Compiler Construction: Assignment 2

Fabian Krause

November 22, 2021

Assignment 2: Compiler for $\mathcal{L}_{Int}$

5 passes:

1. Remove Complex Operands: $\mathcal{L}_{Int} \rightsquigarrow \mathcal{L}_{Int}^{mon}$
2. Select Instructions: $\mathcal{L}_{Int}^{mon} \rightsquigarrow x86_{Var}$
3. Assign Homes: $x86_{Var} \rightsquigarrow x86_{Int}$
4. Patch Instructions: $x86_{Int} \rightsquigarrow x86_{Int}$
5. Prelude and Conclusion: $x86_{Int} \rightsquigarrow x86_{Int}$

1. Remove Complex Operands: $\mathcal{L}_{Int} \rightsquigarrow \mathcal{L}_{Int}^{mon}$

Goal: All operands must be atomic (for example, $3 + (3 + 4)$ is not allowed)
Example: `BinOp(e1, op, e2)`

- `e1` and `e2` have to be atomic, since `addq` and `subq` (and any other n-ary operator) require two (n) atomic operands
- If `BinOp(e1, op, e2)` is supposed to be atomic, assign it to fresh variable `tmp`
  - `BinOp(e1, op, e2)` $\rightsquigarrow$ `tmp`, $[\ldots, (\texttt{tmp}, \texttt{BinOp}(\texttt{atm1}, \texttt{op}, \texttt{atm2}))]$

Not all expressions have to be made atomic!

```
1          3 + 4
2
```

```
1          x = 1 + 6
2
```

2. Select Instructions: $\mathcal{L}_{Int}^{mon} \rightsquigarrow x86_{Var}$

Example: Statement `var = BinOp(atm1, Sub(), atm2)`
Idea:

```
1         movq   atm1, var
2         subq   atm2, var
3
```

What if `atm1 = var`?

```
1         movq   atm1, atm1
2         subq   atm2, atm1
3
```

What if `atm2 = var`?

```
1         movq   atm1, atm2
2         subq   atm2, atm2
3
```

Example: Statement `var = BinOp(atm1, Sub(), atm2)`

If `atm1 = var`

```
        subq   atm2, atm1

```

If `atm2 = var`

```
        negq   atm2
        addq   atm1, atm2

```

Else

```
        movq   atm1, var
        subq   atm2, var

```

3. Assign Homes: $x86_{Var} \rightsquigarrow x86_{Int}$

Spill everything!
Store assignments in a mapping from `Variable` to `arg`. Create new stack locations on the fly.

4. Patch Instructions: $x86_{Int} \rightsquigarrow x86_{Int}$

x86 only allows one operand to be a memory location!

```
1         istr   -16(%rbp), -32(%rbp)
2
```

```
1         movq   -16(%rbp), %rax
2         istr   %rax, -32(%rbp)
3
```

5. Prelude and Conclusion: $x86_{Int} \rightsquigarrow x86_{Int}$

Prelude/Prologue:

- Save old base pointer `%rbp` (callee-saved register!)
- Update base pointer
- Make space for variables

```
1         pushq   %rbp
2         movq    %rsp, %rbp
3         subq    $16, %rsp
4
```

Conclusion/Epilogue:

- Remove space for variables
- Restore old base pointer

```
1         addq    $16, %rsp
2         popq    %rbp
3
```

Questions?