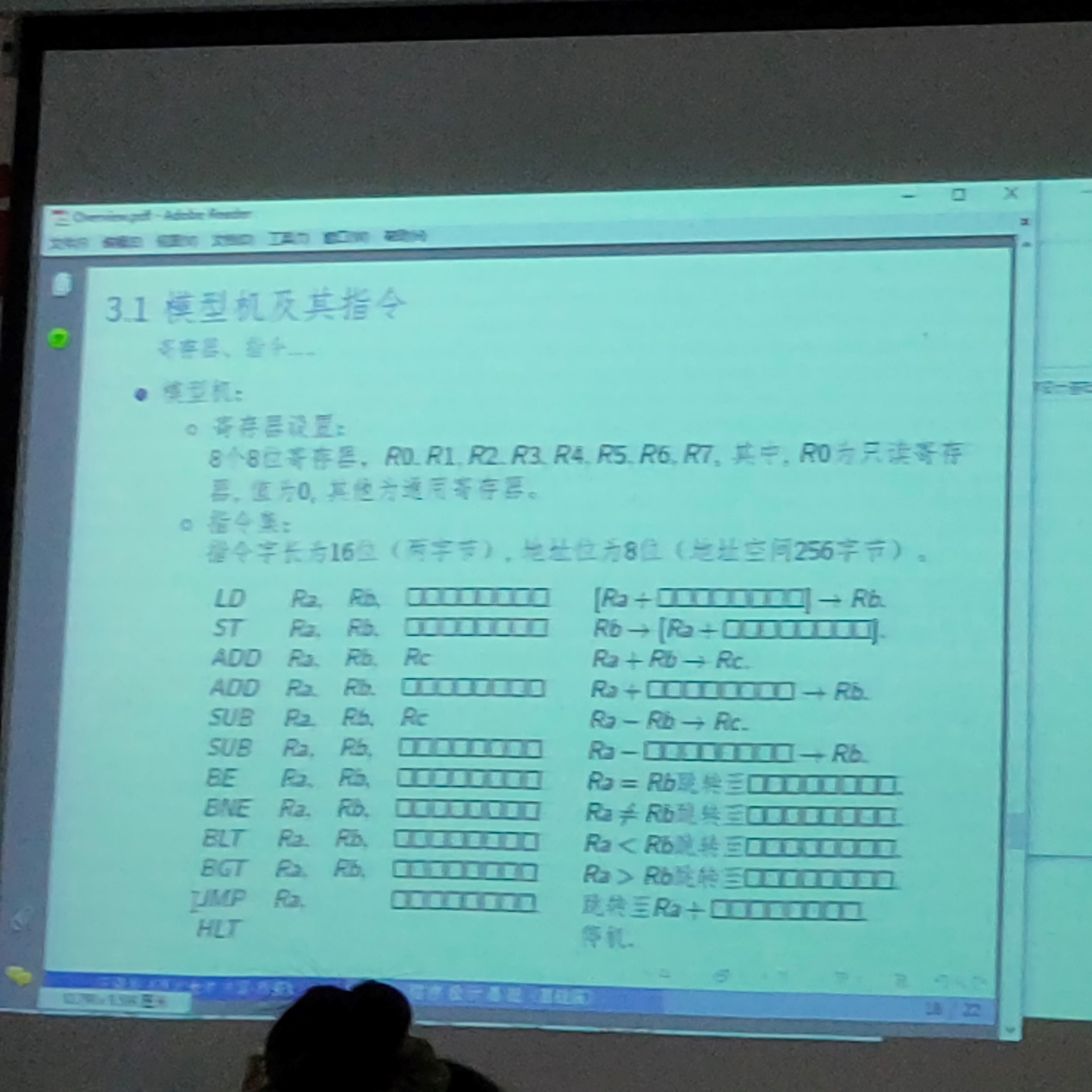
# 假定与已知

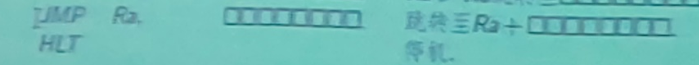
## 0.指令架构设计



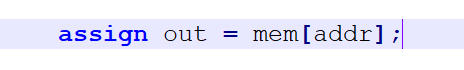
这个指令做一个小改动

不用Ra+立即数作转移的目的地址，

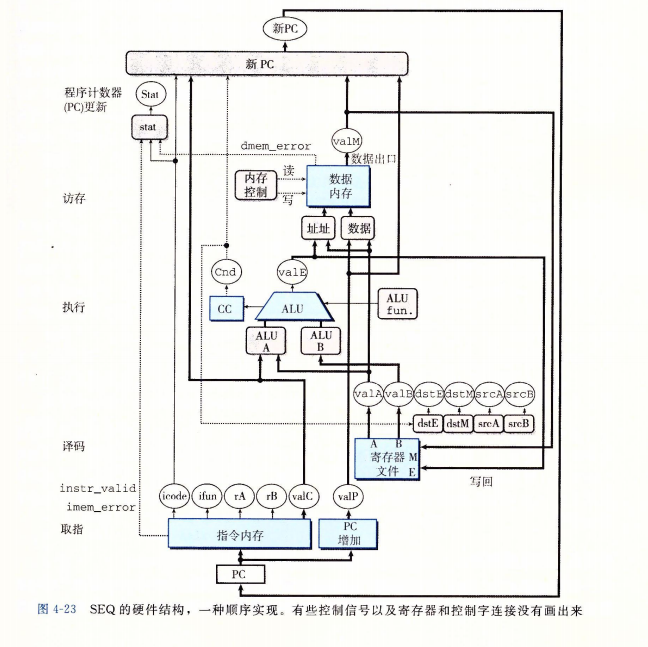
直接用立即数作目的地址



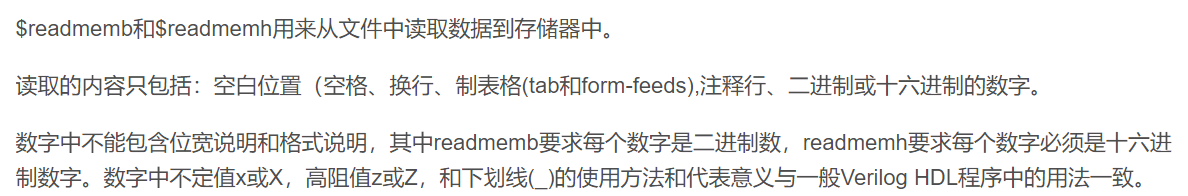
## 1.记住所有线网型变量都在用连续赋值



## 2.假定指令写在ROM中（指令都是预先写好的）



## 3.Verilog系统函数——readmemh

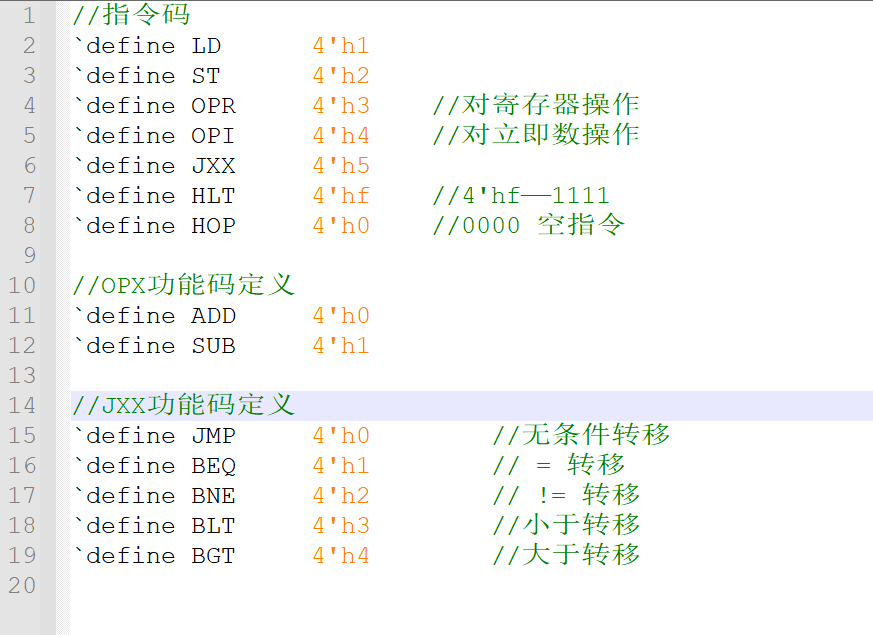


# 头文件——定义的一些宏

## 对指令的定义

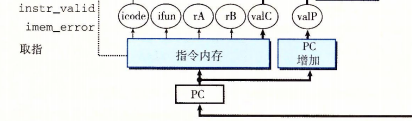
注意ADD和SUB指令都需要功能码，所以直接在指令宏定义部分把他们统一定义成OPX指令; 转移指令宏定义成JXX

那么对于OPX指令， JXX指令，我们需要用低四位表示功能码

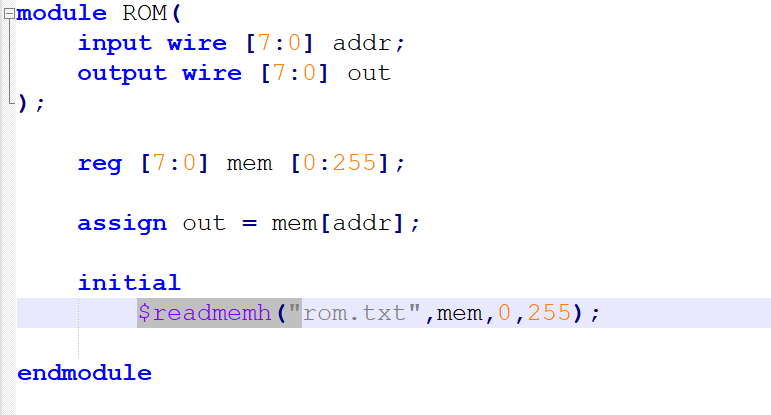


# ROM

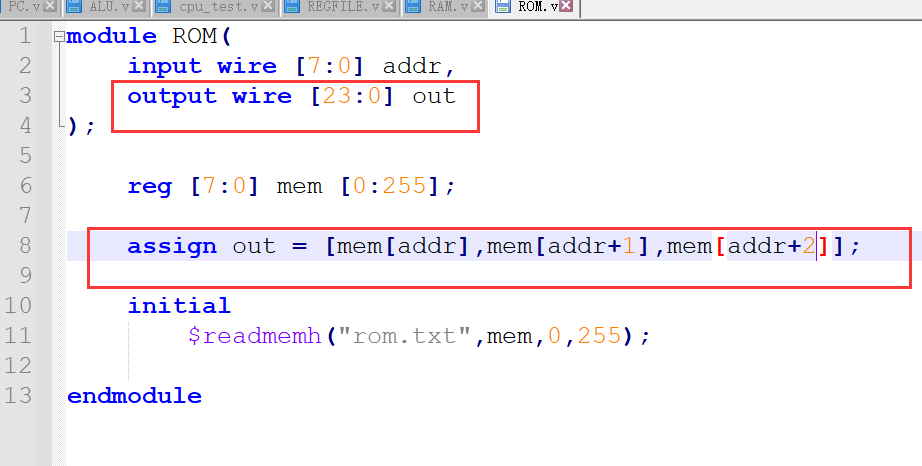
存储指令



像下面这样一次读一个字节显然是不合适的



一次要连续读出3个字节的值（1B-指令码和功能码各半个字节 1B-Ra和Rb 1B-立即数，第二个字节高四位放Ra低四位放Rb——空余的两位可以以后用作寄存器的扩充嘛）



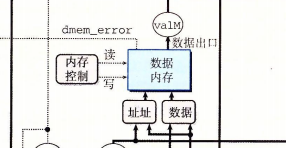
# RAM-data memory

存储数据 data memory

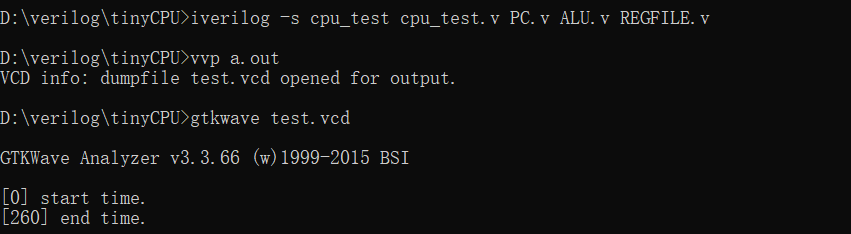
输入：控制信号两个-控制读，控制写，地址addr ，输入数据-in

输出：out

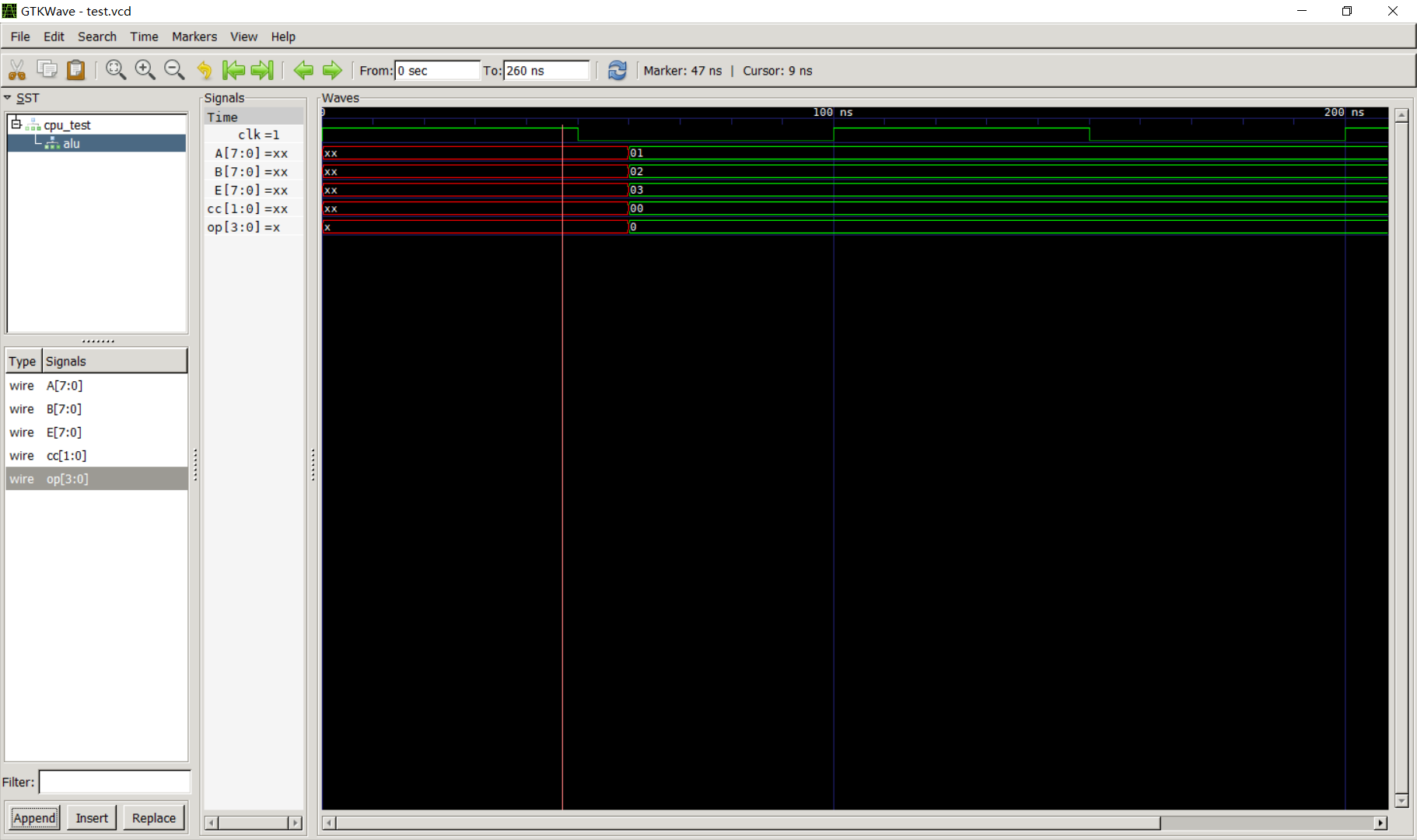
硬件实现：



# Test

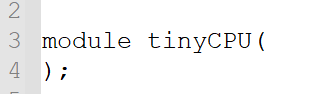


# 查看波形文件



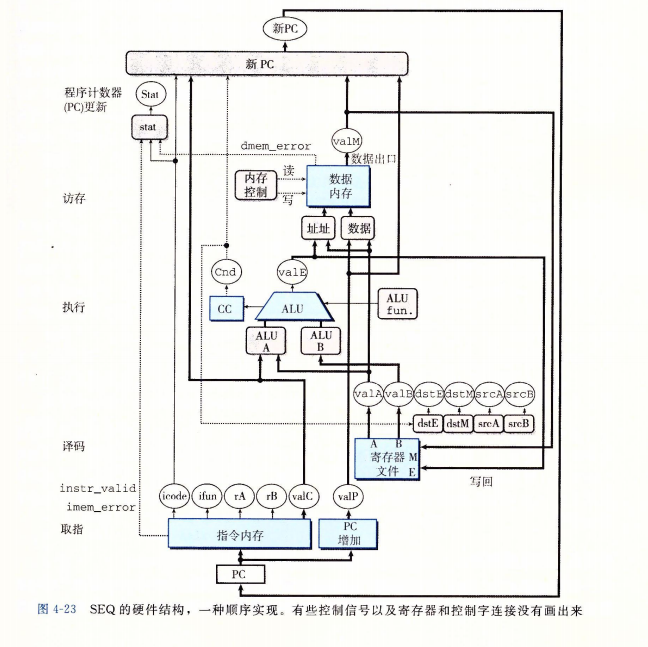
# TinyCPU整合

## 0.整合



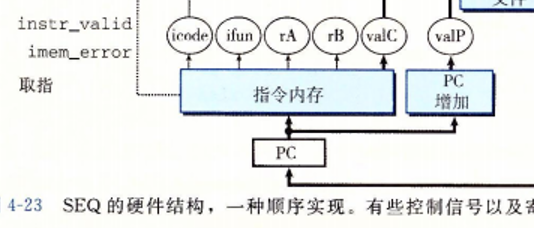
要接两个外部器件ROM,RAM

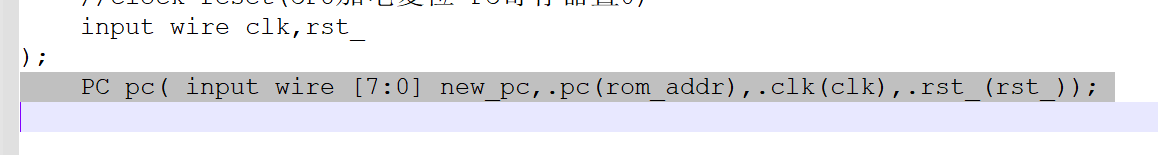
其他的器件包含在CPU内部



参照这个电路来描述

## PC

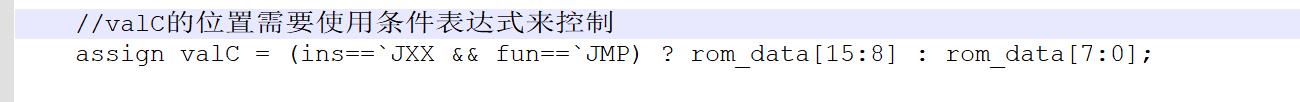




使用宏定义来定义（定义指令中各字节的意义，定义成宏的形式方便使用）

不能用宏定义定义valC在指令中的位置，因为在JMP指令中valC的位置是特殊的

valC rom\_data[7:0] //第三个字节存储valC，但是JMP命令 第二个字节是valC



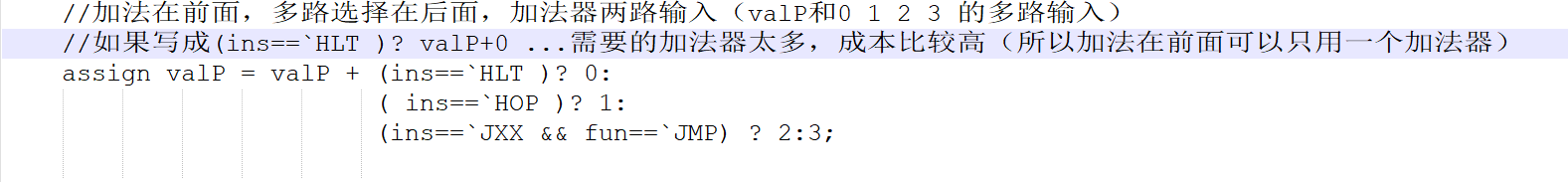
ValP的值也值得研究

·比如JMP 转移指令 需要加两个字节；（之前有改动：把跳转目的地址从Ra+立即数改成了直接JMP到立即数指定的目的地址）

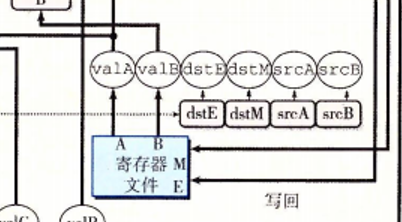
·其他的指令，加三个字节

··但是当指令值HLT\HOP时，是加一个字节

如下：**（注意加法放在前面而不是每个条件中，是为了节省实现时的成本）**

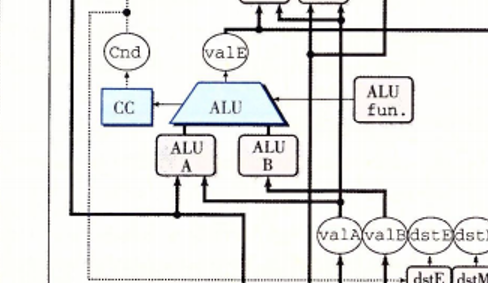


## RF



## ALU

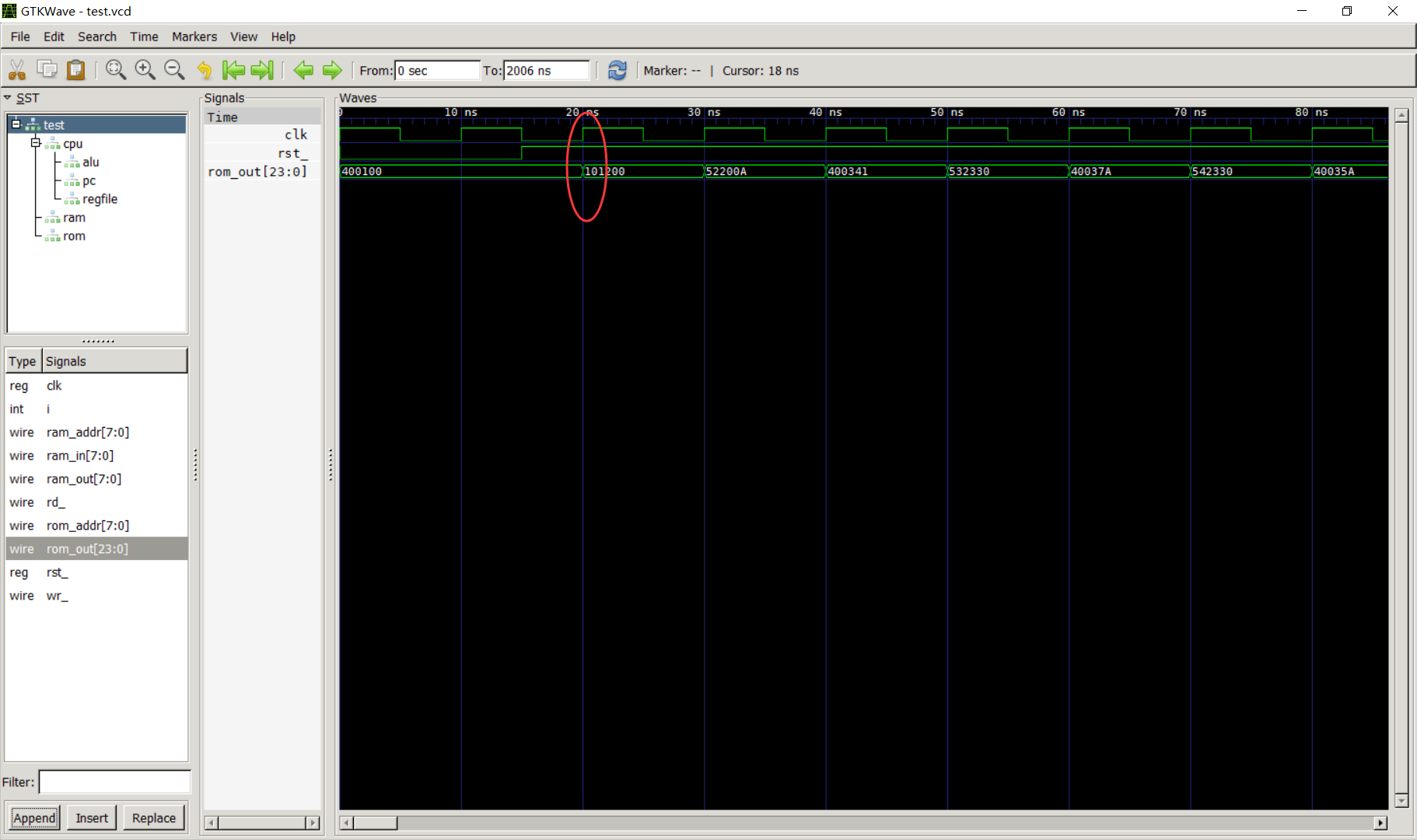
AluB的来源非常



# 仿真波形观察：

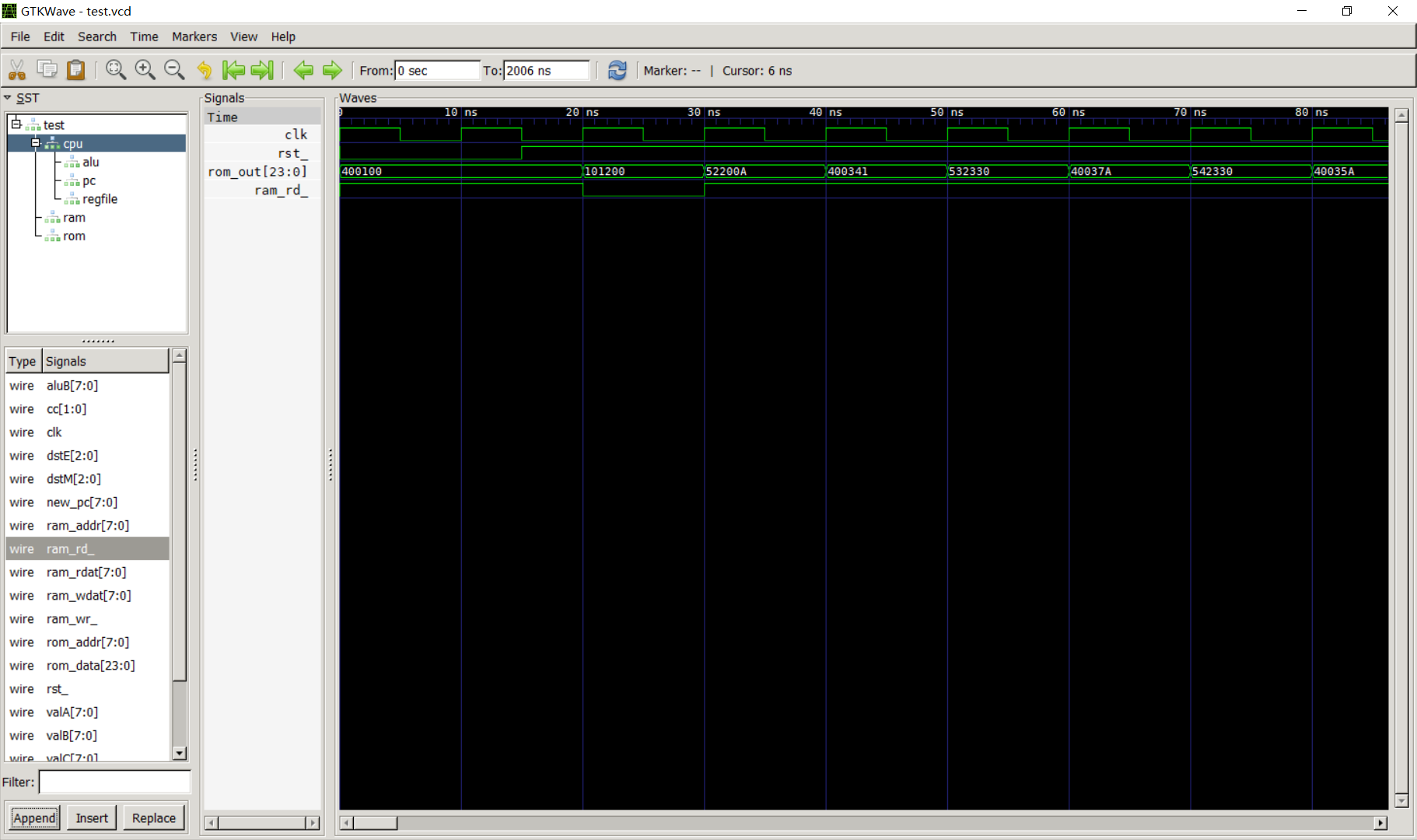
## 观察test 的信号-rom\_out[23:0]

在rst\_置为1之后，才能成功把把下一条指令读入



另外，你也得从rom\_out这个信号的波形输出看出程序是如何执行的，比如执行到52200A（BNE R2,R0,#2）就跳转到400341（#2:ADD R0,R3,’A’）.

## Cpu 的信号 ram\_rd\_



在执行到101200(LD R1,R2,0)时，ram的读允许信号rd\_有效，load指令执行时才可以从ram中读取数据

