CS 241 Week 11 Tutorial

Generating Procedure Calls and Optimization

Spring 2019

1 Procedure Calls

In class, we have introduced one way of saving and restoring registers when calling a procedure.

For caller:

push $29
push $31
lis $31
.word <function label>
jalr $31
pop arguments
pop $31
pop $29

For callee:

Function<procedure name>:
sub $29, $30, $4; set frame pointer for callee
reserve space for local variables
push saved registers
code(statements)
code(return expr)
add $30, $29, $4
jr $31

Consider the following questions:

1. Why do we make our frame pointer pointing to the bottom of the stack frame? What needs to be changed if we want to use the convention of frame pointer pointing to the top of the stack frame?

2. Describe how to reduce the size of the generated code in the case of multiple consecutive function calls, such as:

```c
int f(){
    ....
g();
g();
g();
    ....
}
```

Note: assume that there is a rule for this kind of statement:

```c
statement -> ID LPAREN arglist RPAREN SEMI
```

3. Suppose we would like to support function overloading:

```c
// The following would be legal
int f(int *x, int y){
    ...
}
```
```c
int f(int x, int y){
    ...
}
```

What do you need to change in all the stages of the WLP4 compiler in order to support this?

# 2 Optimization

## 2.1 Constant Folding, Constant Propagation and Dead Code Elimination

For the following C++ functions, re-write the program to fold and propagate all constants where applicable. Also remove all code that will definitely not execute.

```c
int f1(int x, int y){
    if (x + y <= INT_MAX){
        return y;
    } else {
        return x;
    }
    return -1;
}
```

```c
int f2(int x){
    int i;
    int c;
    for (i = 0; i < 10; i++){
        c += i * 2;
    }
    return c + x;
}
```

```c
int f3(){
    int k = 5;
    int i = 0;
    int j = 1;
    int &rj = j;
    int &ri = i;
    while(k != 0){
        int t = rj;
        rj = ri + rj;
        ri = t;
    }
    return j;
    return i;
}
```

## 2.2 Common Subexpression Elimination

Can we perform common subexpression elimination on any subexpression? If so, explain why. If not, provide a counterexample (in C++ or Racket).