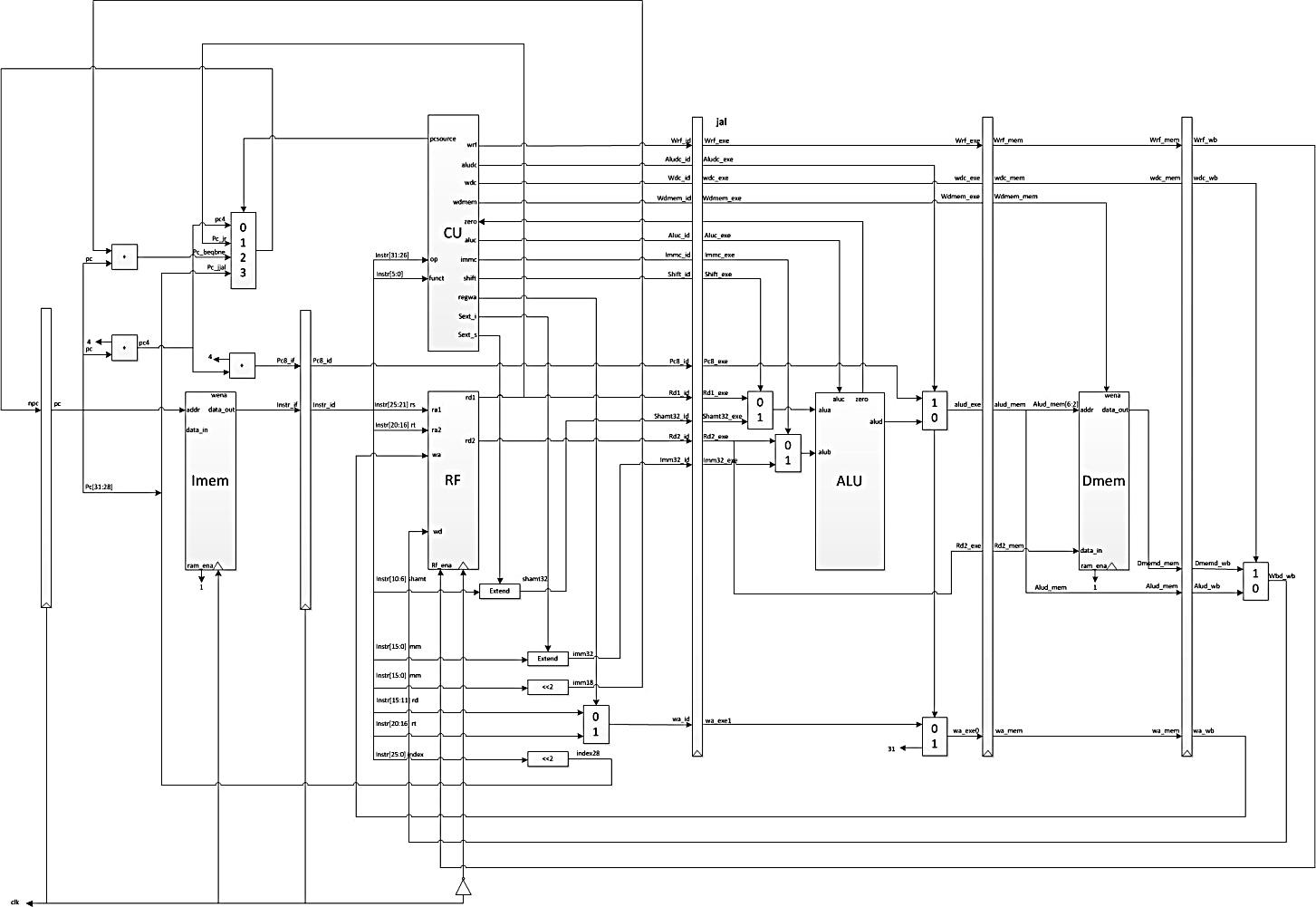
准Pipeline CPU模块

# 设计流程

在原本单周期CPU的基础上，在每一级之间增减流水线寄存器，即可转换成准pipeline CPU，数据通路如下图所示。



# 模块测试

测试方法：

1. 给出不存在流水线相关的指令，将其相应的二进制或十六进制机器码送入到imem的memfile.tv文件中。
2. 在保证单周期CPU正确的情况下，只观察几个必要重要的控制信号和数据就可以快速验证准pipeline CPU模块的正确性，如下所示的汇编指令，在送入准流水线CPU之后，只需要验证pc， npc， instr\_if， instr\_id， wa\_id， wa\_exe1， wa\_exe0， wa\_mem， wa\_wb， wrf\_id， wrf\_exe， wrf\_mem， wrf\_wb指令，就可以观测到如下图的流水线执行过程。

待测汇编指令

beq $1, $0, target

addi $3, $0, 10

target:

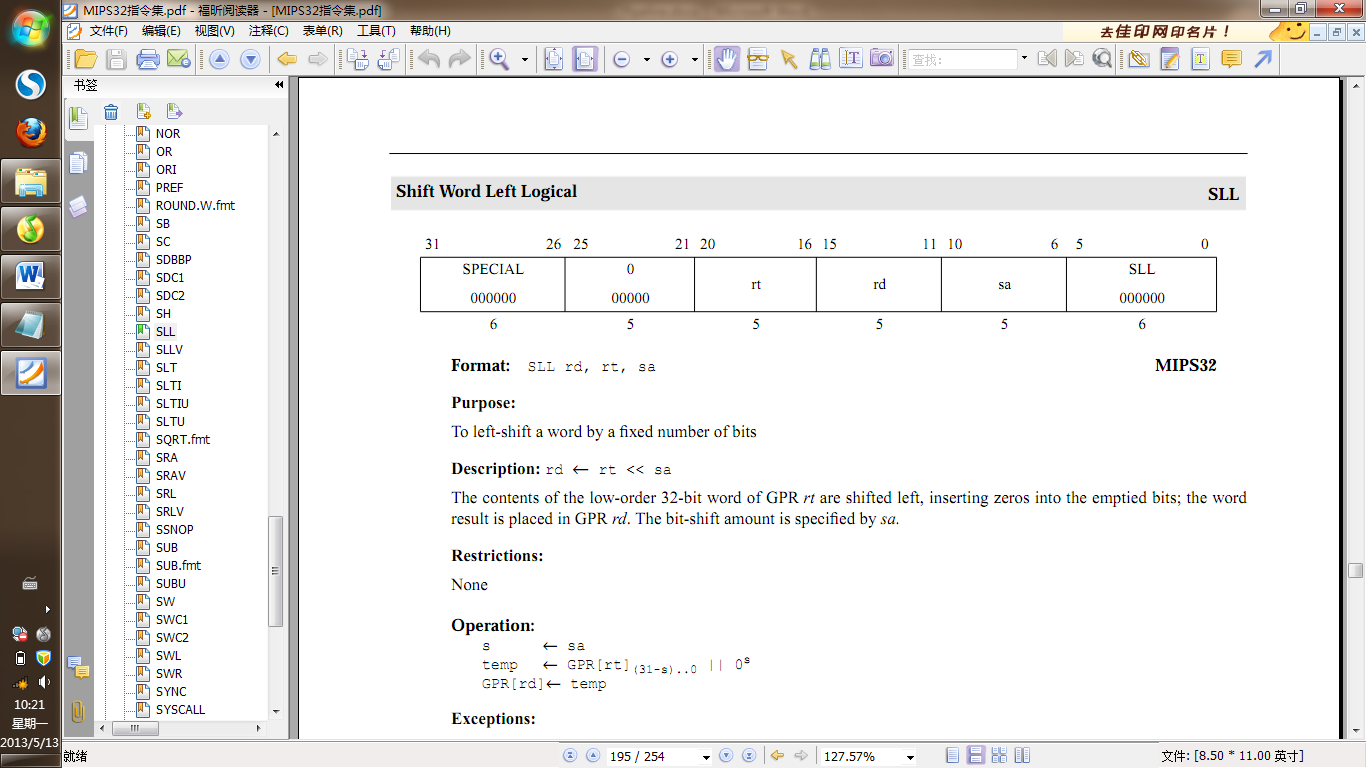
addi $2, $0, 5



注意事项：

1. Rst的复位高电平持续时间必须在下一个上升沿来临前结束，否则由于rst信号的问题，会导致下一条指令无法正确读入到流水线中
2. 由于rst信号的作用，对于dffe触发器，在rst=1时，给的初值为0，所以对于IR寄存器而言，其初始的instr值为0x0000\_0000。由于触发器复位信号的作用，使得第一条流入的指令会是一条0x0000\_0000指令，这条指令对于MIPS指令集而言不是一条不相关指令，而是满足sll指令格式，如下图所示。所以，第一条指令流入后，会在ID阶段，会使相应的wrf\_id控制信号为1，如果正确的第一条指令是不需要写寄存器的指令（beq，bne等），那经过流水线寄存器的传递，就会造成在WB级的一个错误写操作。对于这个错误的写，采取的方式是，这条错误的指令会导致0#寄存器写入一个0值，为了避免错误的写，我们在对寄存器写的时候进行控制，不能对0#寄存器进行写，代码如下所示。这也正好满足了MIPS指令规范，即0#寄存器是一个系统级寄存器，它不能由用户进行修改，且保持一个常量0。

经测试，采用上述方法可以避免造成上述的错误写的问题。



Sll指令格式

寄存器写控制代码

always @(negedge clk or posedge rst) begin

if(rst) begin

for(i = 0; i < 32; i = i + 1) begin

regfiles[i] <= 32'b0;

end

//for r\_type test

regfiles[1] <= 32'b1;

end else if(we && waddr != 32'b0) begin

regfiles[waddr] <= wdata;

end

end