\_pipelined\_unit，让流水线暂停一个周期：    （这里的stop信号会传入控制单元control\_unit，让流水线暂停一个周期）  同时此时的rR1\_sel就是11，表示发生了load use型数据冒险，需要前递，这个信号会连接到rD1\_mux作为选择信号： |
| 此时rD1\_mux的选择信号获取了11，进行了前递，把mem阶段输出的值赋值给了真正的rD1。  这样，前递和流水线暂停就共同完成了对load use型数据冒险的处理  控制冒险：  执行的指令如下：  addi x1,x0,31  addi x2,x0,12  bne x1,x2,J  addi x3,x0,11  J:  addi x4,x0,11  执行到第3条指令的时候遇到了分支指令，此时就会发生数据冒险，二如果你不进行处理的话，那么最终的结果就是x3和x4的值都是11。正确的结果应该如下：    执行到第3条指令的时候x1是31，x2是12，两者的值是不同的，因此此时的bne指令应该是要跳转的，但是由于我实现了静态分支预测，总是预测不进行跳转，而此时的预测失败了，需要进行指令的清空。而检测分支预测是否成功和负责输出清空信号的部件是hazard\_detection\_unit：    Hazard\_detection\_unit接受pc\_sel和npc\_op作为输入，当这两个输入都是0的时候表示此时的下一条指令的pc值是当前指令的pc值加上4，也就表示此时的程序并没有发生跳转，是顺序执行的，因此此时不用进行指令的清空；而一旦这两个输入有任意一个是1的话就说明此时的下一条指令的pc值不再是当前指令的pc值加上4，也就意味着程序进行了跳转，那么也就意味着此时的静态分支预测失效了，那么此时flush信号就会输出为1，这个flush信号最终会连接到id/ex和if/id流水线寄存器中：    一旦这两个流水线寄存器接收到了flush是1的话，那么一方面就会把流水线寄存器中所有的使能信号置为无效，另一方面会把当前的流水线寄存器中的指令设置为add x0,x0,x0，达到清空指令的效果。  这样就很好地解决了控制冒险的问题 |

3 设计过程中遇到的问题及解决方法

|  |
| --- |
| （包括设计过程中的错误及测试过程中遇到的问题）  我刚开始进行流水线CPU的代码编写工作的时候，所有的debug都是自己手写汇编代码，然后生产机器码之后导入vivado工程里面的指令存储器IP核进行调试。等我把所有的指令调试通过，进行trace也是没有问题的，所有选做的指令也全都通过了，接下来我又按照实验指导书的指导进行了上板验证，也是没有问题。但是之后当我想要再次跑trace的时候却发现trace的lw指令突然就过不了了。我刚开始以为是IP核的原因，可是IP核是虚拟机自带的，应该不会出问题。后来我找了半天，才发现原来是我上板的时候在数据存储器那里加了一句  wire [31:0] waddr\_tmp = waddr - 16'h4000;  而当时之所以需要加这句话是因为下板测试使用的汇编程序采用的是IROM和DRAM统一编址，因此需要对原本的DRAM访存地址进行修改。而trace的时候是不需要的。 |

4 总结

|  |
| --- |
| （要求：个人收获以及对课程的建议） |
| 个人收获：  在经历了四个月的实验之后，我的收获颇丰，最重要的一点是我对CPU的设计和工作流程有了更加深入的了解，这对提升我对整个计算机系统的理解大有脾益，同时这次的实验页让我对于verilog代码的编写更加熟悉了。  对课程的建议：  我觉得课程的实验设置地比较合理，实验有一定的难度，但是也在大部分人的能力范围之内。但是我觉得实验报告就比较多余了，因为实验都做完了，就没有必要再写什么实验报告了吧。 |