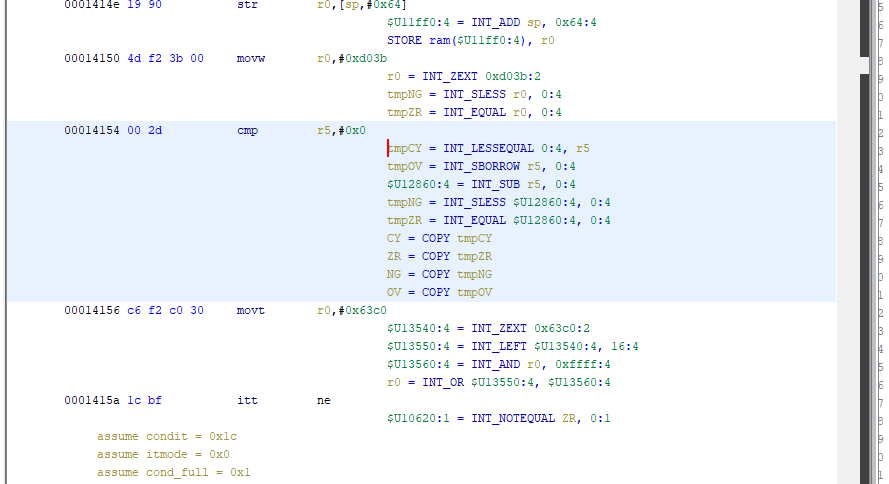
dobc实现文档

# CodeGen

## 代码生成的cpsr干扰问题

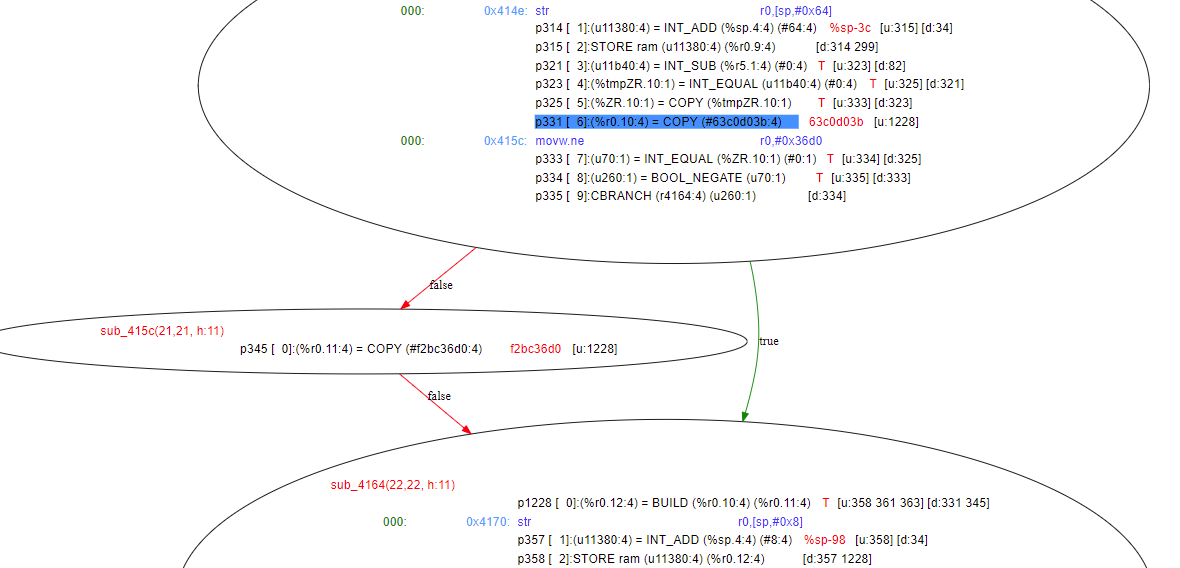
libmakeurl.so

原代码:

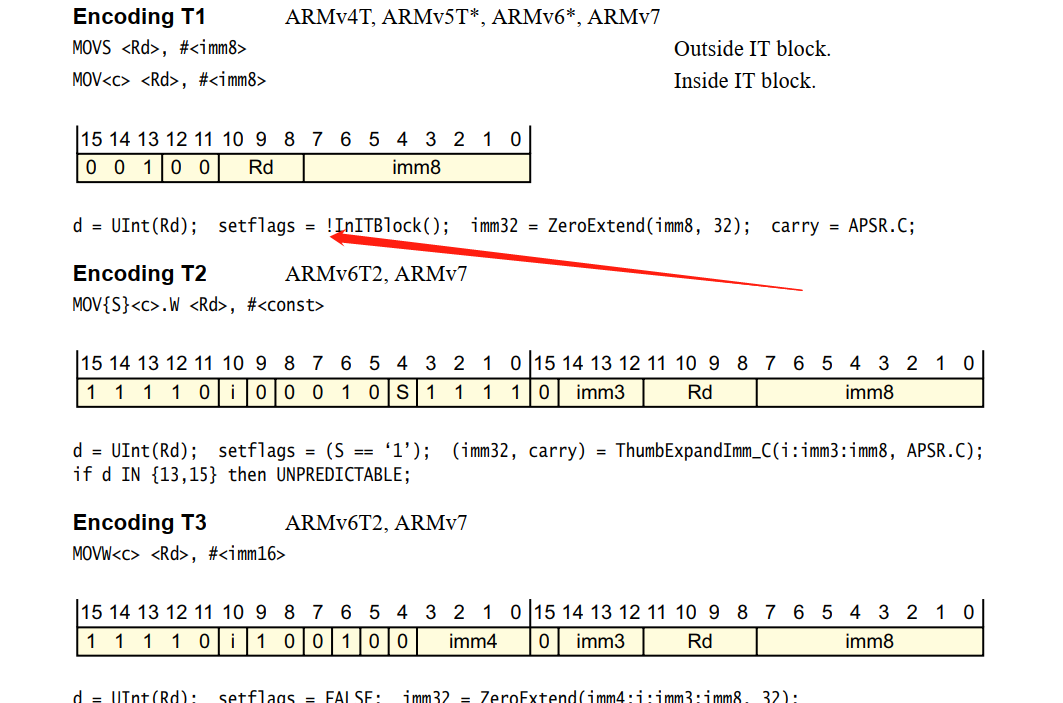


转pcode以后的代码:

看p331，对应上图中 pos:14156处的代码，这个是进行了常量传播以后的。



在进行代码生成时，由于

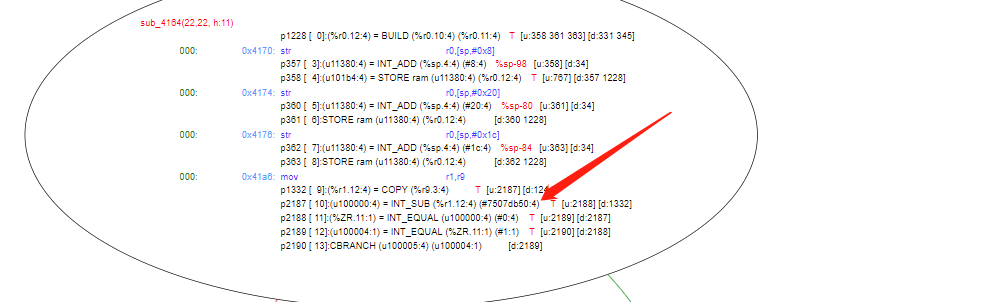


t1指令中的mov会 setflag，所以你生成的代码语义和原始的语义发生了变化。

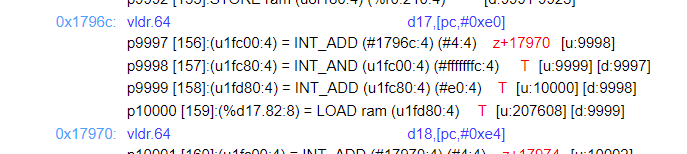
解决方案:

在进行代码生成时，检测有这种特殊的语句，移动到cmp前面去。

## 代码生成时的寄存器分配问题



## simd pc寄存器修复



有部分的simd指令参考了pc相对偏移的值，统统得修复掉

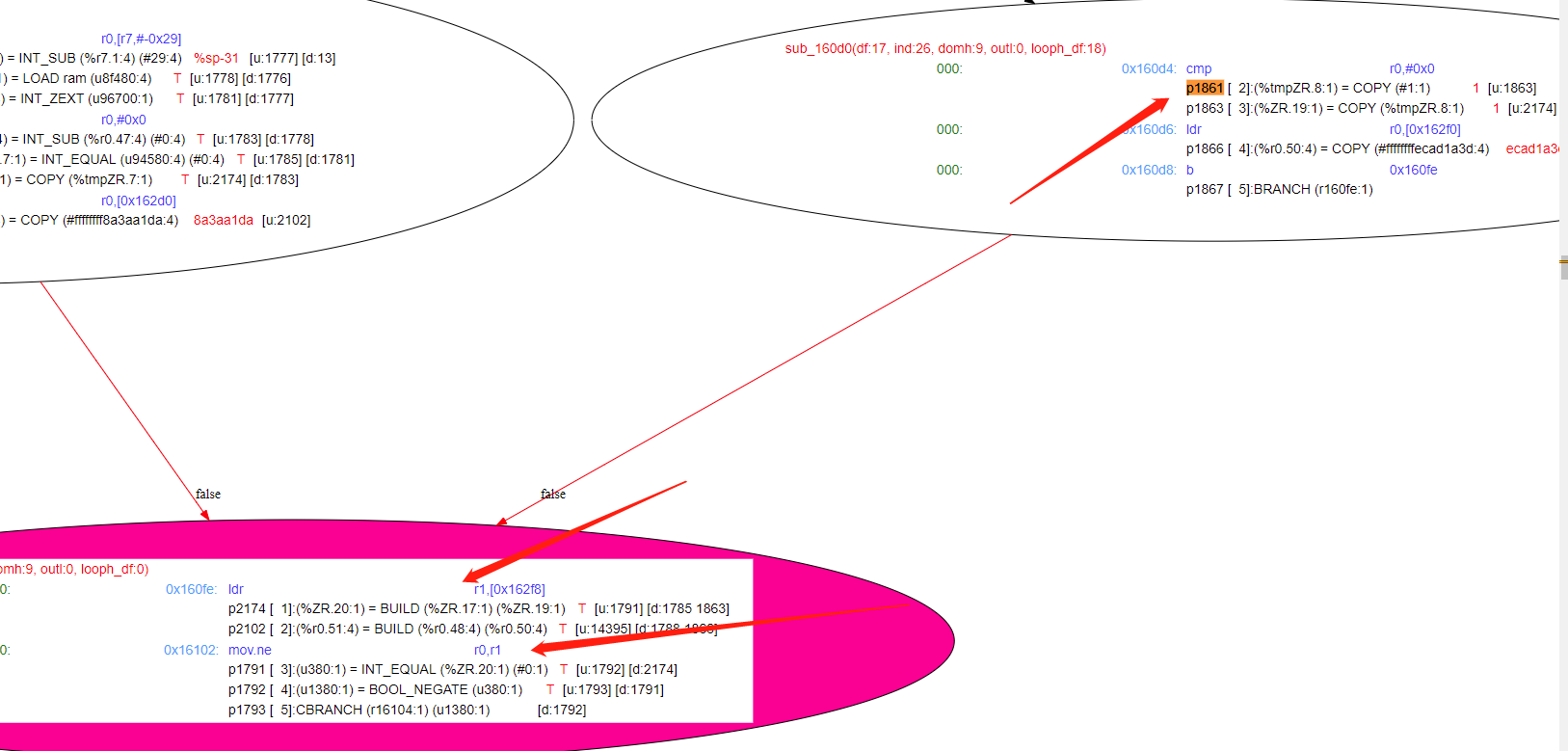
因为simd的ldr搜索的范围过小，所以一般必须得在vld的指令所在的cfg快后插入数据，

## cbz, cbnz关闭

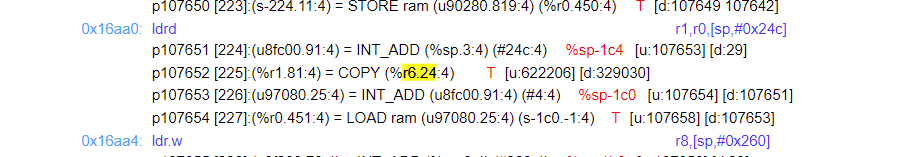
这个cbz和cbna干扰了标准的指令流生成，我们直接把它转换成一般的cmp和bne(eq)指令

|  |
| --- |
| cbz r0, xxx  cmp r0, 0  beq xxx |

## cmp核心指令消失:

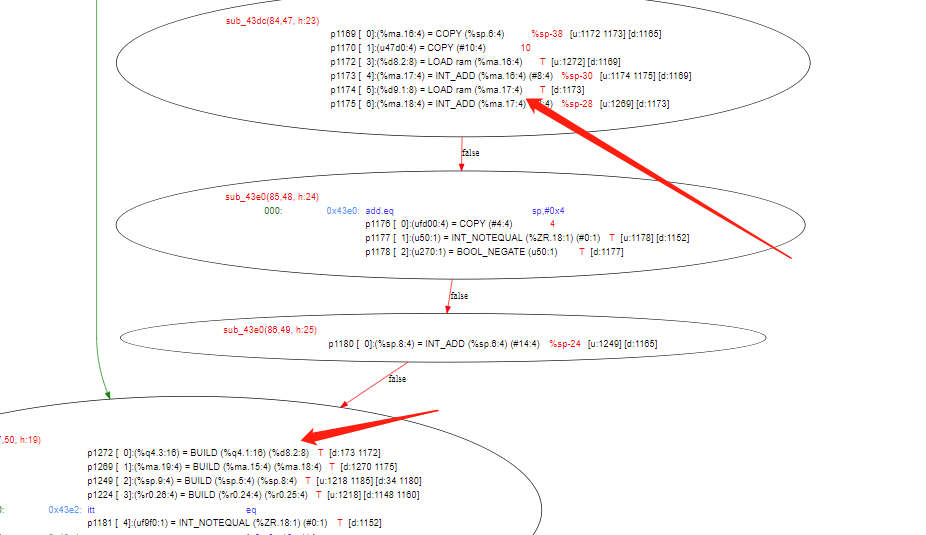


## ldr拷贝传递，导致ldrd结构改变



# SSA:

## reflinement问题



## pattern:别名1

在ollvm\_detect\_framework中，设置的安全区域

# 值分析

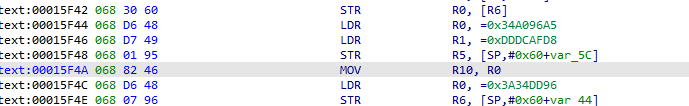
## pc相对常量问题

1000 和 1000可能是不等的

### adr

### ldr(literal)

Sleigh里面有大量语义丢失的行为，导致经常分析中发现缺少了部分信息。



一条ldr指令，直接被转成了copy rn, imm的指令了。这个在代码生成中会导致几个问题：

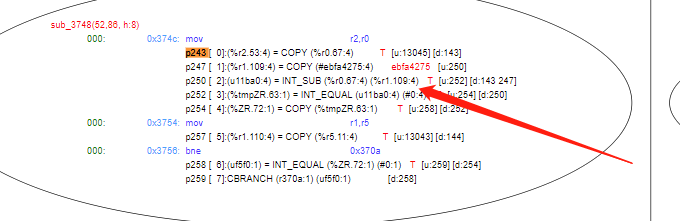
1. 函数本身的大小被强制缩小了，因为你不知道末尾部分是数据
2. 重新生成时，无法引用以前的位置，只能改成mov

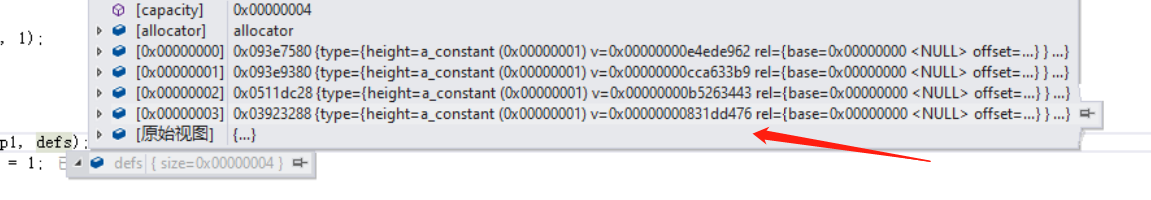
# 语义错误究竟有多严重？

1. ma = sp
2. sp = sp - 16
3. store [ma], d8
4. ..
5. sp = sp -4

拷贝传递以后，5变成 sp = sp – 20

# 值传递无法识别范围



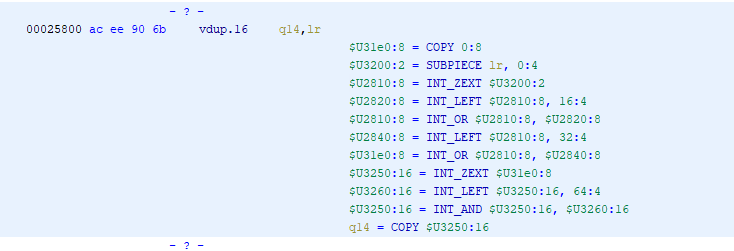


# Ghidra

## vrshr识别错误

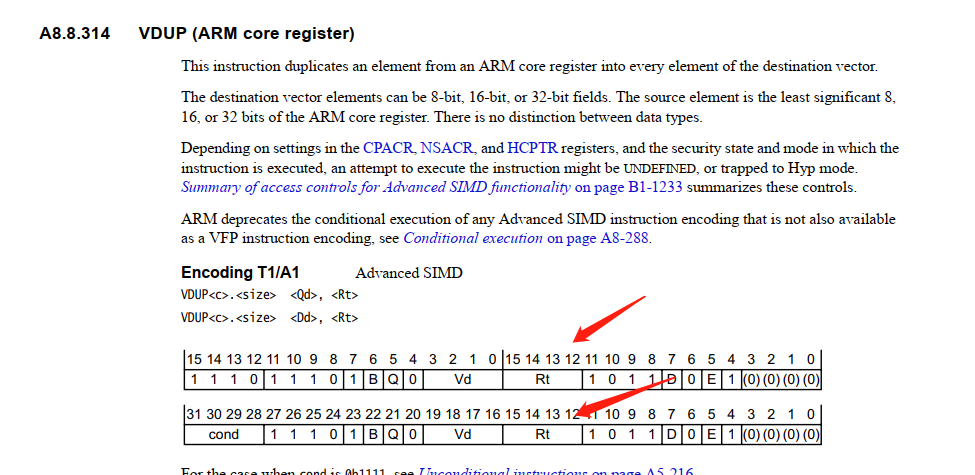
9.2.2已修复

## vdup 反汇编错误



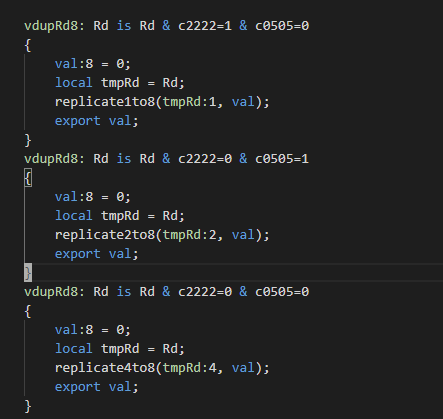
应该是 vdup.32 Q14.R6

### 原因分析:

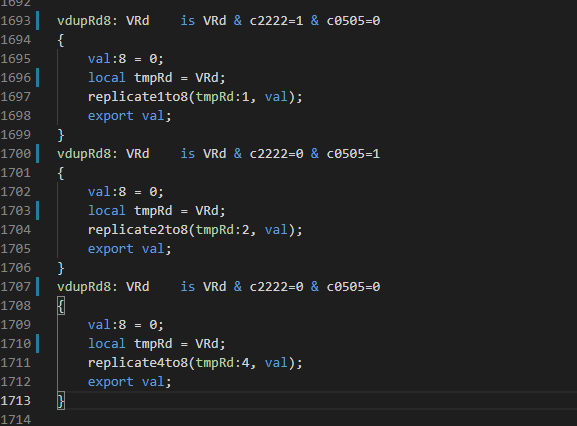


vdup访问的rd,对于arm是 bit[12-15]，对于thumb是[28-31]

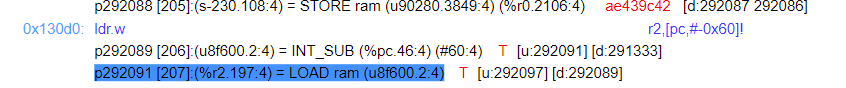
### 解决方案:



使用VRd:

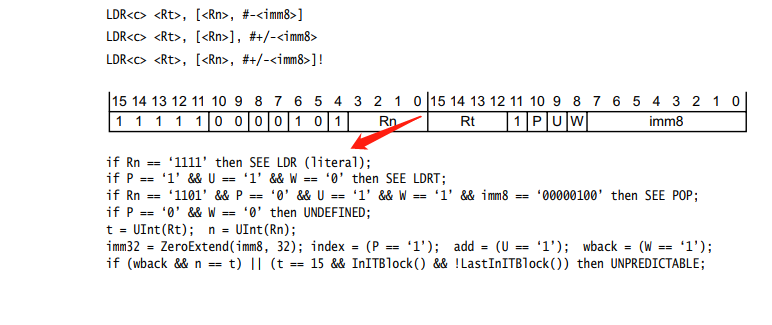


## ldr未识别出pc常数

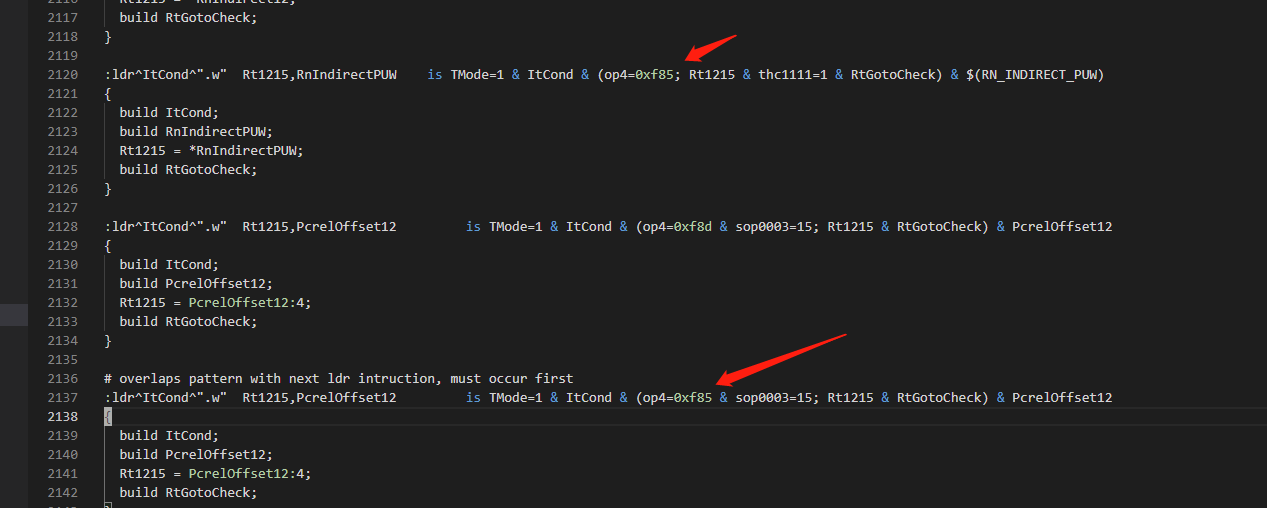




libkwsgmain.so\sub\_cb59

这个是因为白皮书中规定了

当rn=1111时，按照LDR(literal)去解析，



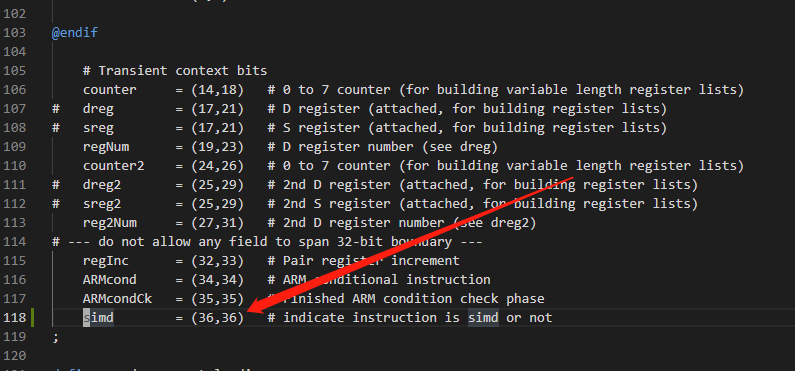
但是，Ghidra的开发人员忘记了这一点，它们还写了一条注释，在第2个箭头上，指明这个f85的ldr必须在第一个，以覆盖当rn=1111的情形。可能多人协同开发时，没有配合好。

### 解决方案:

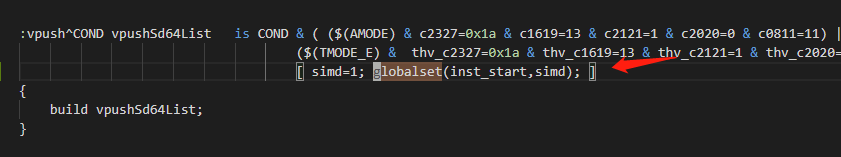
把上面箭头处的第一个移动到第2个后面即可

## simd指令判断

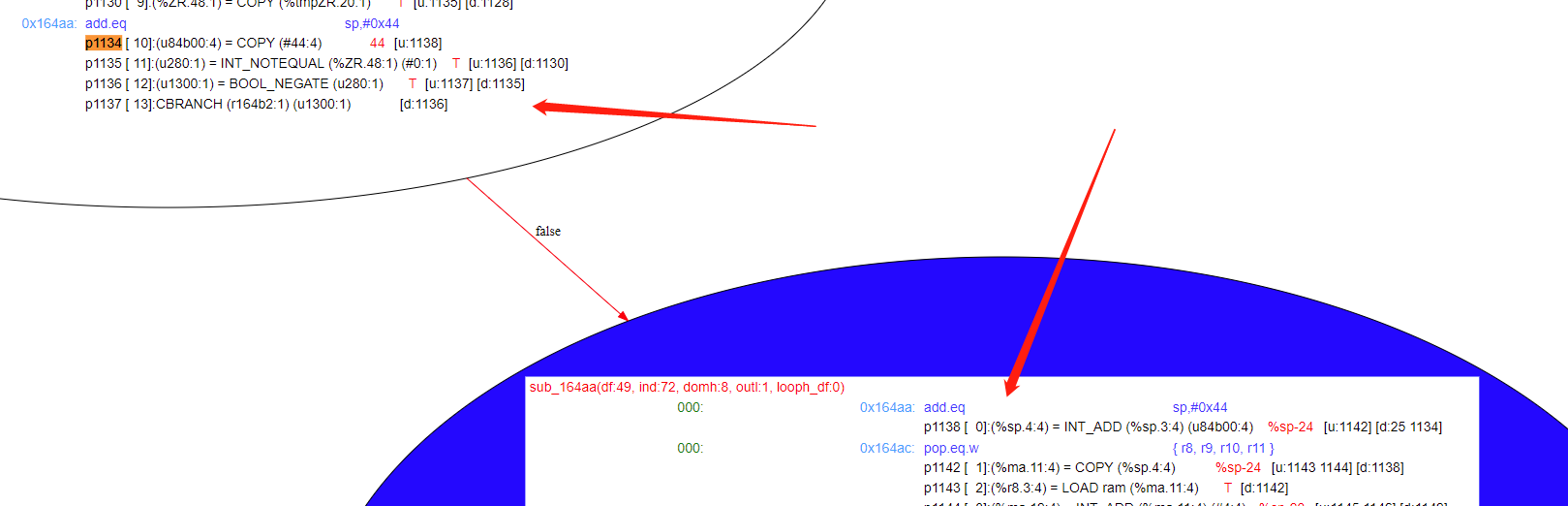
修改arm.sinc中的context，增加一个simd field



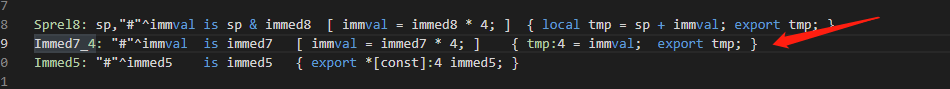
每条指令中分别设置:

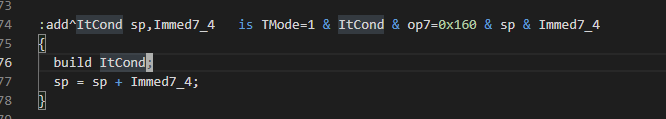


## Ghidra add指令切割，导致uniq变量活跃域跨block



### 原因:

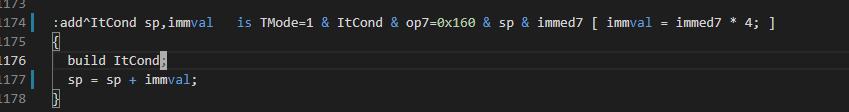




Immed7\_4这个constructor看起来是第一个build，然后它build的时候，把

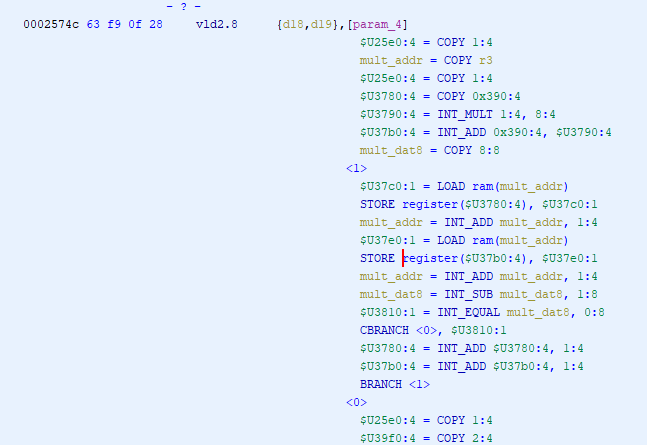
tmp:4 = immval插最前面去了

### 解决方案:



修改了add sp, imm 的sleigh，让imm不再作为默认的第一个constructor

# 常数相对跳转导致的展开问题



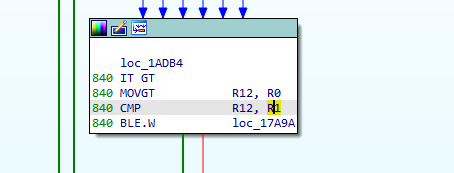
# 对于非规约流图的判断有问题

# ollvm pass

## 不完全it cmp指令比较复写

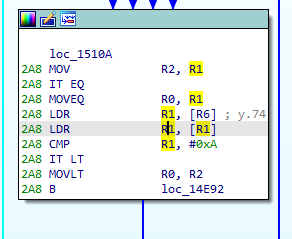
情况1:

正常一个带有cbranch的block，是自带cmp比较节点的，但是有一些情况，比如:



GT的条件是来自于另外的块，

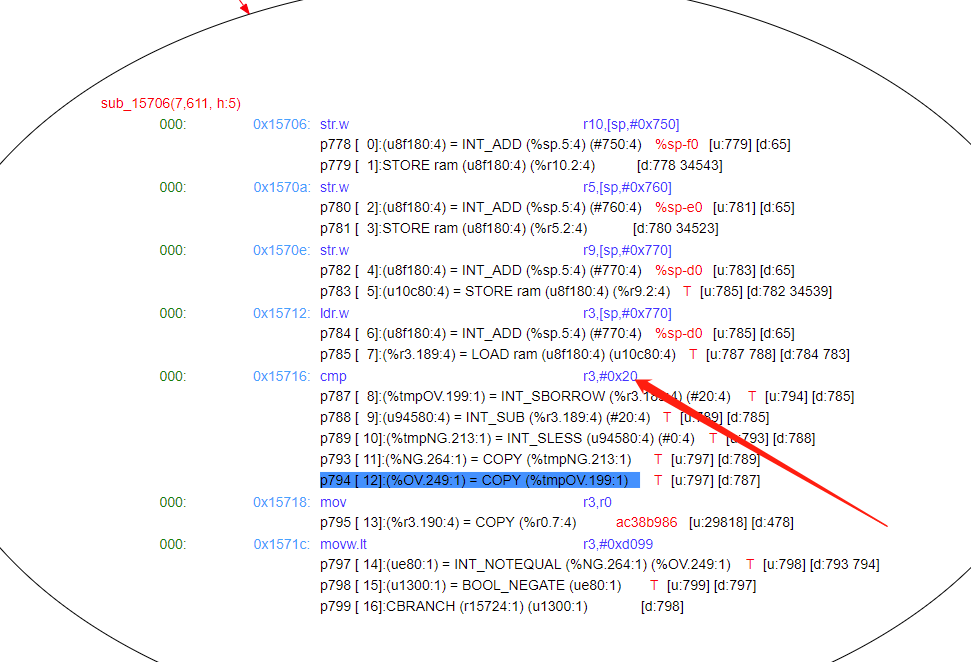
情况2:



来保证范围不溢出。

## 用循环展开ollvm导致普通循环也被展开

普通的循环如



# 堆栈不平衡，情况1:

错误的peephole，导致代码把计算esp的地方合并了，抽取原代码时很容易出错。

# phi的环形依赖，导致死代码无法删除

# 调试

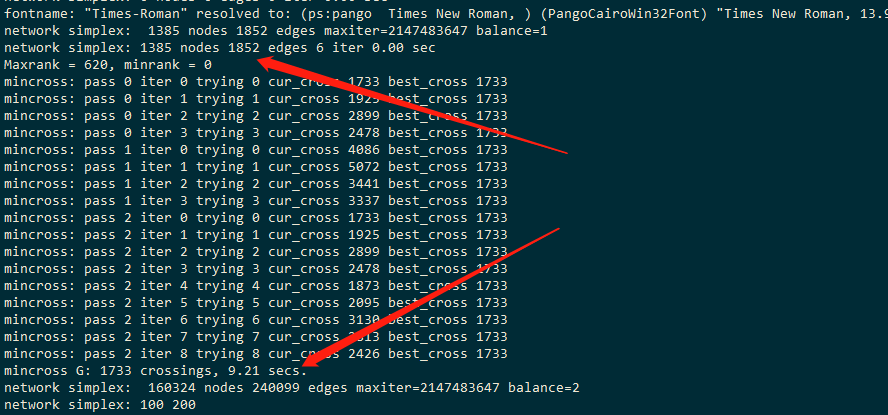
## 超大型cfg流图生成用什么工具？

用dot生成svg，其余的png, jpg都不靠谱。

## dot面对大图，运行卡住？

### 问题分析

dot在生成图时，似乎有2层节点

1. 第一层是 用户直接感受到的 vertex，edge
2. 第一层的节点需要转换成内部的一个 vectex, edge
3. 

上图中，外部的 1385个节点 和 1852 条边，转成内部的边以后有 160324个节点和 240099条边。

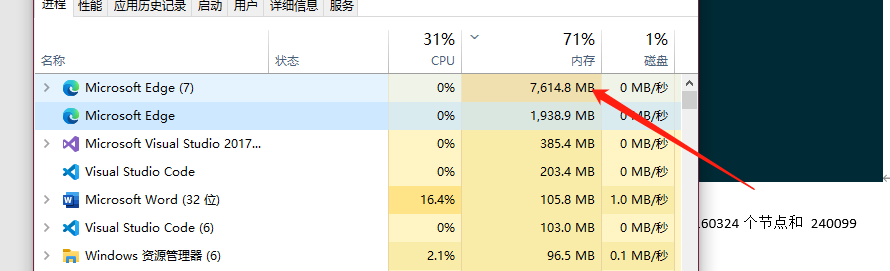
然后那个network simplex就一直卡在那里计算，这些边该如何排

### 解决方案

在.dot 文件中 增加rank = same。

给这些节点排序。人为的尝试安排部分节点的顺序。效果很差

## dot内存消耗过多, edge渲染过慢



# Loader And Exporter

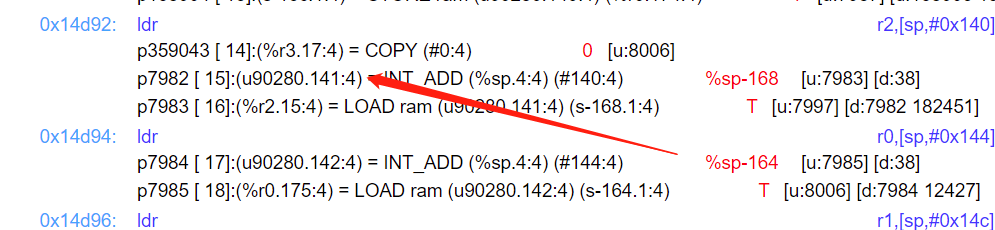
## loadfill抽取问题

在用loadfill来抽取so里的数据时，需要根据pos来判断属于哪个section，然后判断section的flags，假如Shdr里的sh\_flags带有SHF\_WRITE，需要把数据设置为T，不能原文抽取。

# Liveness:

## uniq变量:

uniq变量的活跃性不应该超出一个instruction的范围，比如

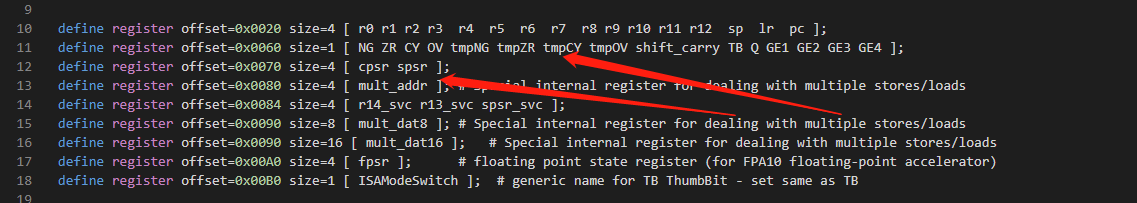


u90280的活跃范围应该从 p7982-p7983，Ghidra的使用和我的不完全一致。

## general-pupose register:

他们是全局的，所以他们需要插入phi节点

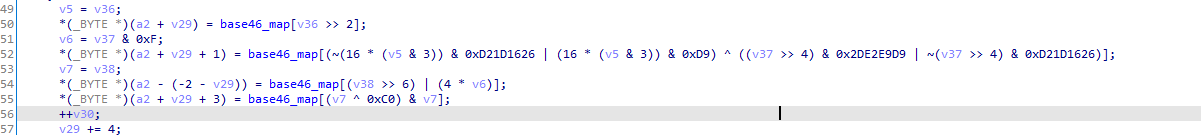
## temporat register:

他们的活跃范围和uniq一致。

# Peephole

简单的peephole我就不说了，我重点讲一下一些复杂的变形

## 变形1



这个变形很麻烦，需要多次变形



变形2:

