Big step up from assembler – higher level notations

- need familiar library of object code
- machine language
- compiler
- assembler
- machine
- code
- processor

Abstract view of computer

Course Overview

Lab 2.2: \textit{programming in java (2nd edition)} by Andrew Appel

\textbf{Pre-requisite:} CSCI 320

\textbf{Textbook is:} \texttt{http://www.cs.mcgill.ca/class/spring2006/csci430/}

\textbf{Office Hours:} The Rm. 3-414, 1400 V.G. Morrison

\textbf{Important Notes:}

- No collaborative coding (code sharing allowed)
- \texttt{compilers}
- \texttt{programmer}
- \texttt{course}
- \texttt{library}
- \texttt{implementations (examples in the library)

- 20% midterm, 30% final, 50% programming projects
Parser generates much of the work
企图外词构造
produce meaningful error messages
convert error messages
Guide context-sensitive analyses
recognize context-free syntax

Parser

Which of front end construction can be accomplished
shape the code for the back end
preliminary source map
produce it
locate errors

Responsibilities:

Front end

lex

Back end is NP-complete

Example: 0^2_1 (n in log n)

Implementation

Scanner
Modern optimizers are usually built as a set of passes.

- **CSE:** Encode common subexpressions.
- **Inliner:** Inlines small functions.
- **Loop Un roll:** Unrolls loops.
- **Strength Reduction:** Performs arithmetic optimizations.
- **Register Allocation:** Allocates registers for variables.
- **Code Motion:** Moves code around to optimize memory access.
- **Dead Code Elimination:** Removes unused code.
- **Constant Folding:** Evaluates constants at compile time.
- **Loop Unroll:** Unrolls loops.
- **Loop Vectorization:** Converts loops to vector instructions.
- **Register Allocation:** Allocates registers for variables.
- **Loop Invariant Code Motion:** Moves code around to optimize memory access.
- **Dead Code Elimination:** Removes unused code.
- **Constant Folding:** Evaluates constants at compile time.
- **Loop Unroll:** Unrolls loops.
- **Loop Vectorization:** Converts loops to vector instructions.
- **Register Allocation:** Allocates registers for variables.
- **Loop Invariant Code Motion:** Moves code around to optimize memory access.
- **Dead Code Elimination:** Removes unused code.
- **Constant Folding:** Evaluates constants at compile time.
- **Loop Unroll:** Unrolls loops.
- **Loop Vectorization:** Converts loops to vector instructions.
- **Register Allocation:** Allocates registers for variables.
- **Loop Invariant Code Motion:** Moves code around to optimize memory access.
- **Dead Code Elimination:** Removes unused code.
- **Constant Folding:** Evaluates constants at compile time.
- **Loop Unroll:** Unrolls loops.
- **Loop Vectorization:** Converts loops to vector instructions.
- **Register Allocation:** Allocates registers for variables.
- **Loop Invariant Code Motion:** Moves code around to optimize memory access.
- **Dead Code Elimination:** Removes unused code.
- **Constant Folding:** Evaluates constants at compile time.
- **Loop Unroll:** Unrolls loops.
- **Loop Vectorization:** Converts loops to vector instructions.
- **Register Allocation:** Allocates registers for variables.
- **Loop Invariant Code Motion:** Moves code around to optimize memory access.
- **Dead Code Elimination:** Removes unused code.
- **Constant Folding:** Evaluates constants at compile time.