姓名：李嘉昊

学号：515015910016

5段流水CPU设计实验报告

# 实验目的

1. 理解计算机指令流水线的协调工作原理，初步掌握流水线的设计和实现原理。

2. 深刻理解流水线寄存器在流水线实现中所起的重要作用。

3. 理解和掌握流水段的划分、设计原理及其实现方法原理。

4. 掌握运算器、寄存器堆、存储器、控制器在流水工作方式下，有别于实验一的设计和实现方法。

5. 掌握流水方式下，通过 I/O 端口与外部设备进行信息交互的方法

# 实验内容

1. 采用 Verilog 在 quartusⅡ中实现基本的具有 20 条 MIPS 指令的 5 段流水 CPU设计。

2. 利用实验提供的标准测试程序代码，完成仿真测试。

3. 采用 I/O 统一编址方式，即将输入输出的 I/O 地址空间，作为数据存取空间的一部分，实现 CPU 与外部设备的输入输出端口设计。实验中可采用高端地址

4. 利用设计的 I/O 端口，通过 lw 指令，输入 DE2 实验板上的按键等输入设备信息。即将外部设备状态，读到 CPU 内部寄存器。

5. 利用设计的 I/O 端口，通过 sw 指令，输出对 DE2 实验板上的 LED 灯等输出设备的控制信号（或数据信息）。即将对外部设备的控制数据，从 CPU 内部的寄存器，写入到外部设备的相应控制寄存器（或可直接连接至外部设备的控制输入信号）

6. 利用自己编写的程序代码，在自己设计的 CPU 上，实现对板载输入开关或按键的状态输入，并将判别或处理结果，利用板载 LED 灯或 7 段 LED 数码管显示出来

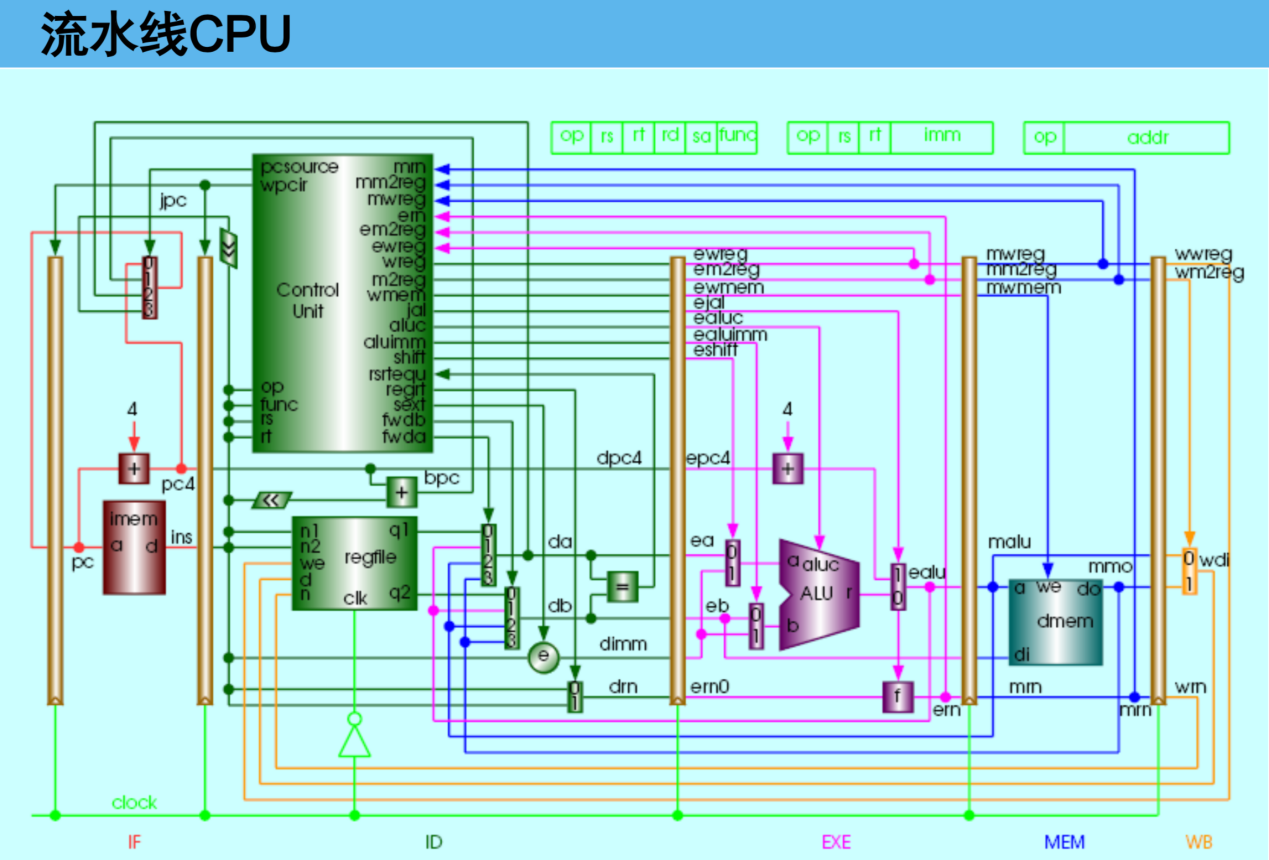
7. 例如，将一路 4bit 二进制输入与另一路 4bit 二进制输入相加，利用两组分别2 个 LED 数码管以 10 进制形式显示“被加数”和“加数”，另外一组 LED数码管以 10 进制形式显示“和”等。（具体任务形式不做严格规定，同学可自由创意）。

8. 在实现 MIPS 基本 20 条指令的基础上，实现 Y86 相应的基本指令。

9. 在实验报告中，汇报自己的设计思想和方法；并以汇编语言的形式，提供以上两种指令集（MIPS 和 Y86）应用功能的程序设计代码，并提供程序主要流程图。

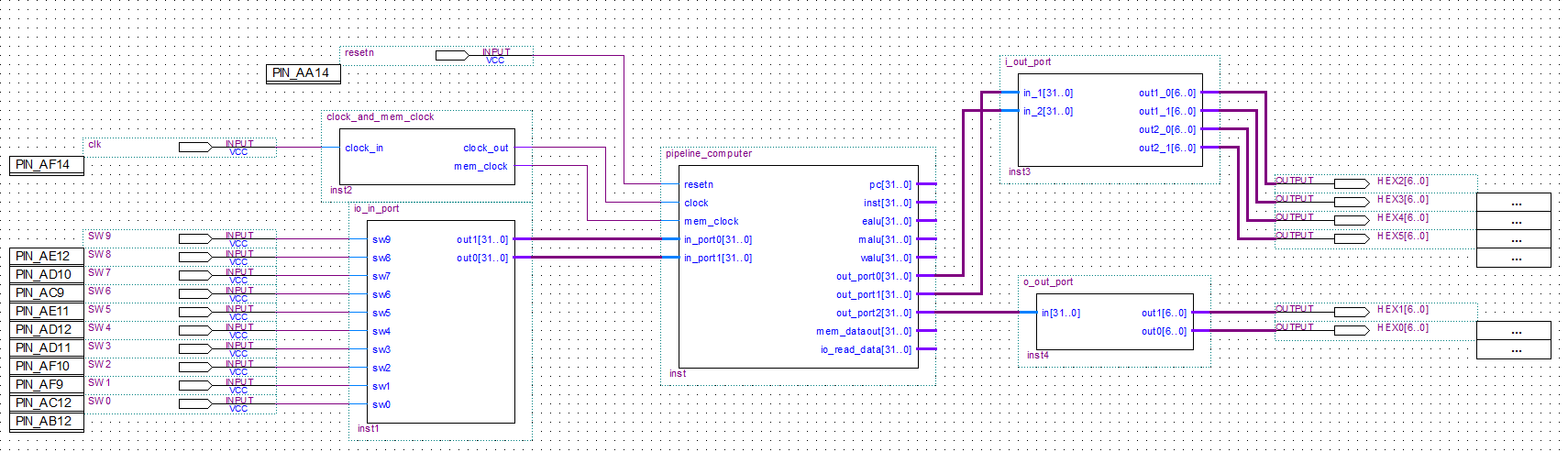
# 设计思路

* 1. 流水线寄存器设计



* + 1. pipe\_reg\_if：
* 该寄存器主要功能在于给IF stage提供pc4&inst的输入值
* 模块输入：
  + New\_pc:在IF stage中计算出的新的PC的值
  + Wpcir:在ID stage中计算出的用来判断cu是否需要stall的信号
  + Clk:时钟信号
* 模块输出：
  + 输出条件：时钟信号clk处于上升沿且wpcir不为零
  + Pc：传入IF stage的pc值
    1. pipe\_reg\_id：
* 该寄存器主要功能在于将IF stage计算得出的结果信号保存，并作为ID阶段的部分输入信号传入ID stage
* 模块输入：
  + pc\_plus4：当前指令的下一条指令的PC，用于后续PC的计算
  + Inst：用于在ID stage中进行位拆解来进行ID stage的操作和计算
  + Wpcir：在ID stage中计算出的用来判断cu是否需要stall的信号
  + Clk：时钟信号
* 模块输出：
  + out\_pc\_plus4：即是输入的pc\_plus4
  + out\_inst：即是输入的inst
    1. pipe\_reg\_exe：
* 该寄存器需要保存许多在ID stage中通过sc\_cu计算得出的&一些其他的信号。在时钟信号为上升沿的时候对输出信号进行赋值并传给EXE stage
* 模块输入：
  + dwreg：是否需要写入reg
  + dm2reg：是否需要从mem写入reg
  + dwmem：是否需要写mem
  + daluimm：alu的b操作数是否使用imm（立即数）
  + dshift：alu的a操作数是否使用移位位数
  + djal：是否执行子程序调用
  + da：寄存器a中的data，用来判断beq和bne指令是否需要进行跳转
  + db：寄存器b中的data，用来判断beq和bne指令是否需要进行跳转
  + dimm：立即数中的数据
  + dpc4：当前pc+4
  + daluc：alu中需要执行的指令的编码
  + drn0：需要写入的寄存器的编码
* 模块输出：
  + ewreg,em2reg,ewmem,ealuimm,eshift,ejal,ea,eb,eimm,epc4,ealuc,ern0
    1. pipe\_reg\_mem:
* 主要将EXE stage要传给MEM stage的信号保存，并在时钟信号为上升沿的时候对输出信号进行赋值并传给MEM stage
* 模块输入：
  + ewreg：是否需要写入reg
  + em2reg：是否需要从mem写入reg
  + ewmem：是否需要写mem
  + ealu：EXE stage中ALU的计算结果
  + eb：寄存器b中的data，用来判断beq和bne指令是否需要进行跳转
  + ern：需要写入的寄存器的编码
* 模块输出：malu,mb, mrn,mwreg,mm2reg,mwmem
  + 1. pipe\_reg\_wb:
* 主要将MEM stage要传给WB stage的信号保存，并在时钟信号为上升沿的时候对输出信号进行赋值并传给WB stage
* 模块输入：
  + mrn：需要写入的寄存器的编码
  + mmo：mem中读出的内容
  + malu：EXE stage中ALU的计算结果
  + mwreg：是否需要写入reg
  + mm2reg：是否需要从mem写入reg
* 模块输出：
  + 1. wmo,walu,wrn,wwreg,wm2reg
  1. 五段流水线每一阶段的概述
     1. pipe\_stage\_if（IF）：
* 根据IF reg传来的pc值取出相应的inst
* 根据其他stage传来的信号计算并选择下一个pc的值并传给pc reg
  + Pc选择受到pcsource（来自ID stage）的控制：
    - Pcsource=0 pc=pc\_plus4（当前指令的下一条指令）
    - Pcsource=1 pc=bpc（条件转移目标地址，针对beq&bne）
    - Pcsource=2 pc=jrpc（选择寄存器中的数据作为pc，针对jr）
    - Pcsource=3 pc=jpc（选择跳转指令的目标地址，针对j&ja）
    1. pipe\_stage\_id（ID）：
* 按位分割传入的指令（inst）
* 根据分割后的指令，用cu计算出各个控制信号,其中
  + Pcsource用来判断下一条指令的PC
  + wpcir用来判断cpu是否需要stall
* 根据分割后的指令，用regfile模块获取寄存器
* 根据分割后的指令，用mux2x5模块选择指令选择写入的寄存器编码
* 根据在cu中计算出的fwda&fwdb信号，利用mux4x32模块判断是否需要执行forwarding操作，若需要，根据fwda&fwdb的值选择forwarding的值写入da&db（作为EXE stage中ALU的操作数a&b）
* 根据da&db是否相等来计算出rsrtequ，用来判断bne&beq指令是否需要跳转，用来在ID stage中用cu模块计算出pcsource
* e为立即数扩展器，用来处理需要用到imm的指令，当符号扩展sext为1的时候做符号扩展（addi），否则做零扩展（andi&ori&xori）
* dimm代表立即数的值
* bpc代表beq&bne的条件跳转地址
  + 1. pipe\_stage\_exe（EXE）：
* 主要功能是根据传入的ea&eb以及指定指令的aluc在ALU模块中计算出结果，并通过ealu传递给pipe\_reg\_mem，通过reg传递给MEM stage
* 需要注意的是要判断一下inst种类是否为jal，若为jal指令，则需要把pc+8存入ealu，因为需要把pc+8作为return address，否则则把ALU的计算结果ealu0存入ealu
* Ealuc信号控制ALU模块执行哪种指令
* ewreg&em2reg&ewmem信号在MEM stage发挥作用，因此直接通过EXE stage转发给pipe\_reg\_mem
  + 1. pipe\_stage\_mem（MEM）：
* 主要功能是通过传入的数据（malu）对mem进行读/写。Mwmem信号指定是否需要写入mem（为1时需要写入，为0时不写入）
* 通过io\_input模块读入输入数据
* 通过io\_output模块输出数据到LED
  + 1. pipe\_stage\_wb（WB）：
* 由wm2reg信号决定是否需要从mem写入reg，并选择数据源是wmo /walu
* 由wrn决定写入哪个寄存器
* 由wwreg信号决定是否需要写入reg

1. IO实现
2. SW[0-4]输入数字一，SW[5-9]输入数字二，输入数字均采用二进制输入。输入模块为io\_in\_port，SW[0-9]代表10个switch开关，在io\_in\_port模块中SW[0-4]按照index赋值给out0作为第一个输入数，SW[5-9]按照index-5赋值给out1作为第二个输入数
3. HEX5&HEX4显示输入数字一，HEX2&HEX3显示输入数字二，由模块i\_out\_port实现输出显示
4. HEX1&HEX0显示输出数字（计算结果：输入数字一和输入数字二的加和），由模块o\_out\_port实现输出显示
5. 按下KEY0将CPU的resetn信号置为0，此时调整SW不会进行计算
6. 利用MIPS汇编指令编写了一个循环，不断从pipeline\_computer模块的in\_port0&in\_port1中读入数字，并通过mips加法指令对其进行相加。将加法结果和两个加数通过out\_port0&out\_port1&out\_port2输出到LED屏上
7. lpm\_rom\_irom和lpm\_ram\_dq\_dram模块设定了mips指令的相对路径
8. 实验电路



六、过程中遇到的问题

* 1. 由于重新建立了项目并更改了项目结构，导致MIPS指令的相对路径与sc\_computer不一样，在更改了lpm\_rom\_irom和lpm\_ram\_dq\_dram模块中的文件路径后解决了问题
  2. ID stage中的信号很多，一开始由于无法弄清楚每个信号的具体作用导致无法进行pipe\_stage\_id模块的编写以及哪些信号该传入到pipe\_reg\_exe，在参考流水线CPU的示意图以及弄清楚每个信号的作用后解决了问题。

七、实验总结

在本次实验中我通过实践，更好的理解了课上老师介绍的流水线CPU，也是第一次在没有大致框架的情况下自己实现一个verilog项目。通过实验我更好的理解了流水线CPU的工作原理和流水线CPU的整体架构和组成，清楚了每个stage的具体作用以及每个信号的作用。同时也发现了和单周期CPU相比，流水线CPU由于多出了很多功能（stall，forwarding）因此也多出了很多信号，并不只是将单周期CPU做简单的功能分割，而是各个stage间都有信号的传递来实现单周期CPU中没有的功能。