Recitation 5: Code Generation*

* Low-level IR

Yotam Feldman

Based on materials by Yannis Smaragdakis and slides by Guy Golan-Gueta
Code Generation

Abstract Syntax Tree (AST) → Intermediate code generation → LLVM code → Target code generation → x86 code

Ex. 2

Today: compiling basic, imperative features
Code Generation

- Valid programs (ASTs) compile to an LLVM program that’s
  - valid,
  - executes,
  - has the same input-output and external behavior (console output)
- Rules for valid MiniJava ASTs: https://www.cs.tau.ac.il/research/yotam.feldman/courses/wcc20/semantic.html
LLVM Recap

• Typed
• Unbounded number of SSA registers
• Stack allocation `alloca`
• Heap allocation and `bitcast`
• `load` and `store`
• Branch and conditional branch: `br`
• Array and `getelementptr`
• Basic binary operations: `add`, `sub`...
• Function calls: `call` and `ret`
Translation (IR Lowering)

Visitor(s) generate LLVM declarations and code
• Class declarations
• Statements
• Expressions
Local variables

• Local variables translated to stack locations
• Load & store
  – Too early to optimize!
Store & Load According to Static Type

- Assume type safety!
  - Otherwise, the behavior is **undefined**
- Use symbol table to obtain the type from the declaration
Simple Expressions

- $\text{TR}[e] = \text{LIR translation of AST expression } e$
  - A sequence of IR instructions
  - Use temporary variables (IR registers) to store intermediate values during translation
Compound Expressions

• SSA, need to allocate fresh registers
• Order of evaluation is important
  – Think about method calls that perform mutations
Compound Expressions: Example

\[ TR[x + y] \]

\[
\begin{align*}
  \texttt{%}_0 &= \text{load i32, i32*}\ \texttt{%x} \\
  \texttt{%}_1 &= \text{load i32, i32*}\ \texttt{%y} \\
  \texttt{%}_2 &= \text{add i32}\ \texttt{%}_0,\ \texttt{%}_1
\end{align*}
\]
Translating expressions – example

\[ \text{TR}[x + 42] \]

\[
\begin{align*}
&\text{IdExpr} \quad \text{id} = x \\
&\text{AddExpr} \\
&\text{IntLitExpr} \quad \text{num} = 42
\end{align*}
\]

\[
\begin{align*}
&_0 &= \text{load i32, i32* } %x \\
&_1 &= \text{i32 42 --- invalid} \\
&_2 &= \text{add i32 } %_0, 42
\end{align*}
\]
Translating Statement Blocks

\[ \text{TR}\{s_1; s_2; \ldots ; s_N\}\]
Translating If-Then-Else

- Conditional branch
- Need to generate code evaluating the condition
Translating Short-Circuit And

Generate code for

• Evaluating the first operand
• If true, continuing; otherwise skipping
• Evaluating the second operand
• Joining using the \texttt{phi} instruction
Translating While

- Jump back to beginning of the loop
- Exercise 😊
Arrays

- Allocation
- Access
- Assignment
- Dynamic checks
  - “ArrayIndexOutOfBoundsException”

Also: array length (exercise 😊)
Summary

• Local (stack) variables
• Generating code for expressions
• Control structures
• Short-circuit and
• Arrays
• Upcoming: object-oriented code generation
Exercise #2

• **Start early**
• Read the requirements carefully!
• **Reference compilation examples**
• Extend symbol table and class hierarchy analysis from ex1
• Assume that the program is semantically valid
  – **List of rules**
  – For the type in LLVM instructions, use the declared type and assume that the usage is valid
• Class fields and (instance) method calls – next week
  – From today: local variables, expressions, control flow, arrays...
• **Submission instructions**