**三步教你用Verilog写一个CPU**

<https://blog.csdn.net/iteye_5971/article/details/82612212>

第一步：牛刀小试

1. 教学大纲

第一步

指令集设计与五级流水线的实现

第二步

内存设计与CPU测试

第三步

指令冲突避免

1. 开发基础

课程要求：数字电路、计算机组成原理、程序设计

编程语言：Verilog

开发平台：xilinx ISE

FPGA开发板：Nexys3

1. 实现目标

本文实现的CPU是一个五级流水线的精简版CPU(也叫PCPU，即pipeline)，包括IF(取指令)、ID(解码)、EX(执行)、MEM(内存操作)、WB(回写)。

指令集：RISC

指令集大小：2^5 = 32

数据宽度：16bit

数据内存：2^8×16bit = 256个16bit

指令内存：2^8×16bit = 256个16bit

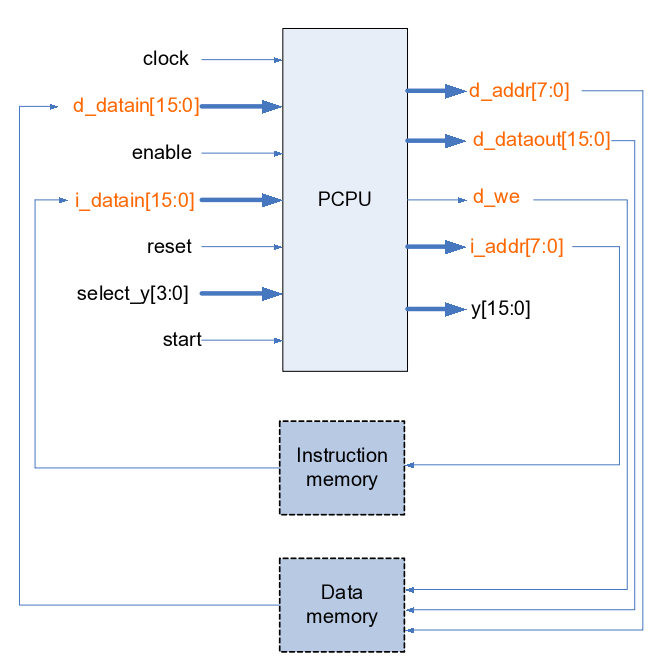
通用寄存器：8×16bit

标志寄存器：NF(negative flag)、ZF(zero flag)、CF(carry flag)

控制信号：clock、reset、enable、start

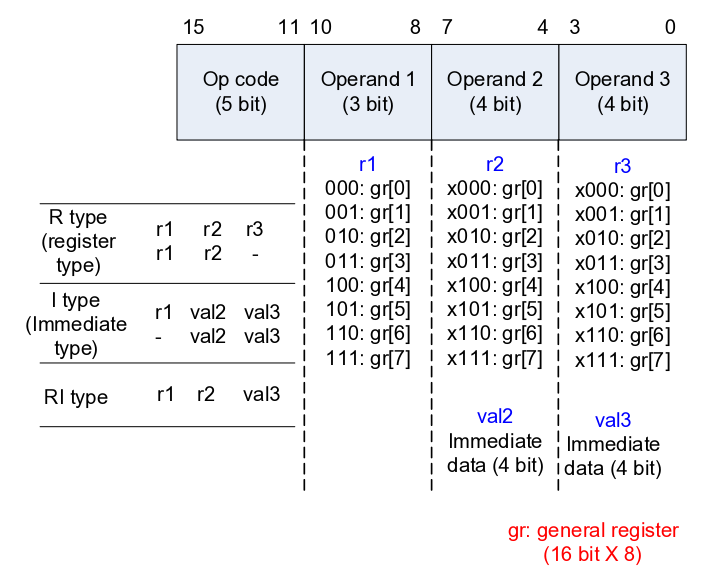
1. 介绍与设计
   1. CPU顶层视图

第一步只要求实现简单的五级流水线，不要求实现指令内存、数据内存模块，因此CPU内部与内存有关的信号都简化为输入输出信号了，CPU的顶层视图看起来应该如下图，其中select\_y、y信号是跟CPU板级测试有关的，这一步暂且没用到。



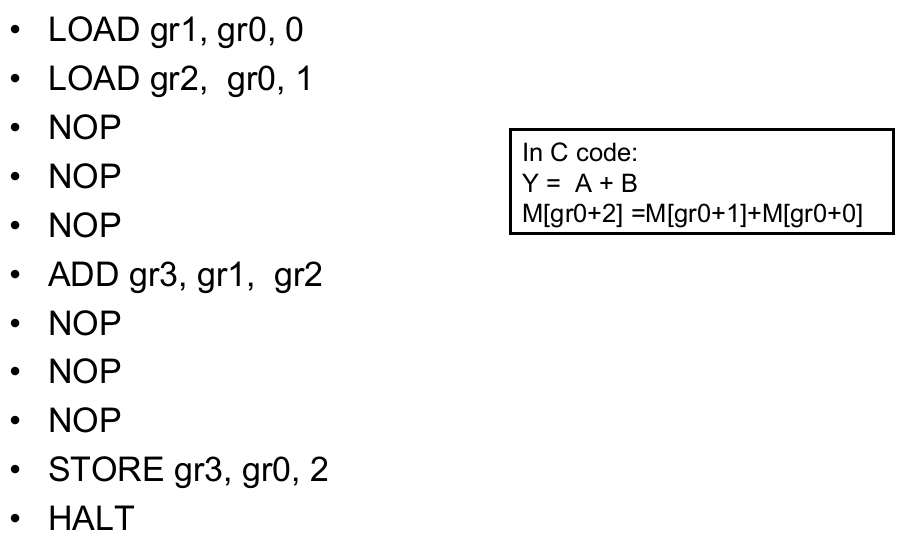
* 1. 指令集

指令为三地址格式，操作码长度5bit，根据操作数的不同可以把指令分为三种类型，即寄存器类型R type、立即数类型I type、混合类型RI type，不过后面在代码编写的时候，为了方便，会依据其它标准进行划分。

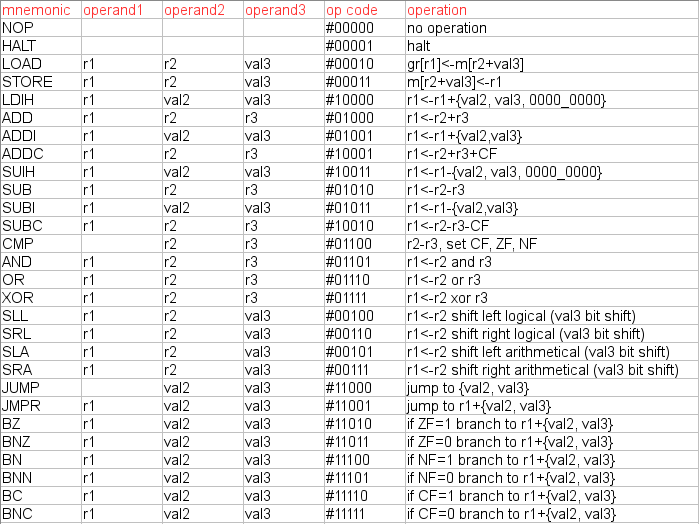


规范一下表示方式，r1或者gr[r1]表示访问寄存器r1，m[r2+val3]表示访问r2+val3这个地址，{val2,val3}表示立即数访问，val2为MSB，val3为LSB。

汇编代码示例：

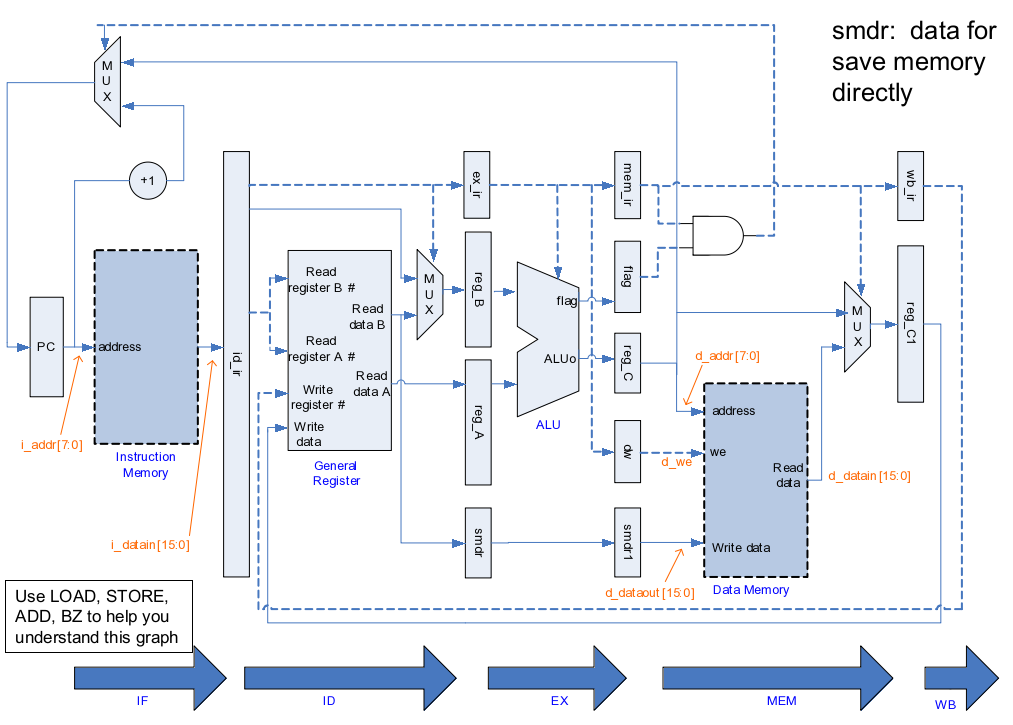


本文设计一共实现了28条机器指令，剩下未用的4个操作码(10100,10101,10110,10111)可自行补充为其它操作，比如自增INC、自减DEC。这里指令的编码是比较随意的，而且由于代码实现中使用了宏定义，因此可以任意更改指令的编码，不过如果想做进一步的优化，就要仔细考虑编码方式了。以下是指令集的具体格式与操作，设计CPU的时候有两张图是必须时刻看着的，我都把它们打印出来，这是其中一张。



* 1. 五级流水线

除了指令集之外，设计CPU最重要的就是下面这张CPU块级电路图，五级流水线的代码实现都必须依赖于这张图，因此必须理解图中每一步的作用。



图中所有矩形框标出来的都是CPU内部的寄存器，整个电路图展示了CPU内部指令以及数据的流动方向。每到时钟上升沿，上一级流水线的寄存器的数据就会经过中间的组合逻辑电路流动到下一级流水线的寄存器，因此，5个时钟周期之后一条机器指令便执行完毕了。

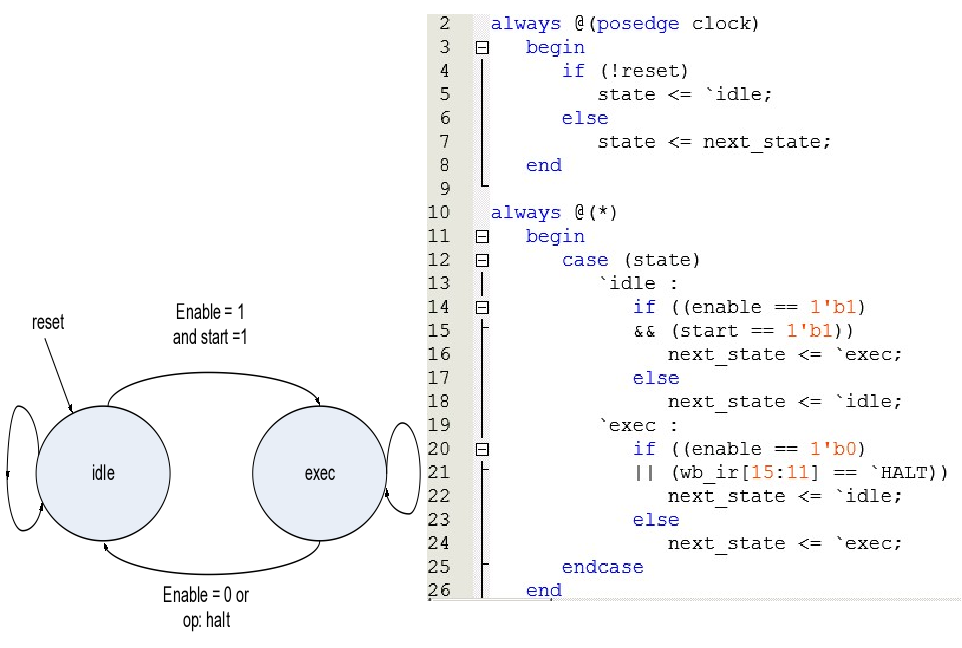
简单描述一条指令的执行过程就是，

* 首先根据PC的值到内存中取一条指令，
* 解码指令提取两个操作数
* 进行运算，
* 根据指令功能以及运算结果决定是否访问数据内存以及如何访问，
* 最后同样根据指令功能决定是否要进行回写操作，即修改寄存器的值。

下面将分别讲解CPU控制以及五级流水线每一级的行为，为了简单起见，这里仅考虑NOP、HALT、LOAD、STORE、ADD、CMP、BZ、BN这几条指令，明白了流水线的行为之后再加上其它的指令也是一样的道理

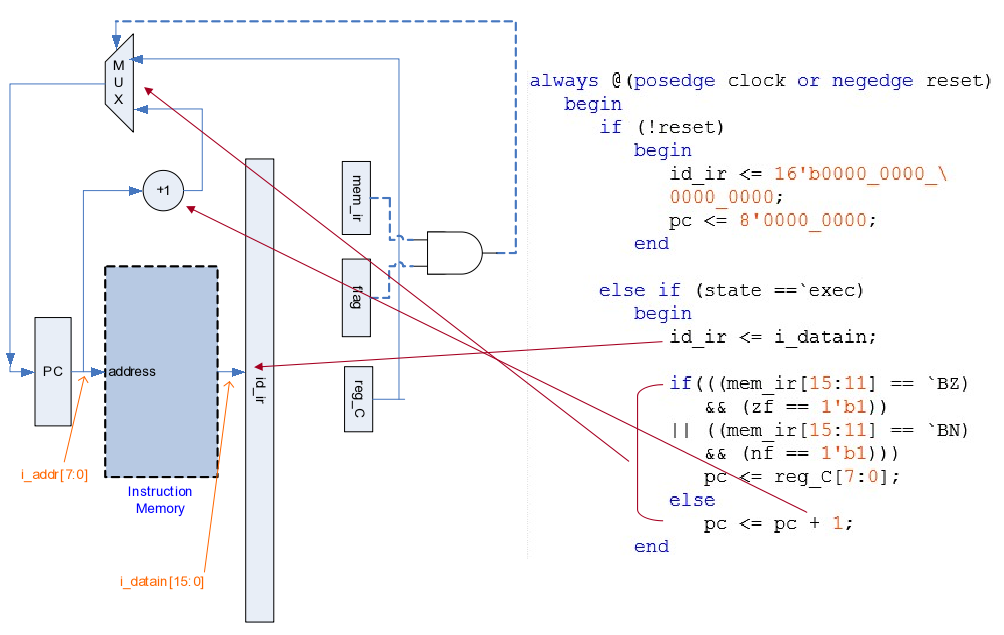
* 1. CPU控制

CPU控制自然是基于状态机，只有两个状态idle和exec，CPU在idle状态下只有enable、start同时使能才会进入exec状态。



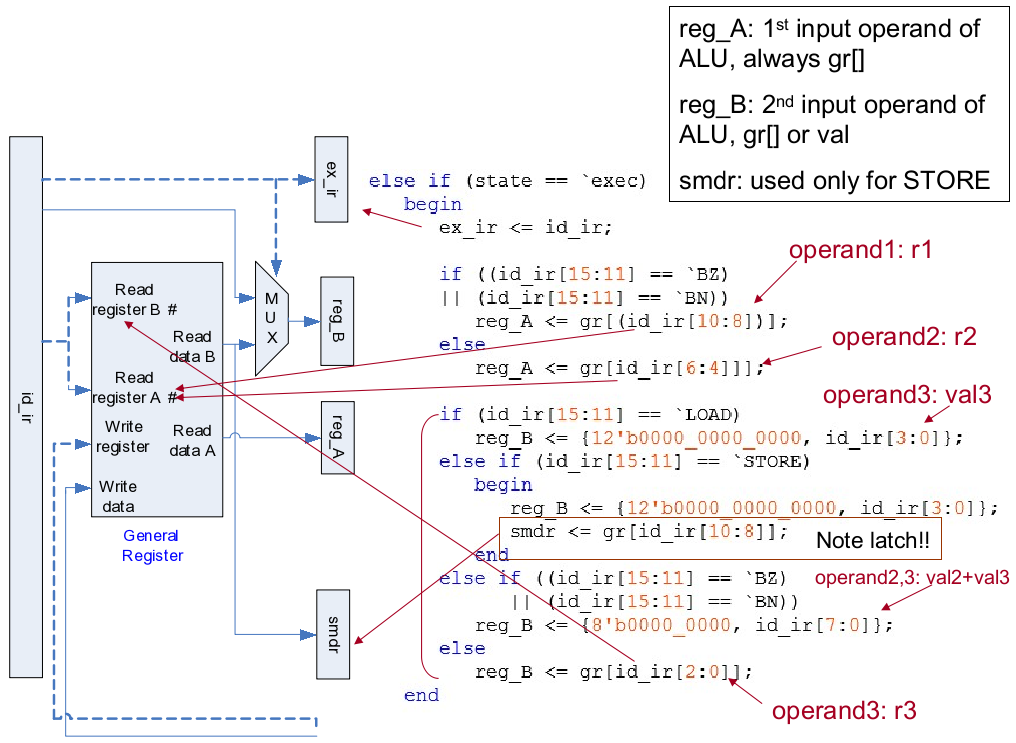
* 1. IF

IF阶段的任务就是要根据PC的值从指令内存中读取一条指令，并且设置下一周期PC的值(指令可以顺序执行，也可以跳转到某个特定的地址)。因为读取内存是内存模块实现的功能，因此这里CPU只需要给出指令地址i\_addr就能得到对应的指令i\_datain。



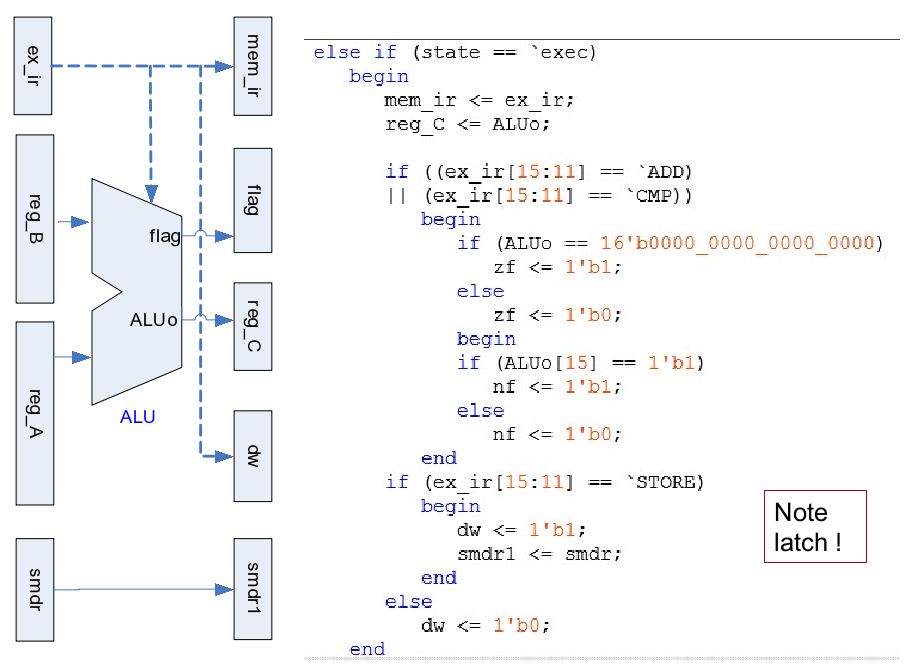
* 1. ID

ID阶段要根据指令的功能(即操作码)从指令中提取对应的操作数，操作数可能来自通用寄存器r0-r7，也可能是立即数。另外如果指令是STORE指令，也要准备好要存储到内存中的数据。



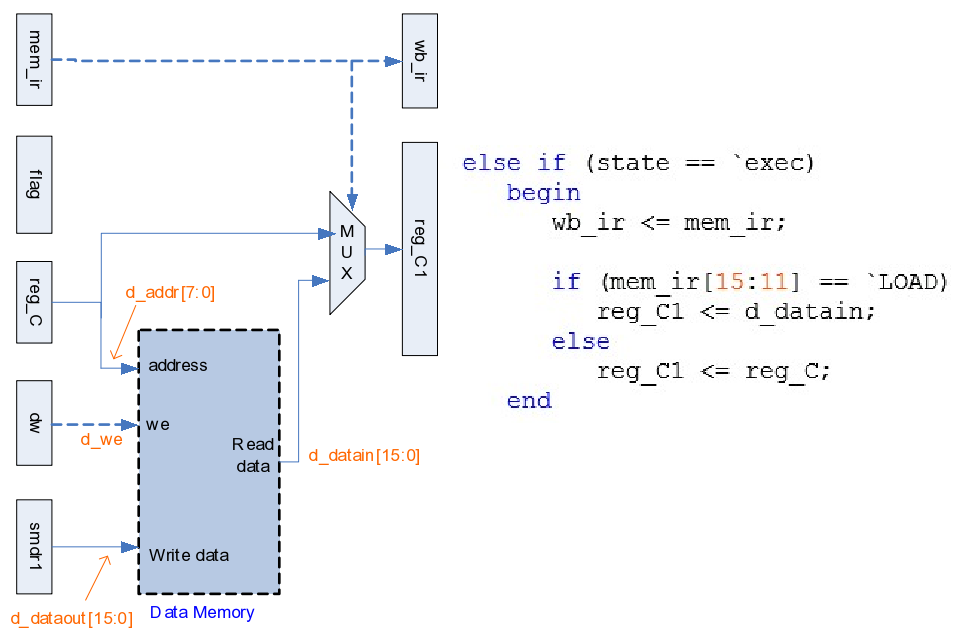
* 1. EX

EX阶段执行的是ALU运算和标志寄存器设置，另外如果是STORE指令也要给出内存写的使能信号dw以及将要写到内存中的数据smdr1。



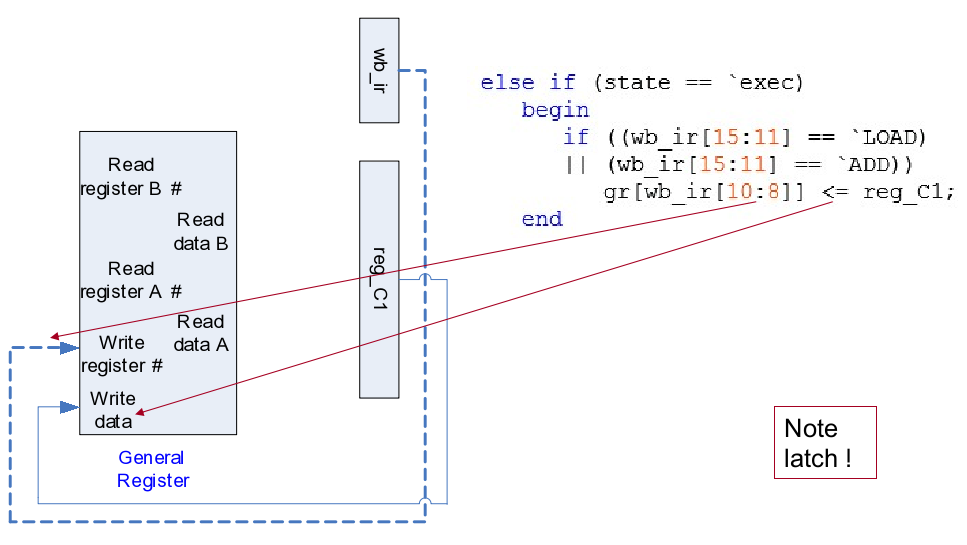
* 1. MEM

MEM阶段要根据指令功能和上一阶段的运算结果(内存操作的时候作为内存地址)决定是否要访问内存以及如何访问，只对需要内存操作的指令有效。



* 1. WB

WB阶段同样根据指令的功能以及上一阶段的结果决定是否要修改寄存器的值以及如何修改，只对需要修改寄存器值的指令有效。



* 1. 代码实现

// def.v

// pcpu.v

第二步：渐入佳境

1. 实现目标

这第二步就是教你怎样把代码转变为一个可运行的CPU，然后对CPU进行测试。CPU测试有多种方法，我们从最简单粗暴的实时赋值开始，到内存模块的实现，从软件仿真到硬件的板级验证，从开关选择的信号LED显示到基于VGA的信号显示，这些都是这一步要完成的。

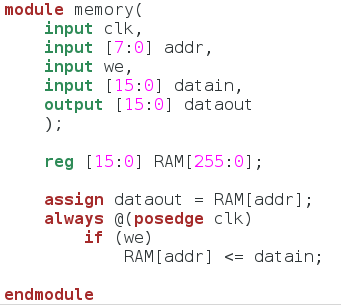
1. CPU测试方法一：实时赋值(软件仿真)

// pcpu\_test.v

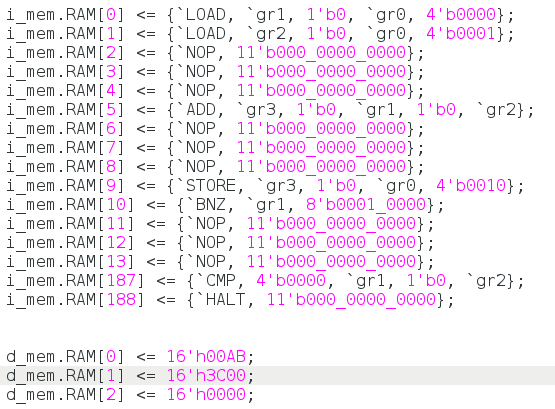
在ISE里新建一个pcpu的测试文件，然后只要在合适的时候分别给i\_datain、d\_datain赋值就可以了，这是最简单粗暴的方法。

1. CPU测试方法二：内存实现(软件仿真)

内存其实就是一个 reg 数组，读内存用组合逻辑电路，写内存用时序电路，如下图代码所示。对内存的初始化可以通过 reset 信号，或者软件仿真的时候可以直接在 test bench 文件的 initial 里面初始化。



这时候，pcpu的测试文件就要另外再通过memory.v实例化两个内存模块i\_mem、d\_mem分别用作指令内存和数据内存了，pcpu模块与内存有关的所有信号(i\_datain、d\_datain等等)都要连接到这两个模块。initial初始化内存的方式如下图：



另外一种reset信号初始化内存的方式需要分开定义指令内存i\_memory.v和数据内存d\_memory.v，然后在pcpu的测试文件里面实例化这两个文件的模块。d\_memory.v大致如下

