单周期CPU设计

**目标**

设计一个单周期**32位**[MIPS](file:///D:\\%5CInternt_of_Thing%5Ce_book%5C%E8%AE%A1%E7%AE%97%E6%9C%BA%E7%BB%84%E6%88%90%E5%8E%9F%E7%90%86%5CA03_%E2%80%9C%E7%B3%BB%E7%BB%9F%E8%83%BD%E5%8A%9B%E5%9F%B9%E5%85%BB%E5%A4%A7%E8%B5%9B%E2%80%9DMIPS%E6%8C%87%E4%BB%A4%E7%B3%BB%E7%BB%9F%E8%A7%84%E8%8C%83_v1.00.pdf) CPU，依据给定过的指令集，设计核心的控制信号。依据给定的数据通路和控制单元信号进行设计。

**指令集**

实现7条指令子集：**ori，lui，addu，sub，bne，lw，sw**，假设不会溢出

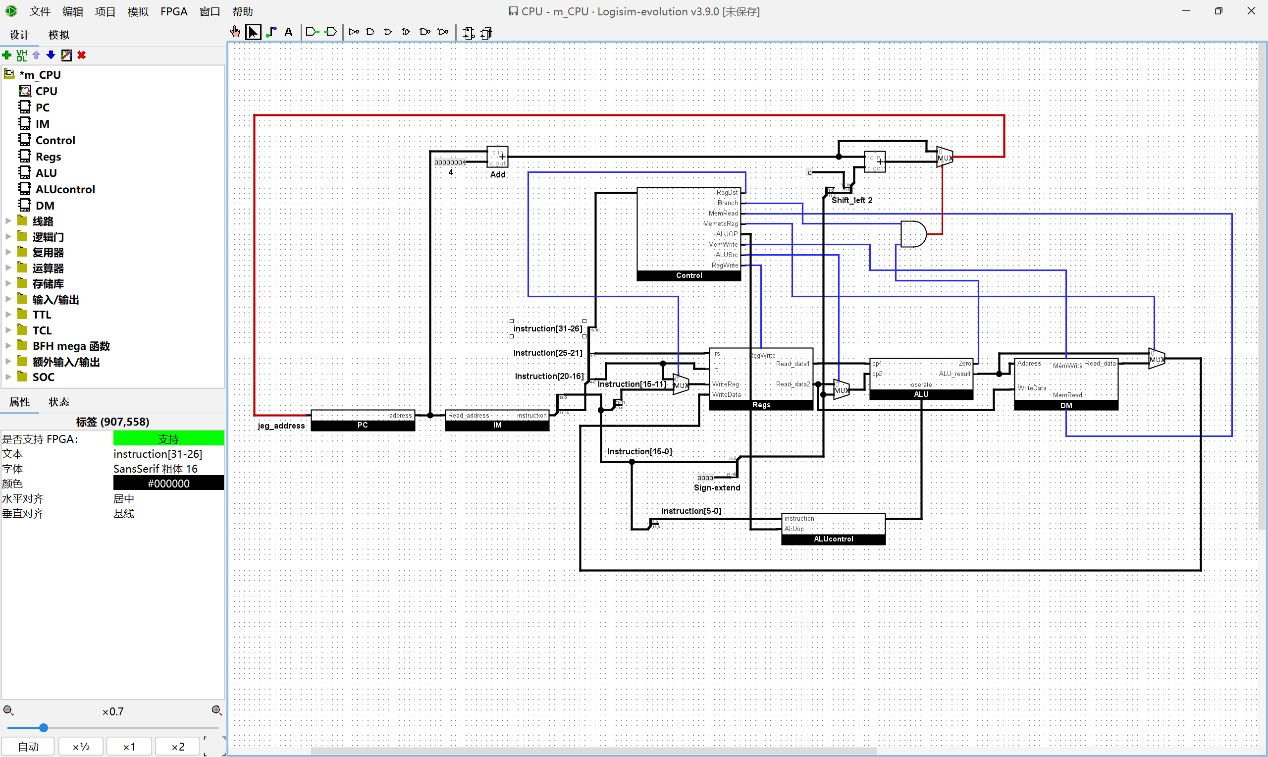
**指令的类型**

**R型**： addu, sub（opcode为0）

**I型**：ori,lui,lw,sw, bne

**设计数据通路**

**思路**

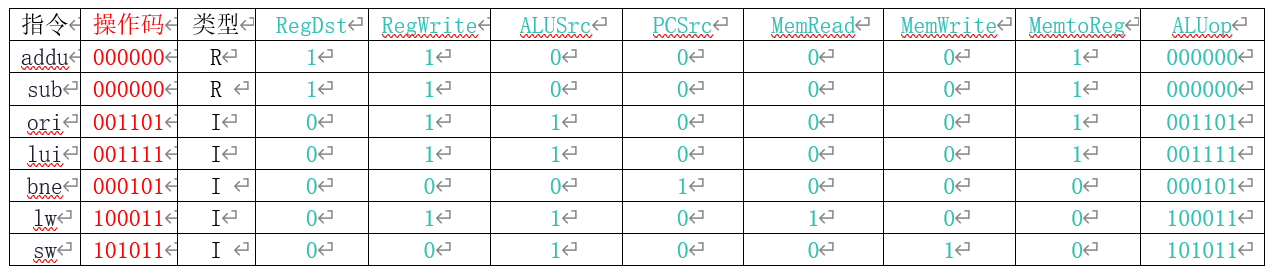
* 依据流水线一般，分为五个阶段分别设计
* 对于一个 MIPS指令包含如下5 个处理步骤：
  + **IF**: 从指令存储器中读取指令
  + **ID**: 指令译码的同时读取寄存器。 MIPS 的指令格式允许同时进行指令译码和读寄存器
  + **EX**: 执行操作或计算地址
  + **MEM**: 从数据存储器中读取操作数
  + **WB**: 将结果写回寄存器
* logisim实现，但内部未完善
* 

**对于控制信号**

**Control**

1. RegDst:写寄存器组的**地址来自rt还是rd**
   1. 当写地址来自rt,即**ori, lui,lw**指令的rt时，DegDst为状态 **0**
   2. 当写地址来自rd,即**addu, sub**指令的rd时，DegDst为状态 **1**
2. RegWrite:是否**往寄存器里写数据**
   1. 往寄存器里写数据时，即**ori, lui, addu, sub, lw** 为状态 **1**
   2. 不写时，即**bne, sw** 为 **0**
3. ALUSrc：第二个**操作数来自寄存器还是立即数扩展**
   1. 第二个**操作数来自寄存器**时 ,即**addu, sub, bne**，为**0**
   2. 第二个**操作数来自立即数扩展**时，即**ori, lui, lw, sw**,为 **1**
4. Branch(在表里为**PCSrc**):是否为**分支指令**
   1. 不是分支指令，为 **0**
   2. 是分支指令 ，即**bne** 为 **1**
5. MemRead： 是否**读存储器**
   1. 不读，为**0**
   2. 读寄存器时，即**lw** ,为 **1**
6. MemWrite :是否**写存储器**
   1. 不写存储器 ，为 **0**
   2. 写存储器时， 即**sw**, 为**1**
7. MemtoReg:写回寄存器的值**来自ALU输出结果还是存储器输出**
   1. **ALU输出作为结果寄存器输入**，即**ori, lui, addu ,sub**, 为**0**
   2. **存储器输出作为结果寄存器输入**， 即**lw, sw** ,为**1**
8. ALUOP:**控制ALU的操作**
   1. 将操作码（opcode)传给ALUcontrol

所以对于上述指令集



**ALUcontrol**

