

x是address的alias,

res4是预定了4byte memory

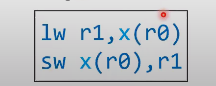
如果你不同scope有两个variable X, 应该加前缀，因为moon没有scope的说法

但是label只支持variable，

只有使用offset才能支持recursive function,object

绝大部分分数都基于tags

但是stack也有一部分分



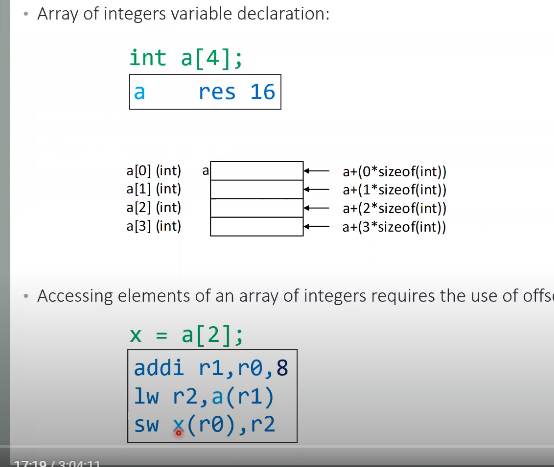
怎么得到x的值

**lw**就是Load的意思，x(r0)，x就是alias, r0就是offset，一般我们把它设置成0，这样就可以直接指向x，我们就可以吧x中的数据load到r1中

通过这个指令我们从 x得到了值，r0

**sw** 就是store的意思，把r1的值存进x(r0)

怎么access array/object里的某一具体元素

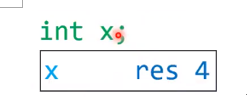


例如我们要access

**a[2]**

先用r0为基础加上他的byte，

注意r0与r1并不是4个byte，他们只是一个开始的**Point，扮演的是个offset**

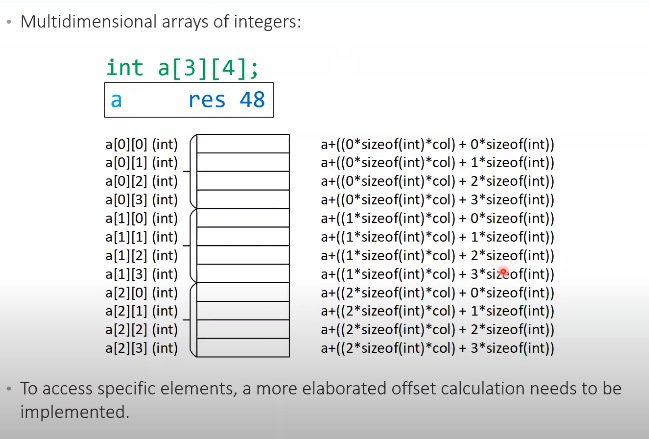


**只有x,a这种tag才记录了具体多大**

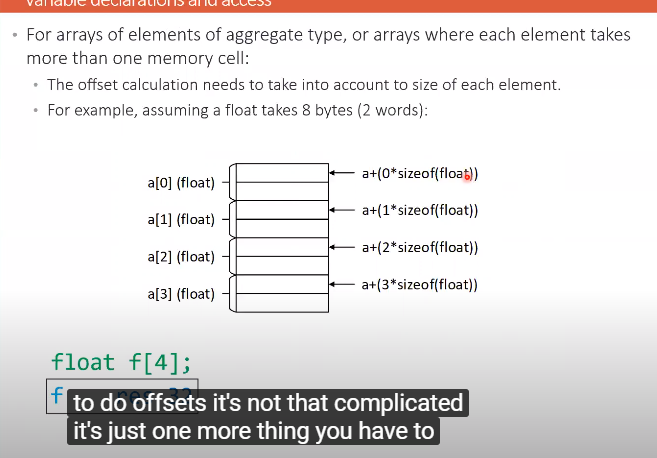
这时r1代表着offset为8，也就是a[2]

我们可以用r1了

最后使用temp r1, r2,存回x

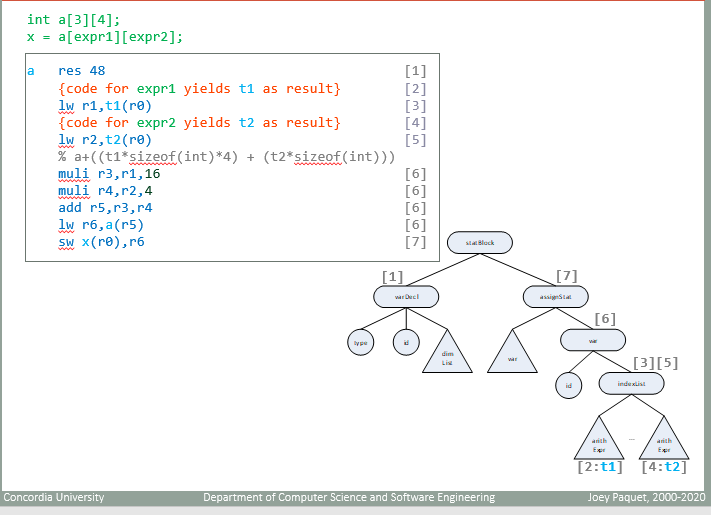


你需要一个function来快速生成array的offset



这个function需要考虑array一个元素的大小，例如object，这种

float需要8byte



首先如果你把memory reservation 装进了executable code， 会让你整个machine crush

所以整体结构是这样的

function

class.......

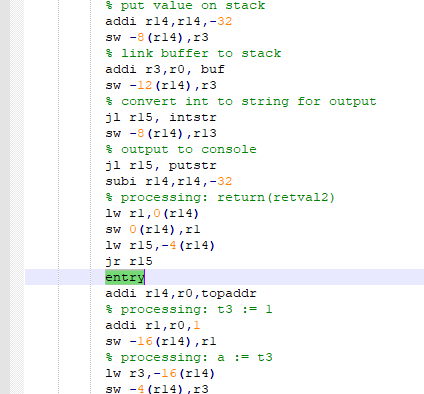
entry

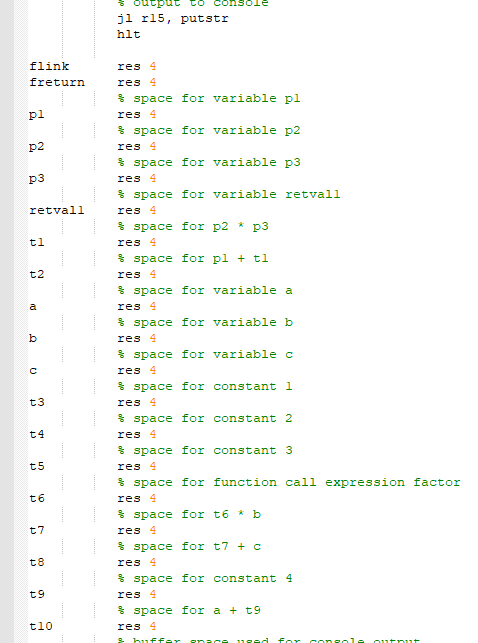
main part

hlt

然后开始数据存储区

也就是每一个变量占据多少





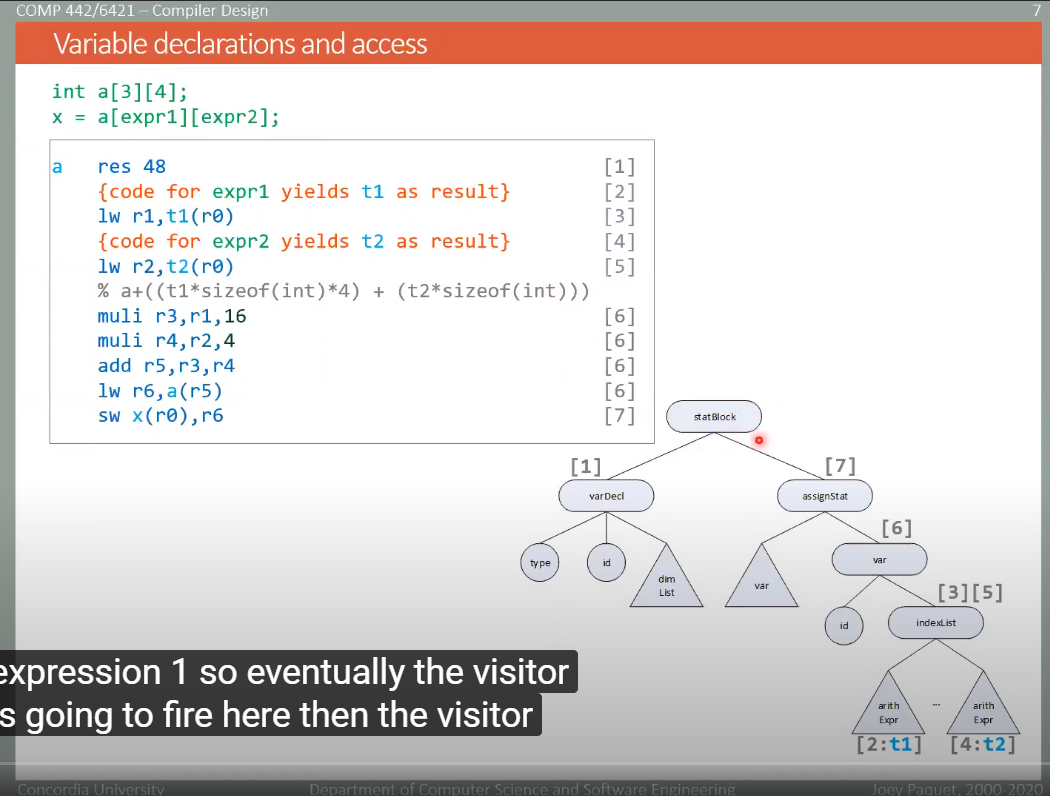
解决方案



分为DataCode

以及ExecCode

最后再把两部分相append



res 48怎么算出来的

他这里说看varDecl，看第一个node type再看dimList

但我觉得更应该看SymbolTable

Visitor Visit的顺序还是一样的

我们左边生成完以后，

会一直到右边的t1，

t1指的是temporary register ，在我们使用expr1得到t1这个结果以后，把这个结果附给r1

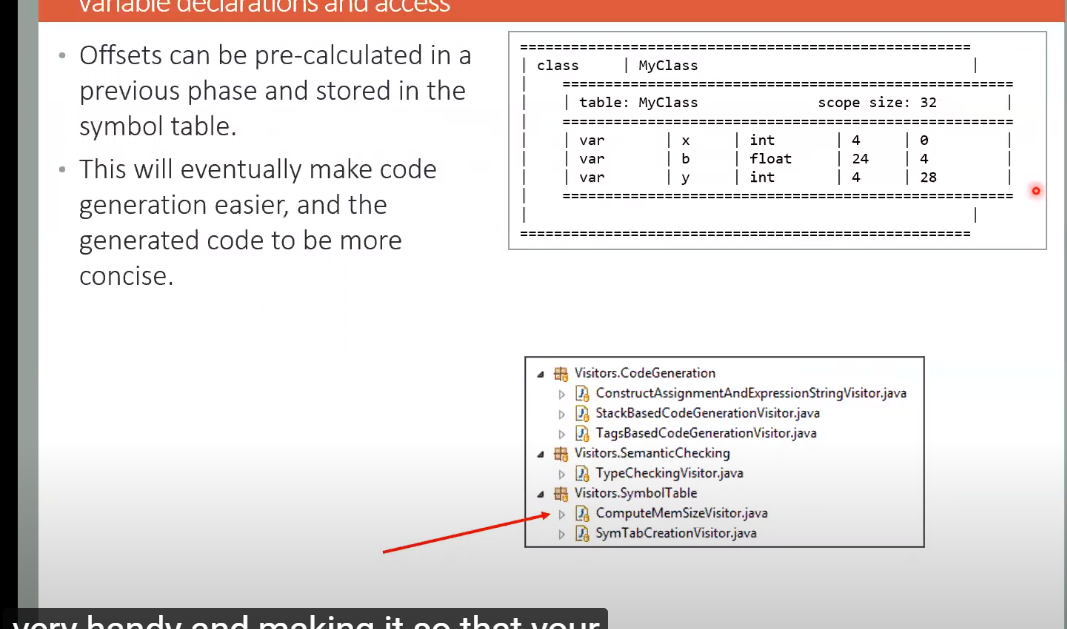
r2同样

然后

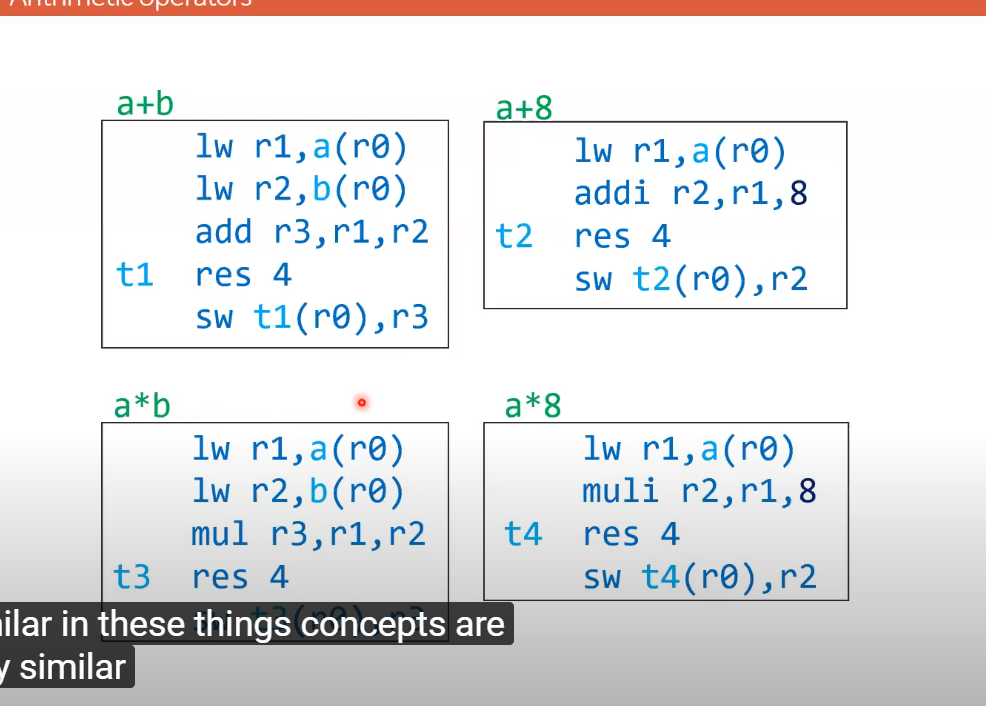
r1r2拥有两个能计算的值

可以利用他们计算出 他在Array中的offset

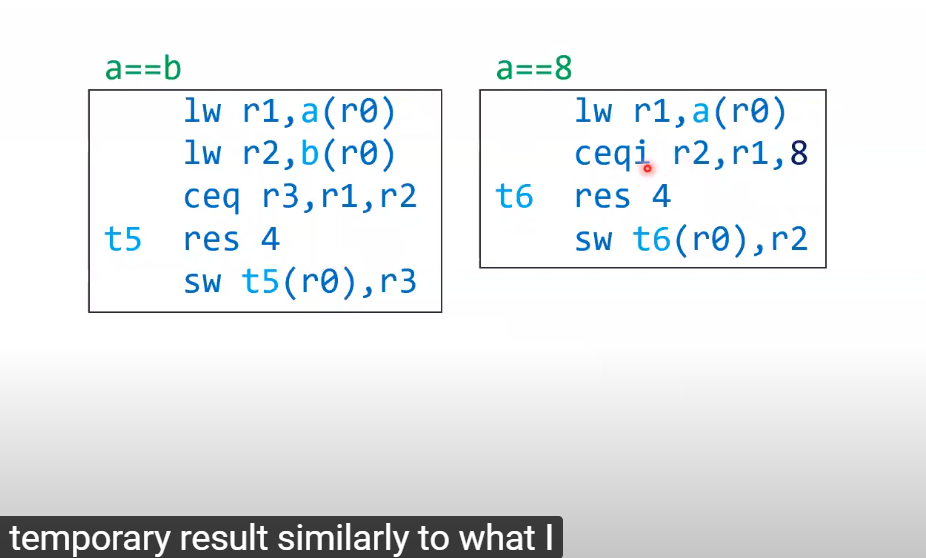
也就是[6]那一部分



还有一种方法是直接使用symboltable，事先算好

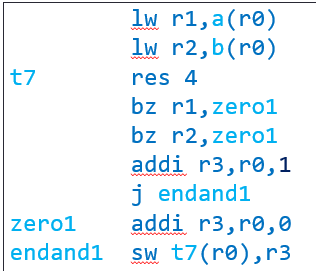


addi可以直接加int，（只对Int有效



ceq，比较两边大小

and逻辑



bz的意思是branch on zero

如果你r1是0（也就是false），那么会跳转到zero1

r3=r0+0=0，也就是false

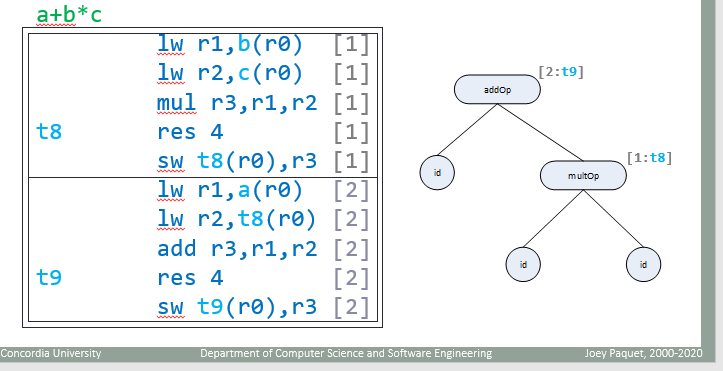
也就是说任意r1r2是zero，就会进行branch zero1，

如果都不是，我们成功进行addi r3,r0,1也就是yes

最后j就是jump,调到最终end,





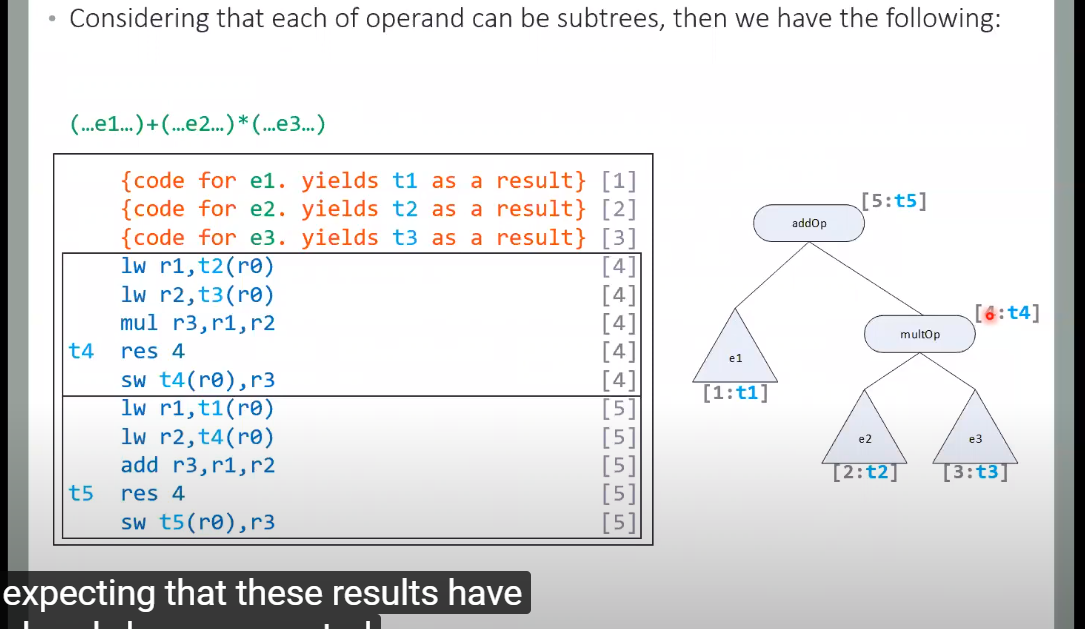


对于复核表示

我们要将结果存在t8里面，

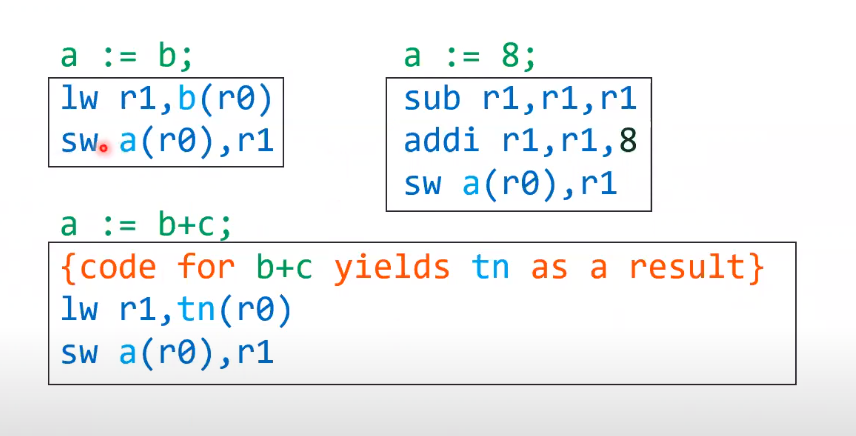
//怎么知道multOp是t8，可以直接给multOp这个点加一个t8

然后再addOp的时候，利用这两个tag，load， 加起来，



总而言之，expr就把结果记录在tx,

然后从下往上一步一步记录



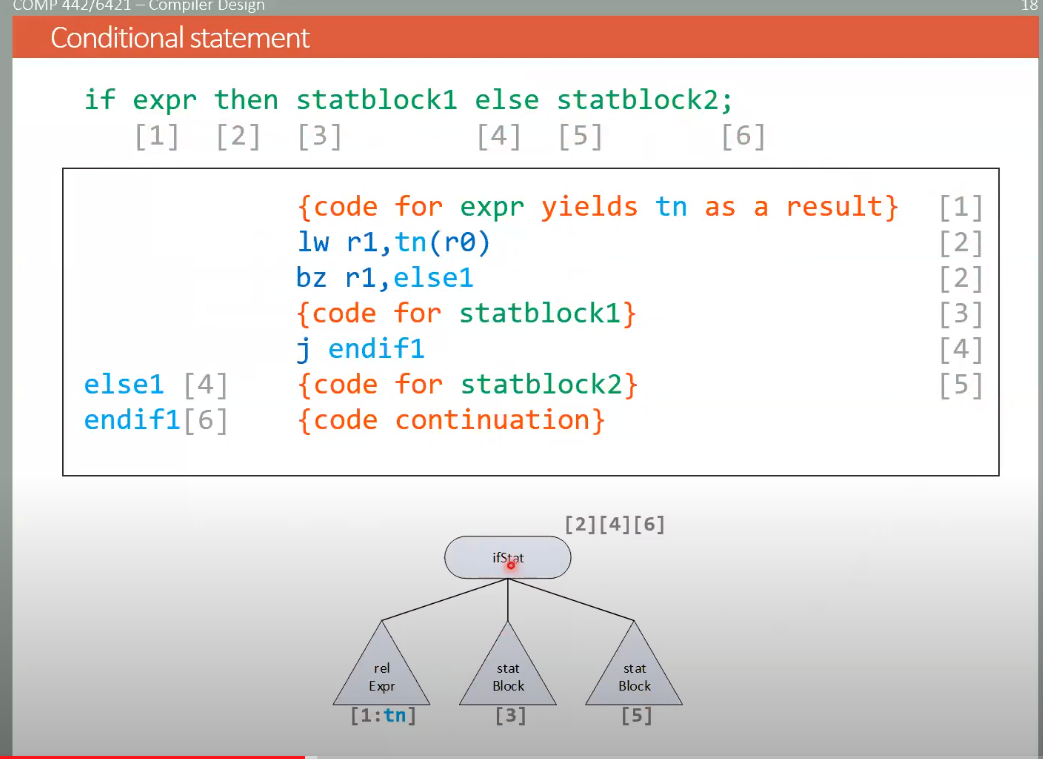
注意r1r2这种东西作为最基础的buffer，并不需要res 4

他们有时候作为暂时存储的buffer

有时候作为offset

而alias ab这种，必须加括号，

等于就很简单



if statement //直接visit if的时候生成整套，因为我们没必要判断if是对是错，都要写

我们也没有 then, else node，

//**注意，这里我们并不能用传统的visit**

**传统的visit会先浏览下面的statBlock，那么格式为**

**xxxexpr**

**xxxxstatBlock**

**xxxstatBlock**

**已经写的东西没法改，**

**你必须手动**

**visit(expr),**

**lw r1,tn**

**br z1,else1**

**stat1**

**j...........**

注意的是哪怕execution, 我们也可以加前缀

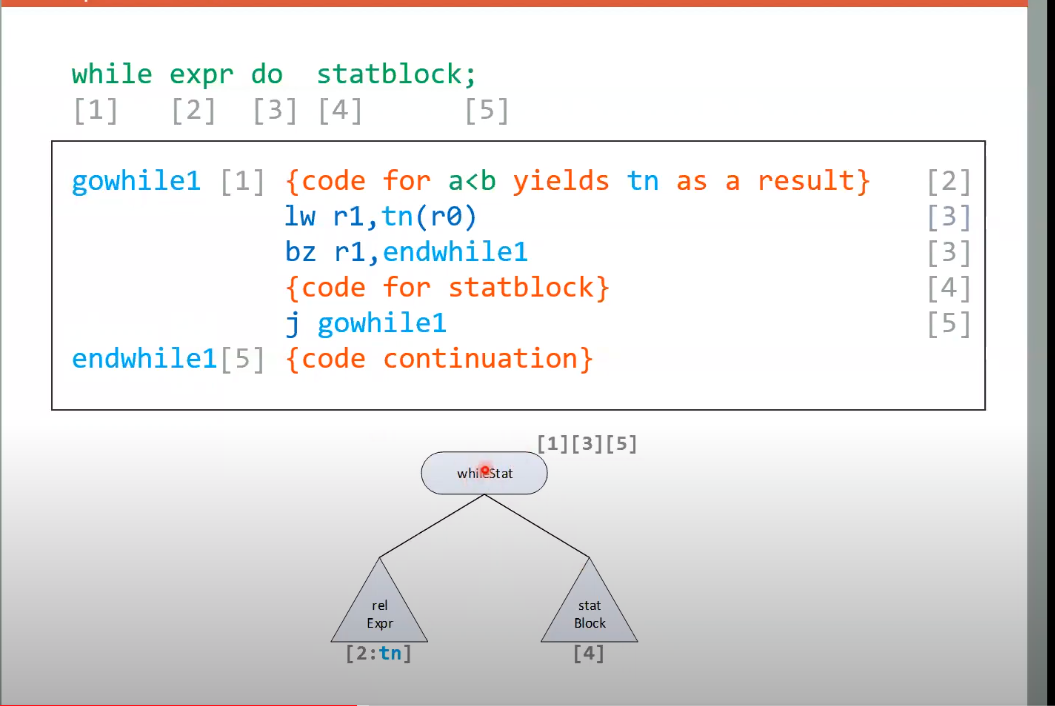
把tn(r0) load到r1，

bz r1是不是0，

是0，进入else1

不是0，继续,

j Jump到endif1



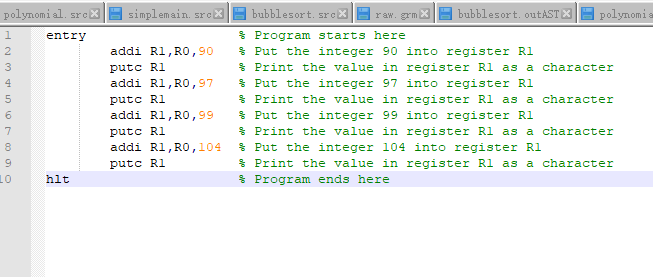
while

首先来一个起始点gowhile1

然后验证a-b,把tn作为result，

问题在于能不能改写tn//我觉得应该可以，tn就是res一个位置而已

虽然他说read write都可以不要，因为很难  
但最好有个write，因为我们需要write来证明我们算出来的结果是对的



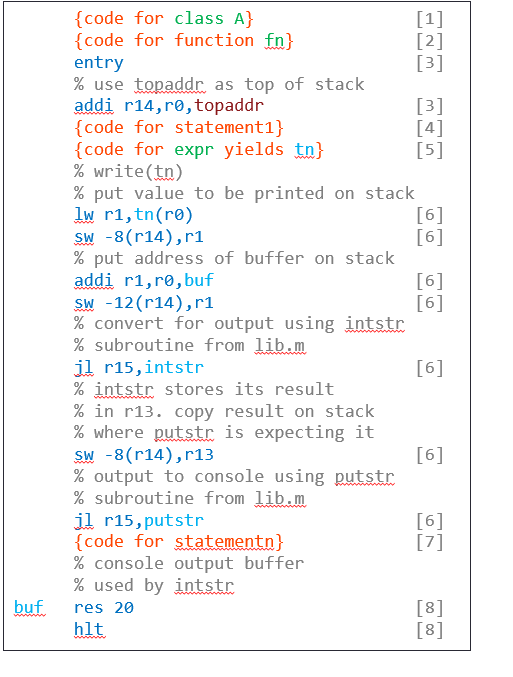
是可以用putc一个一个用char 来Output的

但是

可以使用library



moon.exe simple.m lib.m



我们是实际上是用两个subroutine， 子程序， 就是intstr与putstr

intstr可以吧output从int转换成string  
putstr可以吧str输出到控制台

代码解释：



这个很好理解，我们要把结果tn存储在r1里



我们的subroutine 需要把value存在一个指定address，也就是-8(r14)

r14是我们 function stack的底部



我们把buf 的address load到r1上，

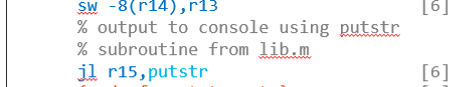
然后把这个address存储到r14 -12

//这样我们intstr两个参数就设置好了buffer(-12(r14) -8（r14），



jl意思是进行Intstr，进行完以后再跳回这一行，r15是固定的

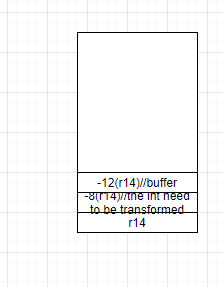
实际上return result会被存在**r13**



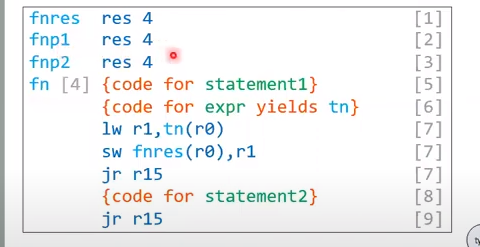
然后我们再把r13的结果存到指定位置-8(r14)

然后就可以putstr（只需要一个parameter）

function stack 的样子



FuncDef



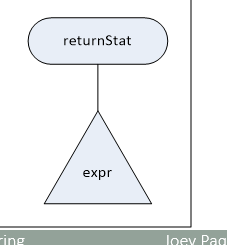
fnres 是return value, 基本上你生成了啥最后会存储在return value

fnp1

fnp2是这个function所需要的parameter

做完这些你就给所有parameter以及return value res了memory

然后在生成第一生成句子之前，你需要generate label fn，用来branch to this function



return stat会生成一个tn

然后你需要load这个tn存储到fnres里

然后在所有一切结束后，还要加第二句jr r15

这个方法不支持recursive，因为第二个function会把fnp1 fnp2覆盖