2023 春季学期-计算机组成原理-作业三

(要求：独立完成，5 月 5 日上课时上交)

1. (8 points)（课本 P163 页习题 5）假定课本中单周期数据通路对应的控制逻辑发生错误，使得控制信号RegWr、RegDst、ALUSrc、Branch、MemWr、ExtOp、R-type、MemtoReg中某一个在任何情况下总是为 0，则该控制信号为 0 时哪些指令不能正确执行？要求分别讨论。

   - 若RegWr=0，则所有需要写结果到寄存器的指令，例如R型指令、lw指令等，都不能正确执行，因为寄存器不发生写操作。
   - 若RegDst=0，则所有R型指令都不能正确执行，因为目的寄存器指定为Rt而不是Rd。
   - 若ALUSrc=0，则所有涉及立即数的指令，例如ori、addi、lw、sw、beq等，都不能正确执行，因为ALU的b端口数据来源一直是寄存器。
   - 若Branch=0，则branch类指令可能出错，因为永远不会发生转移。
   - 若MemWr=0，则store类指令不能正确执行，因为存储器不能写入数据。
   - 若ExtOp=0，则需要符号扩展的指令，例如 beq、lw/sw 等，都会发生错误，因为进行的是 0 扩展。
   - 若R-type=0，则ALU的局控失效，所有ALUctr来源于ALUop，造成R型指令执行错误。
   - 若MemtoReg=0，则写入寄存器的结果都来自于ALU的输出端，因此，lw指令不能正确执行。

2. (8 points)（课本 P163 页习题 6）假定课本中单周期数据通路对应的控制逻辑发生错误，使得控制信号RegWr、RegDst、ALUSrc、Branch、MemWr、ExtOp、R-type、MemtoReg中某一个在任何情况下总是为 1，则该控制信号为 1 时哪些指令不能正确执行？要求分别讨论。

   - 若RegWr=1，则所有不需要写结果到寄存器的指令，例如sw指令、beq指令等，都不能正确执行，因为可能错误地将某个值写入某个寄存器。
   - 若RegDst=1，则lw等部分I型指令不能正确执行，因为目的寄存器指定为Rd而不是Rt。
   - 若ALUSrc=1，则所有非立即数的运算指令，例如add、sub等，都不能正确执行，因为ALU的b端口数据来源一直是立即数。
   - 若Branch=1，则非branch类指令可能出错，因为会发生不必要的转移。
   - 若MemWr=1，则除store类指令外的其他指令都不能正确执行，因为会写入数据到存储器。
   - 若ExtOp=1，则需要零扩展的指令，例如ori等，会发生错误，因为进行的是符号扩展。
   - 若R-type=1，则ALU只有局控有效，所有ALUctr来源于func局控，造成R型指令之外的指令执行错误。
   - 若MemtoReg=1，则写入寄存器的结果都来自于数据存储器，因此，除lw指令之外的需要将结果写回寄存器的指令不能正确执行。

3. (10 points)（课本 P163 页习题 7）要在 MIPS 指令集中增加一条`swap`指令，可以有两种做法。一种做法是采用伪指令方式（即软件方式），这种情况下，当执行到`swap`指令时，用若干条已有指令构成的指令序列来代替实现；另一种做法是直接改动硬件来实现`swap`指令，这种情况下，当执行到`swap`指令时，则可在 CPU 上直接执行。要求：

   （1）写出用伪指令方式实现`swap rs, rt`时的指令序列（提示：伪指令对应的指令序列中不能使用其他额外寄存器，以免破坏这些寄存器的值）。

   （2）假定用硬件实现`swap`指令时会使每条指令的执行时间增加 10%，则`swap`指令要在程序中占多大的比例才值得用硬件方式实现？

   - 若在伪指令序列中，使用除 rs 和 rt 以外的寄存器，寄存器中的内容会遭到破坏，因此伪指令序列一般不能用额外的寄存器。swap 的伪指令序列如下：

     ```
     xor rs, rs, rt
     xor rt, rs, rt
     xor rs, rs, rt
     ```

   - 假设 swap 所占比例为 x，其他指令比例为 $1-x$，则程序执行时间变为原来的 $1.1\times(x+1-x)=1.1$ 倍。而用软件实现 swap 时，程序执行时间为原来的 $3x+1-x=2x+1$ 倍。因此，当 $1.1<2x+1$ 时卖硬件实现才有意义，此时，$x>0.05$，即 swap 所占比例大于 5% 时，才值得用硬件方式实现该指令。

4. (8 points)（课本 P164 页习题 8）假定课本中多周期数据通路对应的控制逻辑发生错误，使得控制信号`PCWr`、`MemtoReg`、`IRWr`、`RegWr`、`BrWr`、`MemWr`、`PCWrCond`、`R-type`中某一个在任何情况下总是为 0，则该控制信号为 0 时哪些指令不能正确执行？要求分别讨论。

   - 若 PCWr=0，则所有指令都不能正确执行，因为无法正确地更新 PC。
   - 若 MemtoReg=0，则所有 load 类型指令执行错误，因为写入寄存器的是 ALU 输出而不是读出的存储单元的内容。
   - 若 IRWr=0，则所有指令都不能正确执行，因为 IR 中不能写入当前指令，也就无法进行正确的指令译码，因此，后续的指令执行过程都不正确。
   - 若 RegWr=0，则所有需要写结果到寄存器的指令都不能正确执行，因为寄存器不会发生写操作。
   - 若 BrWr=0，则 branch 类指令会出现错误，因为转移地址不能正确写入。
   - 若 MemWr=0，则所有 store 类指令会执行错误，因为数据不能写入存储器中。
   - 若 PCWrCond=0，则 branch 类指令会出现错误，因为永远不会发生转移。
   - 若 R-Type=0，则所有 R 型指令都会出错，因为 ALUctr 的取值不是来自对 R 型指令进行解释的 ALU 局部控制器。

5. (8 points)（课本 P164 页习题 9）假定课本中多周期数据通路对应的控制逻辑发生错误，使得控制信号`PCWr`、`MemtoReg`、`IRWr`、`RegWr`、`BrWr`、`MemWr`、`PCWrCond`、`R-type`中某一个在任何情况下总是为 1，则该控制信号为 1 时哪些指令不能正确执行？要求分别讨论。

- 若 PCWr=1，则程序执行顺序失控，因为每个时钟都会更新 PC 值。
- 若 MemtoReg=1，则除 load 类型指令外的指令执行错误，因为选择了错误的信息写入寄存器。
- 若 IRWr=1，则所有指令都不能正确执行，因为每个时钟周期都会写入 IR，因此写入的不是当前指令。
- 若 RegWr=1，则所有不需要写结果到寄存器的指令都不能正确执行，因为寄存器错误地写入了数值。
- 若 BrWr=1，则分支目标地址寄存器会一直写入，但是对指令的执行没有影响。
- 若 MemWr=1，则除 store 类之外的指令会执行错误，因为存储器中写入了错误的数据。
- 若 PCWrCond=1，则除 branch 类之外的指令会出现错误，因为可能发生不必要的转移。
- 若 R-Type=1，则所有非 R 型指令都会出错，因为 ALUctr 的取值来自对 R 型指令进行解释的 ALU 局部控制器。

6. (10 points)（课本 P164 页习题 11）对于课本中的 MIPS 多周期处理器，假定将访问数据的过程分成两个时钟周期，则可使时钟频率从480MHz提高到560MHz，但这样会使得lw和sw指令增加时钟周期数。已知基准程序CPUint 2000中各类指令的频率为：load-25%、Store-10%、Branch-11%、Jump-2%、ALU-52%。那么，以基准程序CPUint 2000为标准，处理器时钟频率提高后的性能提高了多少？若将取指令过程再分成两个时钟周期，则可进一步使时钟频率提高到640MHz，此时，时钟频率的提高是否也能带来处理器性能的提高？为什么？

   - 假设 M1、M2、M3 分别表示时钟频率为 480MHz、560MHz 和 640MHz 的多周期处理器。根据多周期指令执行过程可知，load、store、branch、jump 和 ALU 类指令在 M1 中的 CPI 为 5、4、3、3、4，在 M2 中为 6、5、3、3、4，在 M3 中为 7、6、4、4、5。因此，三个处理器的平均 CPI 为：

     $\mathrm{CPI(M1)} = 25\% \times 5 + 10\% \times 4 + 11\% \times 3 + 2\% \times 3 + 52\% \times 4 = 4.12$
     $\mathrm{CPI(M2)} = 25\% \times 6 + 10\% \times 5 + 11\% \times 3 + 2\% \times 3 + 52\% \times 4 = 4.47$
     $\mathrm{CPI(M3)} = 25\% \times 7 + 10\% \times 6 + 11\% \times 4 + 2\% \times 4 + 52\% \times 5 = 5.47$
   - 则每秒可执行的指令数为：

     $\mathrm{IPS(M1)} = 480 \times 10^6 / 4.12 = 116.5 \times 10^6$
     $\mathrm{IPS(M2)} = 560 \times 10^6 / 4.47 = 125.3 \times 10^6$
     $\mathrm{IPS(M3)} = 640 \times 10^6 / 5.47 = 117.0 \times 10^6$
   - 所以 M2 相较于 M1 性能提升了 1.08 倍。
   - 数据访问只涉及 load/store 指令，而取指则涉及所有指令，使得 CPI 显著增大，从而降低了性能。

7. (30 points)（课本 P164 页习题 14）假定选择以下 9 条RV32I指令作为实现目标设计单周期RISC-V处理器：3 条R型指令（“add rd, rs1, rs2”、“slt rd, rs1, rs2”、“sltu rd, rs1, rs2”）；2 条I型指令（“ori rd, rs1, imm12”、“lw rd, rs1, imm12”）；1 条U型指令（“lui rd, imm20”）；

1 条S型指令（“sw rs1, rs2, imm12”）；1 条B型指令（“beq rs1, rs2, imm12”）；1 条J型指令（“jal rd, imm20”），参照课本中 MIPS 单周期处理器设计方案，完成以下任务并回答问题：

（1）给出RISC-V处理器中扩展器的设计方案。

（2）给出RISC-V处理器中ALU的设计方案。

（3）给出RISC-V单周期数据通路和对应的控制逻辑。

（4）为何RISC-V处理器中无需专门的局部 ALU 控制器？

（1）RISC-V 规定对立即数只进行符号扩展。因此，扩展器中只需要符号扩展的相关逻辑即可。

（2）本题相关指令涉及带溢出判断的加法、带符号和无符号整数的大小判断、相等判断、逻辑或操作等。ALU 需要具备这些功能。因此，ALU 可以采用与课本图 5.7 相同的设计方案。

（3）在设计 CPU 数据通路和控制电路时，首先需要分析每条指令的功能，然后根据功能设计所需的元件，并考虑如何将它们互连。之后，确定每个元件控制信号的取值，生成指令与控制信号之间的关系表。最后根据这个关系表，生成控制信号的表达式，并据此设计控制电路。

①指令功能分析

指令功能分析如表 1所示。

②数据通路设计方案

具体过程参考课本 5.2.2 节的设计过程，数据通路的设计方案可以参考图 5.16 的设计，但扩展器和取指部件设计方案不同。此处，扩展器只需要考虑符号扩展。对于取指部件，在计算无条件跳转指令的地址时，也需要将立即数作为相对偏移量与 PC 做加法运算，而且在计算条件跳转和无条件跳转偏移地址时，立即数只需要左移 1 位。此外，指令执行结果需要写回寄存器堆的时候，其目的地址只有 rd 一种选择，因此用于 rd 和 rt 选择的多路选择器也需要删除。其他的操作与图 5.16 所示数据通路相同。

②控制电路设计方案

具体过程参考课本 5.2.3 节的设计过程，控制逻辑的设计方案可以参考图 5.23 的设计，此处，扩展器由于只需要进行有符号扩展，因此无需额外的控制信号。寄存器写回地址只有 rd 一个选择，因此对应的多路选择器及其控制信号可以删除。

（4）此 CPU 只涉及 3 条 R 型指令，因此局部控制仅需 2 位信号，与全局控制信号的位宽不一致。因此，可考虑对 ALU 仅使用全局控制。

表 1: 9 条目标指令功能的 RTL 描述

指令	功能	说明
公共操作	M[PC], PC←PC +4	从 PC 所指的内存单元中取指令，并将 PC 加 4
add rd, rs1, rs2	R[rd]←R[rs1] + R[rs2]	从 rs 中取数后相加，若溢出则异常处理，否则结果送 rd
slt rd, rs1, rs2	if (R[rs1] < R[rs2]) R[rd]←1 else R[rd] ←0	从 rs1 和 rs2 中取数后按带符号整数来判断两数大小，小于则 rd 中置 1，否则 rd 中清 0（不进行溢出判断）
sltu rd, rs1, rs2	if (R[rs1] < R[rs2]) R[rd]←1 else R[rd] ←0	从 rs1 和 rs2 中取数后按无符号整数来判断两数大小，小于则 rd 中置 1，否则 rd 中清 0（不进行溢出判断）
ori rd, rs1, imm12	R[rd]←R[rs1] | SEXT(imm12)	从 rs1 取数，将 imm12 进行符号扩展，然后两者按位或，结果送 rd
lw rd, rs1, imm12	Addr←R[rs1] + SEXT(imm12) R[rd]←M[Addr]	从 rs1 取数，将 imm12 进行符号扩展，然后两者相加，结果作为访存地址 Addr 取数并送 rd
lui rd, imm20	R[rd]←{imm20, 12'b0}	将 imm20 载入 rd 的高 20 位，同时低 12 位置零
sw rs1, rs2, imm12	Addr←R[rs1] + SEXT(imm12) M[Addr]←R[rs2]	从 rs1 取数，将 imm12 进行符号扩展，然后两者相加，结果作为访存地址 Addr 写入 rs2
beq rs1, rs2, imm12	Cond←R[rs1] - R[rs2] if (Cond == 0) PC←PC + {SEXT(imm12), 1'b0}	做减法比较 rs1 和 rs2 的大小，计算下条指令地址，根据比较结果修改 PC
jal rd, imm20	R[rd]←PC + 4 PC←PC + {SEXT(imm20), 1'b0}	立即数符号扩展并与 PC 相加作为跳转地址，下指令地址保存在 rd 寄存器