Code Generation

Compiler Construction

2024-12-17

Prof. Dr. Michael Kuhn
michael.kuhn@ovgu.de

Parallel Computing and I/O
Institute for Intelligent Cooperating Systems
Faculty of Computer Science
Otto von Guericke University Magdeburg
https://parcio.ovgu.de

Outline

- Code generation deals with transforming some IR into assembly
  - Code can also be generated from the AST, DAG etc.
- Generated code may not necessarily be fast
  - Optimizations can improve performance significantly

Outline

Group Exercise

- Generate code for the following expressions.

```
foo: function void () = {
    i: integer;       // r10d
    j: integer = 0;   // r11d
    k: integer = 23; // r12d

    i = 42;
    j = i + j;
    k = i * j;
}
```

Group Exercise

- Generate code for the following expressions.

```
1  foo:
2      ...
3      MOVL $0, %r11d
4      MOVL $23, %r12d
5
6      MOVL $42, %r10d
7
8      ADDL %r10d, %r11d
9
10     MOVL %r10d, %eax
11     IMULL %r11d
12     MOVL %eax, %r12d
```

Group Exercise

- Generate code for the following code using the register calling convention.

```
1 foo: function integer (a: string, b: integer) = {
2     return b;
3 }
4 main: function integer () = {
5     foo("foo", 42);
6     return 1;
7 }
```

Group Exercise

- Generate code for the following code using the register calling convention.

```
1  foo:
2      PUSHQ %rbp
3      MOVQ %rsp, %rbp
4      SUBQ $12, %rsp
5      MOVQ %rdi, -8(%rbp)
6      MOVL %esi, -12(%rbp)
7      MOVL -12(%rbp), %eax
8      POPQ %rbp
9      RET
10 .str_foo:
11     .string "foo"
12 main:
13     PUSHQ %rbp
14     MOVQ %rsp, %rbp
15     MOVQ $.str_foo, %rdi
16     MOVL $42, %esi
17     CALL foo
18     MOVL $1, %eax
19     POPQ %rbp
20     RET
```

- Why is it necessary to generate the values for all function arguments before moving them into the argument registers? Come up with an example where calling a function breaks if we do not do this.

- Why is it necessary to generate the values for all function arguments before moving them into the argument registers? Come up with an example where calling a function breaks if we do not do this.

```
1 foo: function integer (a: string, b: integer, c: integer, d: integer) = {
2     return b;
3 }
4 main: function integer () = {
5     foo("foo", 42, 42, 42 * 23);
6     return 1;
7 }
```

Group Exercise

- Why is it necessary to generate the values for all function arguments before moving them into the argument registers? Come up with an example where calling a function breaks if we do not do this.

```
.str_foo:
    .string "foo"
main:
    PUSHQ %rbp
    MOVQ %rsp, %rbp
    MOVQ $.str_foo, %rdi
    MOVL $42, %esi
    MOVL $42, %edx
    MOVL $42, %ecx
    MOVL $23, %eax
    IMULL %ecx // clobbers %edx
    MOVL %eax, %ecx
    CALL foo
    MOVL $1, %eax
    POPQ %rbp
    RET
```

- What are the advantages and disadvantages of caller-saved (r10, r11) and callee-saved (rbx, rbp, rsp, r12, r13, r14, r15) registers?

- What are the advantages and disadvantages of caller-saved (r10, r11) and callee-saved (rbx, rbp, rsp, r12, r13, r14, r15) registers?
- Callee-saved registers have to be saved and restored all the time
  - Callees often do not know which registers the caller is using

- What are the advantages and disadvantages of caller-saved (r10, r11) and callee-saved (rbx, rbp, rsp, r12, r13, r14, r15) registers?
- Callee-saved registers have to be saved and restored all the time
  - Callees often do not know which registers the caller is using
- Caller-saved registers can be skipped if not required anymore

Group Exercise

- Can a global variable declaration have a non-constant initializing expression?

Group Exercise

- Can a global variable declaration have a non-constant initializing expression?

```
1  .data
2  x:
3      .quad 42
4  y:
5      .quad 42 * 23
6  z:
7      .quad 42 * x
```

Group Exercise

- Can a global variable declaration have a non-constant initializing expression?

```
1  .data
2  x:
3      .quad 42
4  y:
5      .quad 42 * 23
6  z:
7      .quad 42 * x
```

- The data section can only contain constants

- Can a global variable declaration have a non-constant initializing expression?

```
1 .data
2 x:
3     .quad 42
4 y:
5     .quad 42 * 23
6 z:
7     .quad 42 * x
```

- The data section can only contain constants
- Constant expressions can be evaluated to a constant value

- Come up with an expression that exhausts the available scratch registers (rbx, r10, r11, r12, r13, r14, r15).

Outline

- Code generation is the final necessary piece of our compiler
  - It allows actually running our translated applications
- Code generation is important for correctness and performance
  - There are multiple ways to achieve the same effect
- Optimizations allow reducing the number of instructions
  - Optimizations are typically performance before assembly is generated

References

[Thain, 2020] Thain, D. (2020). ***Introduction to Compilers and Language Design: Second Edition.*** http://compilerbook.org/.