# SysYVM LIR/Runtime

## 运行环境

1. 主体结构：

runtime\_stack: 操作栈

reg\_file: 寄存器堆

data\_mem: 数据区（内存）

InstrSet: 指令集

2. 特殊寄存器：

ip寄存器：存放当前的指令地址（在InstrSet中的位置）

stride寄存器：辅助数组计算，存放数组各维度的步幅（类似torch中的stride机制）

ret寄存器：存放函数返回值

3. 其它数据结构

AddrMaps: 记录每个函数在内存中分配的变量及其在data\_mem中的地址

LabelMap: 记录每一个label在InstrSet中的地址，便于进行跳转

## 运算类指令

1. itof: 弹出栈顶的一个数据将其从整数转化为浮点数

2. ftoi: 弹出栈顶的一个数据将其从浮点数转化为整数

3. add / sub / mul: 弹出栈顶的两个数据进行加/减/乘

4. div / div\_f: 弹出栈顶的两个数据进行整数除法/浮点数除法

5. mod: 弹出栈顶的两个数据进行整数的同余运算

6. eq / ne / gt / lt / ge / le: 弹出栈顶的两个数据进行等于/不等于/大于/小于/大于等于/小于等于比较

## 内存/寄存器操作指令

首先定义一些特殊标记：

@x: 表示内存变量x

%rx: 表示第x个寄存器

@%rx: 表示寄存器间接寻址，即data\_mem[%rx]

&@x: 表示内存变量x的地址

1. alloca @x:

在data\_mem中为内存变量x分配一块存储单元，并将地址记录在AddrMaps中

2. with\_space [x1, x2, ...]:

补充说明x是一个[x1][x2][...]的数组，需要补充分配对应数量的空间。该指令一定出现在alloca之后

3. push num/@x/%rx/@%rx/&@x:

将立即数num/变量x/寄存器x/data\_mem[%rx]/变量x的地址推入栈中

4. pop @x/%rx/@%rx:

将栈顶的一个数据弹出，并存入变量x/寄存器x/data\_mem[%rx]

5. mov @x/%rx/@%rx:

与pop类似，不过不弹出栈顶数据，只将其复制到对应位置（mov指令是窥孔优化的产物，pop X; push X可以优化为mov X）

6. printf\_int(printf\_float) num/@x/%rx:

将立即数num/变量x/寄存器x进行输出，\_int表示输出整数，\_float输出浮点数

7. with\_type [x1, x2, ...]:

补充说明数组标识符（其实就是指针）的类型，根据下一条指令（一定为gep或gp）计算出步幅，并存在stride寄存器中

gep指令对应的stride为x2 \* x3 \* ... \* xn，用于一般数组的寻址

举例：对于数组A[x1][x2][x3]以及数组表达式A[y1][y2][y3]，为找到A[y1][y2][y3]的地址，首先执行with\_type [x1, x2, x3]; gep y1;得到地址addr(A) + y1 \* x2 \* x3；然后执行with\_type [x2, x3]; gep y2;得到地址addr(A) + y1 \* x2 \* x3 + y2 \* x3；最后执行with\_type [x3]; gep y3;得到地址addr(A) + y1 \* x2 \* x3 + y2 \* x3 + y3 \* 1；得到A[y1][y2][y3]的实际地址

gp指令对应的stride为x1 \* x2 \* x3 \* ... \* xn，用于作为函数参数的数组的寻址

举例：对于作为函数参数的数组A[][x1][x2]以及数组表达式A[y1][y2][y3]，为找到A[y1][y2][y3]的地址，首先执行with\_type [x1, x2, x3]; gp y1;得到地址addr(A) + y1 \* x1 \* x2；之后再执行相应的gep指令

8. gep / gp x:

参见with\_type中对两种指令的解释

9. init / array\_init X:

在全局变量的alloca语句后使用，将全局变量/数组赋值为X

## 跳转指令

1. label L:

作为跳转的目的地址，L为一个基本块名或一个函数名。LabelMap[L]即为该语句的地址

2. goto / else L:

根据LabelMap跳转到L标记的指令地址，else与goto均表示跳转，只不过表示了不同的分支选择

3. if:

用于处理分支语句，使用格式为：if; goto L1; else L2，通过栈顶元素进行分支判断，为真则执行goto L1，否则执行else L2

## 其它指令

1. push\_ip / push\_ret: 将ip/ret寄存器中的值推入栈中

2. pop\_ip / pop\_ret: 将栈顶值弹出到ip/ret寄存器中

3. push\_all\_reg: 将reg\_file中的值全部推入栈中，即保护现场

4. pop\_all\_reg: 将栈中元素一次弹出到reg\_file中，即恢复现场

5. new\_func: 创建函数的内存空间，具体而言就是在AddrMaps中新增了一个Map并记录下该函数空间的起始地址\_\_MEM\_\_

6. free\_func: 释放函数的内存空间，销毁当前函数的Map（位于AddrMaps的栈顶），并释放从\_\_MEM\_\_开始的的内存空间