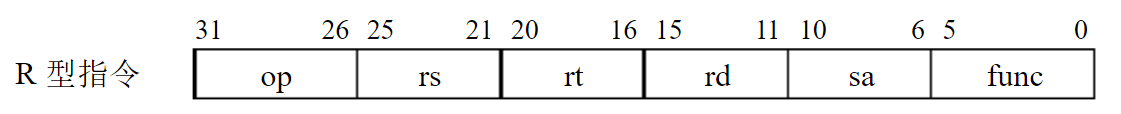
1. **实验名称**

单周期CPU的设计与实现

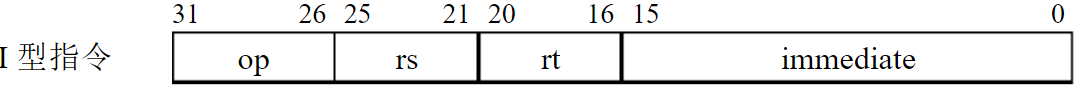
1. **实验目的**
2. 熟悉并掌握Quartus软件的使用，会使用调试工具SignalTap来辅助分析实验预期内容；
3. 熟悉并掌握单周期CPU的指令存储器、数据存储器及数据通路的构成、原理、设计及其代码实现；
4. 结合课本与课程内容认识和掌握指令与CPU的关系，感受由指令功能关系设计出具体的部件，以及部件间的优化等内容；
5. 掌握单周期CPU中各指令的具体的执行流程，以此来感受CPU设计的合理性；
6. 将一段程序存储在指令存储器中，实验要用设计出的CPU运行这段程序验证其可行性；
7. 掌握单周期CPU的测试方法；
8. **实验过程**
9. **熟悉MIPS的3种指令格式及相应指令的内容**

**1.R-型指令**

其中，op为操作码，操作类型由func字段指定：若为双目运算类指令，则rs和rt的内容分别作为第一和第二源操作数，结果送到rd；若为移位指令，则对rt的内容进行以为结果送到rd，移位数由sa给出；

5条R-型指令：add, sub, subu, slt, sltu;

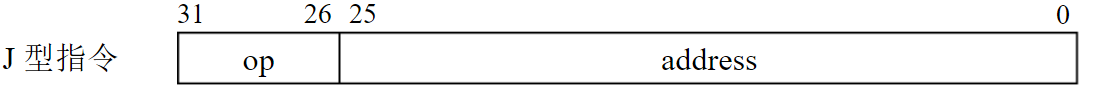
指令格式为： op rd, rs, rt

**2.I-型指令**

立即数指令：若为双目运算符类指令，则将rs的内容和立即数分别作为第一和第二源操作数，结果送到rt；若为Load/Store指令，则将rs的内容和立即数符号扩展后的内容相加作为内存单元地址，Load将内存单元中的内容送rt，后者将rt内容送内存单元；若是条件转移（分支）指令，则对rs和rt内容进行指定的运算，根据运算的结果，决定是否转到转移目标地址出执行，转移目标地址通过相对寻址方式得到，即将PC的内容和立即数符号扩展后的内容相加得到；

5条I-型指令： ori, addiu, lw, sw, beq;

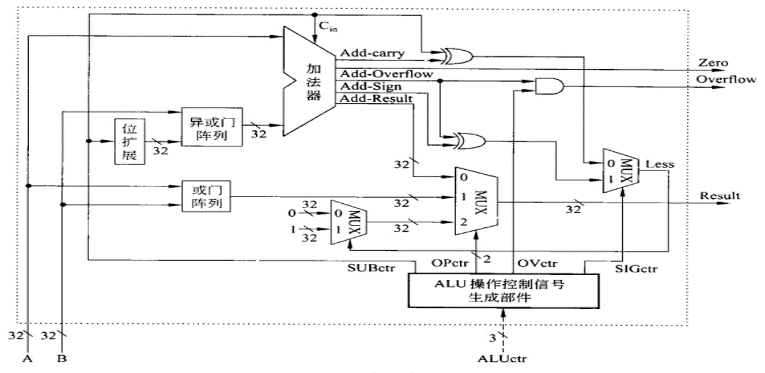
指令格式： op rt, rs, imm16

**3.J-型指令**

无条件跳转指令，指令中给出的是26位直接地址address，只要将当前的PC的高4位拼上26位直接地址，最后填两个0就可以得到32位的跳转目标地址；

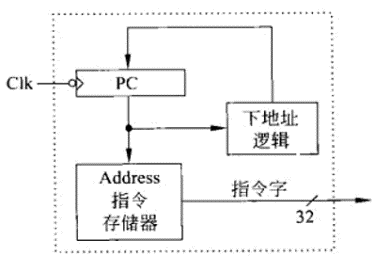
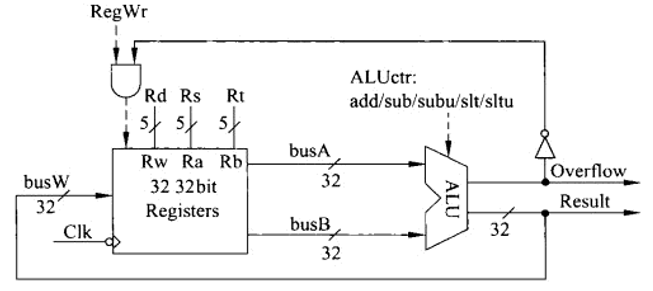
1条J-型指令：j target

1. **各种指令的数据通路的设计**
2. **算术逻辑部件ALU的设计**

ALU的主要功能为解决带溢出判断的加减法、带符号整数的大小判断、无符号数的大小判断、相等判断以及各种逻辑运算等运算的实现。

输入为两个32位的操作数AB，输出有运算的结果以及一些标志信息，同时有一个可以控制不同运算的操作控制端ALUctr；

1. **取指令部件的设计**

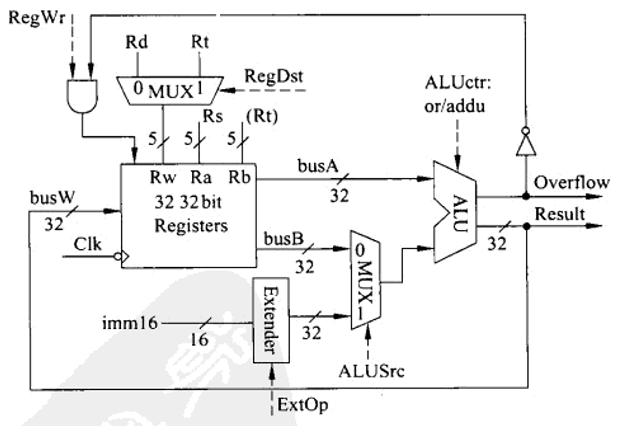
指令存储与指令存储器中，仅有读操作，每一个时钟周期PC的值更新，顺序执行时PC+4，转移执行时要区分时分支指令还是跳转指令；

1. **R-型指令的设计**

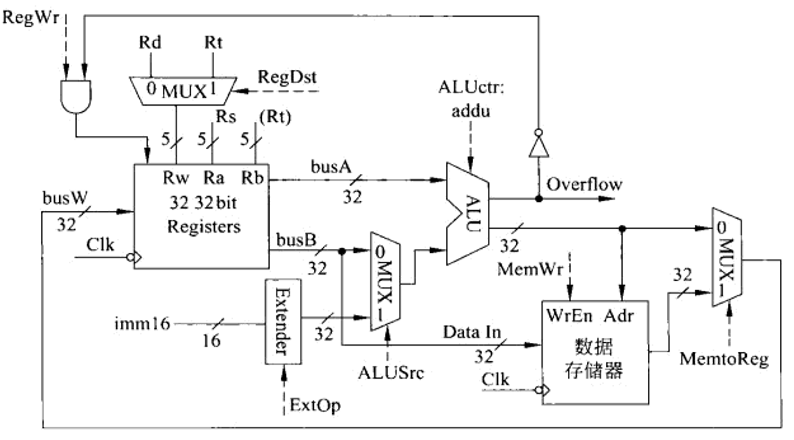
R-型指令将两个寄存器Rs和Rt内容运算后写入Rd寄存器，同时要有判断溢出的的功能；运行流程为将Rs和Rt的内容送到ALU运算，然后再合法的情况下通过busW写到Rd指向的寄存器；

1. **立即数运算指令的数据通路的设计**

I-型指令涉及对立即数imm16进行符号扩展或零扩展，然后和Rs的内容运算，将结果送到Rt中；

所以与R-型指令相比，可以通过改动一些线路使得上一个数据通路支持I-型指令；

**新增部件及作用：**

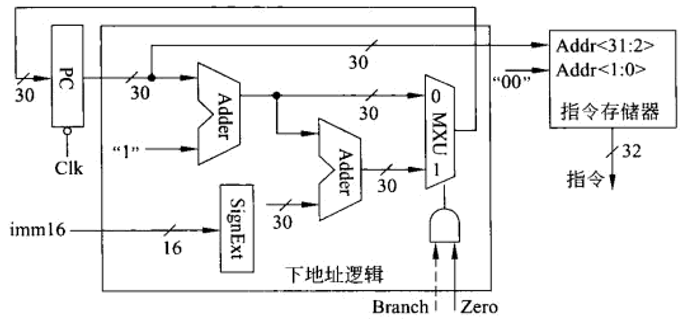
1. Rw处增加多路选择器，使用RegDst来控制不同指令的不同目的寄存器；
2. ALU B口的数据来源增加一个多路选择器，使用ALUSrc来控制不同指令进入ALUB口的数据不同的来源；
3. 增加一个扩展器，用于对立即数的扩展；逻辑运算时采用零扩展，算术运算时采用符号扩展；
4. **Load/Store指令的数据通路的设计**

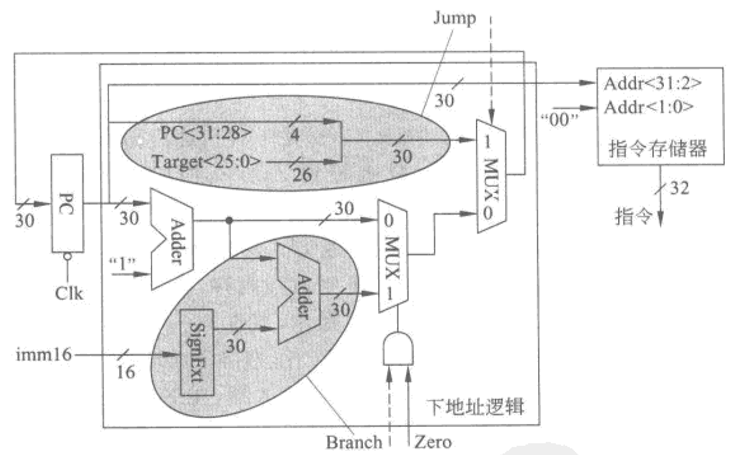
Lw/Sw都为I-型指令，但是与之前的不同，他们都要对数据存储器进行操作，故增加一个数据存储器，同时用MemtoReg来控制是将ALU运算的结果存入寄存器堆还是将取得的数据存到寄存器堆；

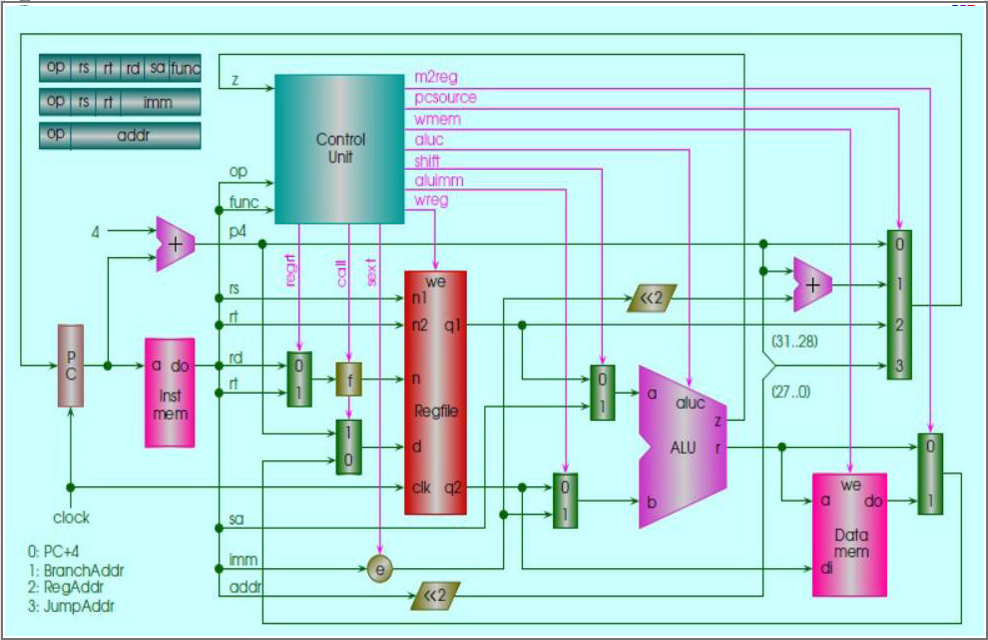
1. **分支指令的数据通路**

根据不同的条件进行分支的转移，转移的实质是CPU下一个始终周期运行的地址的不同，所以具体的实现由下地址逻辑部件完成；在分支指令中，下地址逻辑的输出就是下一条指令的地址，输入为PC、Zero标志、立即数imm16和控制信号Branch；顺序执行时：PC<31:2> <= PC<31:2>+1；转移执行时：PC<31:2> <= PC<31:2>+1+SignExt[Imm16]；这时得到的只是地址的前30位，因为指令地址时4的倍数，所以要左移（乘四）两位得到最终的指令地址；

1. **无条件转移指令的数据通路**

J-型指令中给出了要跳转的26位目标地址，表示下一条指令的执行由指令决定，计算方法为：PC<31:2> <= PC<31:28> || target<25:0> 之后再将30位的指令左移得到最后的32位地址；

将各种取指令合并即为：

1. **完整的单周期数据通路**

ControlUnit: 控制部件，对每一个部件在不同指令下的控制；

Instmem: 指令储存器，本实验中内部存有一段预先设置的代码

Regfile： 寄存器堆

DataMem: 数据存储器

e: 扩展器，对imm16进行不同的的扩展（零扩展、符号扩展）

<<2: 左移两位

1. **各数据通路具体代码设计实现**

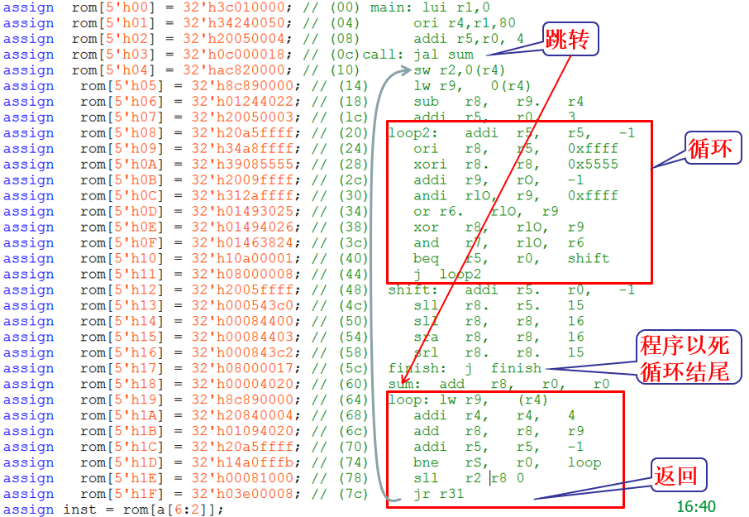
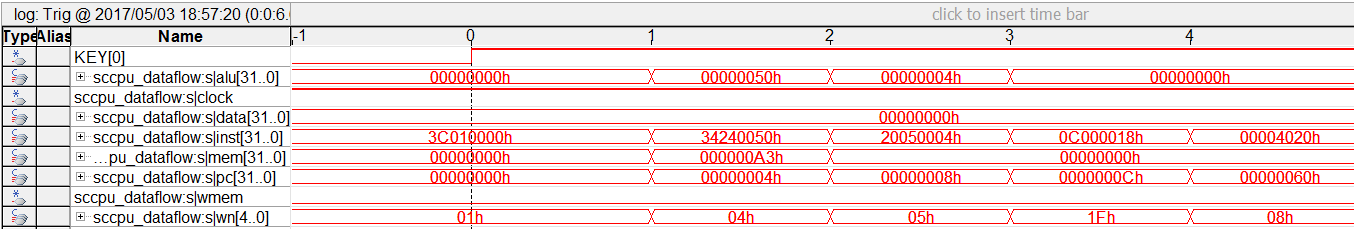
图片包含 室内, 墙壁

描述已自动生成图片包含 天空

描述已自动生成图片包含 天空

描述已自动生成图片包含 时钟, 天空, 墙壁, 物体

描述已自动生成下面依次为：R-型指令执行过程、I-型指令执行过程、Load指令执行过程、Store指令执行过程、Branch指令执行过程、Jump指令执行过程：

1. **运行指令存储器中的一段程序**
2. **结果以及分析讨论**
3. **实验结果**

通过SignalTap调试工具的分析可以看出，PC信号的改变的情况是：00h->04h->08h->0Ch->60h…

前面PC在一个时钟周期会增“1”，也就是加4，而后面改变为60h的原因是与指令存储器中存储的程序有关，在程序中有这样一句语句：call: jal sum 这句指令，此时执行了无条件转移指令，PC指向sum所在的位置也就是60h处；除此之外，其他的部件的变化情况都与程序有关；

1. **分析讨论**

通过这次实验，我深刻的认识到了CPU 内部结构的组成与设计的原理，从最初的运算需求出发，分析得到3种不同的指令格式，对于这些指令提取其中公共部分，从一开始的R-型指令开始，逐步加入I-型指令、J-型指令以及相关的下地址逻辑部件等等，得到最后的单周期CPU数据通路；之后利用Quartus软件以及FPGA的帮助下，通过VerilogHDL语言实现的单周期CPU工程，深入的认识到了CPU内部各部件之间的联系，数据的传输、计算、控制等等内容，还在SignalTap 的帮助下理解了一段储存在指令存储器中的程序，同时也进一步的认识到了程序每一步执行时不同寻址方式的计算方式以及实现；当然这次实验我也遇到了很多理论课上一直为解决的问题以及未曾在意的问题，在同学老师的帮助下，我基本都弄明白了它们，总的来说，这次实验在各个方面对我受益匪浅。