CS 301: Compiler and Runtime System Design

What is a compiler?
What is a runtime system?
A compiler is a program translator.

(high-level) source language code

Compiler

(low-level) target language code
A compiler is a program translator.

```c
int expr(int n) {
    int d = 4 * n * n * (n + 1) * (n + 1);
    return d;
}
```

```assembly
lda $30,-32($30)
stq $26,0($30)
stq $15,8($30)
bis $30,$30,$15
bis $16,$16,$1
stl $1,16($15)
lds $f1,16($15)
sts $f1,24($15)
ldl $5,24($15)
bis $5,$5,$2
s4addq $2,0,$3
ldl $4,16($15)
mull $4,$3,$2
ldl $3,16($15)
addq $3,1,$4
mull $2,$4,$2
ldl $3,16($15)
addq $3,1,$4
mull $2,$4,$2
ldl $0,20($15)
br $31,$33
$33:
bis $15,$15,$30
ldq $26,0($30)
ldq $15,8($30)
addq $30,32,$30
addq $30,32,$30
addq $5,30,1
ret $31,($26),1
```
A compiler is a program improver.

<table>
<thead>
<tr>
<th>Unoptimized Code</th>
<th>Optimized Code</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>lda $30,-32($30)</code></td>
<td><code>s4addq $16,0,$0</code></td>
</tr>
<tr>
<td><code>stq $26,0($30)</code></td>
<td><code>mul $16,$0,$0</code></td>
</tr>
<tr>
<td><code>stq $15,8($30)</code></td>
<td><code>addq $16,1,$16</code></td>
</tr>
<tr>
<td><code>bis $30,$30,$15</code></td>
<td><code>mul $0,$16,$0</code></td>
</tr>
<tr>
<td><code>bis $16,$16,$1</code></td>
<td><code>mul $0,$16,$0</code></td>
</tr>
<tr>
<td><code>stl $1,16($15)</code></td>
<td><code>ret $31,($26),1</code></td>
</tr>
<tr>
<td><code>lds $f1,16($15)</code></td>
<td></td>
</tr>
<tr>
<td><code>sts $f1,24($15)</code></td>
<td></td>
</tr>
<tr>
<td><code>ldl $5,24($15)</code></td>
<td></td>
</tr>
<tr>
<td><code>bis $5,$5,$2</code></td>
<td></td>
</tr>
<tr>
<td><code>s4addq $2,0,$3</code></td>
<td></td>
</tr>
<tr>
<td><code>ldl $4,16($15)</code></td>
<td></td>
</tr>
<tr>
<td><code>mul $4,$3,$2</code></td>
<td></td>
</tr>
<tr>
<td><code>ldl $3,16($15)</code></td>
<td></td>
</tr>
<tr>
<td><code>addq $3,1,$4</code></td>
<td></td>
</tr>
<tr>
<td><code>mul $2,$4,$2</code></td>
<td></td>
</tr>
<tr>
<td><code>ldl $3,16($15)</code></td>
<td></td>
</tr>
<tr>
<td><code>addq $3,1,$4</code></td>
<td></td>
</tr>
<tr>
<td><code>mul $2,$4,$2</code></td>
<td></td>
</tr>
<tr>
<td><code>stl $2,20($15)</code></td>
<td></td>
</tr>
<tr>
<td><code>ldl $0,20($15)</code></td>
<td></td>
</tr>
<tr>
<td><code>br $31,$33</code></td>
<td></td>
</tr>
</tbody>
</table>

$33:$

<table>
<thead>
<tr>
<th>Unoptimized Code</th>
<th>Optimized Code</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bis $15,$15,$30</code></td>
<td><code>bis $15,$15,$30</code></td>
</tr>
<tr>
<td><code>ldq $26,0($30)</code></td>
<td><code>ldq $26,0($30)</code></td>
</tr>
<tr>
<td><code>ldq $15,8($30)</code></td>
<td><code>ldq $15,8($30)</code></td>
</tr>
<tr>
<td><code>addq $30,32,$30</code></td>
<td><code>addq $30,32,$30</code></td>
</tr>
<tr>
<td><code>ret $31,($26),1</code></td>
<td><code>ret $31,($26),1</code></td>
</tr>
</tbody>
</table>
A compiler is a program checker.
A compiler is a programmer's assistant.

Static program analysis

Type checking

Scope checking

Control flow checking

Instrumentation for dynamic program analysis
A runtime system is a compiler's co-conspirator.

(high-level) source language code

Compiler

(low-level) target language code

Runtime System
A runtime system is
{ a program translator, a program improver, a program checker,
a programmer's assistant, a compiler's co-conspirator }

Memory management +
Garbage Collection (GC)

Feedback-directed optimization

Profiling + profile guided optimization (PGO)

Dynamic program analysis

Just in Time compilation (JIT)

Resource management
Simplified Compiler Structure

Source code
if (b == 0) a = b;

Understand source code

Intermediate code

Optimize

Intermediate code

Generate target code

cmp $0,%ecx
cmovz %edx,%ecx

Front end (target-independent)

Optimizer

Back end (target-dependent)
Compiler Front End

Source code (character stream)

if (b == 0) a = b;

Token stream

if ( b == 0 ) a = b ;

Abstract syntax tree (AST)

if

b

0

a

b

Decorated AST

boolean

if

int

b

int

0

int

a

int

b

Lexical Analysis
(Lexing)

Syntax Analysis
(Parsing)

Semantic Analysis
(Name Resolution, Type Checking)
Compiler Middle / Back End

Decorated AST

Intermediate code

Target code

boolean ==
int 0
int a
int b

if

...  

Intermediate Code Generation

Optimizations

Target Optimizations and Code Generation

t = (b == 0)
jumpunless t, L
a = b
label L

t = (b == 0)
jumpunless t, L
a = 0
label L

cmp $0, ecx
cmovz edx, ecx
Zooming out: compiling to machine code...

Compiler

Lexical Analysis
Syntax Analysis
Semantic Analysis
Optimization
Code Generation

Assembler

Assembly code
Object code (machine code)
Fully-resolved object code (machine code)
Executable image
Course Material

- Theoretical Foundations
- Implementation
- Synthesis
  - Programming Languages (251), Theory (235), Algorithms (231), Computer Systems (240), Software Construction

- Semester project: full compiler
  - for small statically typed language
  - mostly from scratch
  - implemented in Scala
  - other tools: git, IntelliJ, gcc, gdb, ...

- Tutorial

Project Groups

- groups of 3 (or 2)
- you choose, I can help
- work with new people on the Tiny compiler this week
- Collaboration and Honor Code
Tutorial Meetings

I form trios with your schedule/people input, end of this week.

Weekly assignment for 1-hour small-group meeting with me.

**Preparation:** Read, work on all problems.

**Participation:** Discuss any of the problems in detail.
- work through solution on board
- extend problem in new directions
- explain where you got stuck and why

**Reviews:** Write clean solutions to a few questions I choose.
- in LaTeX
- due 24 hours after your meeting
- better prep before meeting => fewer written solutions after
Books

Compilers
Principles, Techniques, & Tools
Second Edition
Alfred V. Aho
Monica S. Lam
Ravi Sethi
Jeffrey D. Ullman

Engineering a Compiler
Second Edition
Keith D. Cooper & Linda Torczon
Weekly Schedule

Tutorial Day: Monday/Tuesday/Wednesday
- tutorial meetings
- next tutorial assignment posted
- Tuesday scheduled class time not required after today
  - May be used for tutorial
  - Otherwise: guaranteed project group availability, code review, etc.

Lab/Work Day: Friday scheduled class time (+ drop-in hours after)
- Introduce and start new project stages/checkpoints
- Project work time
- Code reviews
- Occasional mini-lectures framing upcoming tutorial material
- Project checkpoints / due dates

Other Days:
- Code reviews?
- Drop-in hours, appointments
Tiny compiler!
Source code (character stream)

```c
if (b == 0) a = b;
```

**Lexical Analysis**

**Token stream**

```c
if ( b == 0 ) a = b;
```

**Identifiers:**
- x  
- y11  
- elsen  
- _i00

**Integers:**
- 2  
- 1000  
- 5L

**Floating point:**
- 2.0  
- .02  
- 1.  
- 1e5  
- 0.e-10

**Strings:**
- "x"  
- "She said, \"Hey!\""

**Comments:**
- /* don't change this */  
- // or this

**Keywords:**
- if  
- else  
- while  
- break

**Symbols:**
- +  
- *  
- {  
- }  
- =  
- <  
- <<  
- ==  
- [  
- ]  
- >=
Regular Expressions (regex, RE)

A language is a set of words: \{ SCI, KSC \}, \{ a,b,c,d,... \}

A regular expression describes a (regular) language

\[
abab \ a|b \ (a|b)^* \ [1-9][0-9]^* \ [a-z][a-z0-9]^*
\]

Regular expression primitives:

- **a**: ordinary character stands for itself
- **ε**: the empty string (epsilon)
- **R|S**: either R or S (alternation), where R,S are REs
- **RS**: R followed by S (concatenation)
- **R\***: R repeated 0 or more times (Kleene star)

\( L(R) = \) the language defined by regular expression R

\[
L(a \ (SCI \mid KSC)) = \{ a\text{SCI}, a\text{KSC} \}
\]

\[
L([1-9][0-9]*) = \{ 1,2,3,4,5,6,7,8,9,10,11,12,13,....\}
\]
If R is a regular expressions, so are:

- \( R? \) = \( \varepsilon | R \) (zero or one R)
- \( R+ \) = \( RR^* \) (one or more R’s)
- \( (R) \) = R (no effect: grouping)

- \([abc]\) = a|b|c (any of the listed characters)
- \([a-e]\) = a|b|…|e (character ranges)
- \([^ab]\) = c|d|… (any character except the listed characters)
- . = a|b|c|d|… (any character except newline)

- \( \backslash . \) = a literal . character
- \( \backslash * \) = a literal * character
- \( \backslash \backslash \) = a literal \ character

...
Acceptors:
(a.k.a. recognizers)

Abstract machines that determine if an input string belongs to a language, answering Yes/No.

Finite Automata:
acceptors for languages described by regular expressions
Finite Automata

Regular Expression: \((-|\epsilon)[0-9][0-9]^*\)

Nondeterministic Finite Automaton:

Deterministic Finite Automaton:
Building an acceptor for a regular expression:

1. Regular Expression $R$
2. $R \Rightarrow \text{NFA Conversion}$
3. $\text{NFA} \Rightarrow \text{DFA Conversion}$
4. $w \Rightarrow \text{DFA Simulation}$
5. $w \in L(R)$: Yes
6. $w \notin L(R)$: No
Example Lexer Generator Specification

%%

digits = 0|[1-9][0-9]*
letter = [A-Za-z]
identifier = {letter}({letter}|[0-9_])*
whitespace = [\ \t\n\r]+
%

{whitespace} {/* discard */}
{digits} { return new Token(INT, parseInt(yytext())); } 
"if" { return new Token(IF, yytext()); } 
"while" { return new Token(WHILE, yytext()); }
...
{identifier} { return new Token(ID, yytext()); }