编译原理 MiniDecaf 编译器实验
Step 10 实验报告

李祥泽
2018011331
lixiangz18@mails.tsinghua.edu.cn

实验内容

实现数组

数组部分大致按实验指导书的指导完成.

实现指针算术

没有按实验指导书的方法, 利用几条 IR 指令拼接得到指针算术, 而是新加入了涉及指针的 IR 指令 `addip`, `addpi`, `subpi`, `subpp` (`addip` 意为 "**add i**nteger **p**ointer", 其余亦然. add 做区分是为了保持运算数顺序的约定). 这些 IR 指令在生成汇编时将对需要的操作数进行移位, 以得到合适的偏移量.

下标操作利用以上的指针算术指令完成.

思考题

设有以下几个函数, 其中局部变量 `a` 的起始地址都是 `0x1000` (4096), 请分别给出每个函数的返回值 (用一个常量 MiniDecaf 表达式表示, 例如函数 `A` 的返回值是 `*(int*)(4096 + 23 * 4)`)

```
int A() {
    int a[100];
    return a[23];
}

int B() {
    int *p = (int*) 4096;
    return p[23];
}

int C() {
    int a[10][10];
    return a[2][3];
}

int D() {
    int *a[10];
    return a[2][3];
}

int E() {
    int **p = (int**) 4096;
    return p[2][3];
```

```
24  }
```

A

`*(int*)(4096 + 23 * 4)`

B

`*(int*)(4096 + 23 * 4)`

C

`*(int*)(4096 + 2 * 40 + 3 * 4)`

D

`*(int*)(*(int*)(4096 + 2 * 4) + 3 * 4)`

E

`*(int*)(*(int*)(4096 + 2 * 4) + 3 * 4)`

如果我们决定支持一维的可变长度的数组, 而且要求数组仍然保存在栈上, 应该在现有的实现基础上做出那些改动?

在编译期, 统计变长数组的数量, 在进入函数分配栈帧空间时为每个变长数组分配 1 个指针的空间 (这个空间可以用静态的 `frameaddr` 来管理).

每次遇到变长数组的声明, 动态地为它分配栈空间 (需要新增分配栈空间的 IR 指令), 并将其起始地址 (即分配后的 `sp`) 填入预留的空间 (也需要新增对应的 IR 指令). 考虑到 IR 指令和运算栈的性质, 这样做并不会切断运算栈.

在访问变长数组时, 先用 `frameaddr` 在静态栈帧空间查其地址, 然后再以该地址为基址作下标运算或进一步访存.

销毁栈帧时, 不再严格要求 `sp` 指针回到初始位置, 而是直接用 `fp` 覆盖 `sp`. 按分配栈空间时对 `fp` 的使用方法不同, 这将使 `sp` 指向进入函数时的位置 (也即可以直接返回的位置) 或保存了 `fp` 等寄存器之后的位置 (即局部变量开始的位置). 再对 `ra`, `fp` 等做适当的恢复操作即可安全返回上层.

> 这里说 **"不再严格要求 `sp` 指针回到初始位置"**, 是因为实验指导书中的函数 epilogue 采用了 `addi sp,sp,FRAMESIZE` 的方式销毁栈帧, 并且恢复保存的寄存器都是通过 `sp` 寻址的. 这要求 `sp` 严格地回到刚开始分配栈帧的位置.

Honor Code

主要参考实验指导书实现. 参考了助教的示例代码.