Lecture 1: Introduction

Intro. to Computer Language Engineering
Course Administration info.
Outline

• Course Administration Information

• Introduction to computer language engineering
  – What are compilers?
  – Why should we learn about them?
  – Anatomy of a compiler
Course Administration

• Staff
• Text
• Course Outline
• The Project
• Project Groups
• Grading
Staff

• Lecturers
  – Prof. Saman Amarasinghe
  – Prof. Martin Rinard
Optional Textbooks

**Tiger Book**
- Modern Compiler Implementation in Java
  by A.W. Appel
  Cambridge University Press, 1998

**Whale Book**
- Advanced Compiler Design and Implementation
  by Steven Muchnick
  Morgan Kaufman Publishers, 1997

**Dragon Book**
- Compilers -- Principles, Techniques and Tools
  by Aho, Sethi and Ullman
  Addison-Wesley, 1988
The Five Segments

1. Lexical and Syntax Analysis
2. Semantic Analysis
3. Code Generation
4. Data-flow Optimizations
5. Instruction Optimizations
Each Segment...

• Segment Start
  – Project Description
• Lectures
  – 2 to 4 lectures
• Project Time
• (Project checkpoint)
• Project Time
• Project Due
Project Groups

• Each project group consists of 3 to 4 students

• Grading
  – All group members (mostly) get the same grade
Weekly Group Meeting with the TA

• TA will schedule a weekly 30 minute slot
• The group will get to:
  – Ask questions about the project
  – Discuss design decisions
  – Describe what each person is doing
  – Answer questions the TA has about the project
• TA will use this to measure individual contribution to the project ➔ be there!
Grades

- Compiler project: 58%
- Paper Discussion: 12% (3% each)
- In-class Quizzes: 30% (10% each)
Grades for the Project

- Scanner/Parser 10%
- Semantic Checking 10%
- Code Generation 14%
- Data-flow Opt. 12%
- Instruction Opt. 12%

58%
The Quiz

• Three Quizzes

• In-Class Quiz
  – 50 Minutes (be on time!)
  – Open book, open notes
Paper Discussions

• Will be giving out 3 papers
• Read the paper, think about…
  – Do you like the research?
  – What is the context of the research?
  – Did anything surprise you about the paper?
  – Do the results allow you to evaluate the proposed technique?
  – Do the authors understand the wider ramifications of their research?
  – How have things changed since the paper was written?
  – How important will the research be in the future?
• Write a 150 to 200 word summary of the paper.
• Will schedule a one-on-one (15 or 20 minute) meeting with a professor or a TA to talk about the paper.
Outline

• Course Administration Information

• Introduction to computer language engineering
  – What are compilers?
  – Why should we learn about them?
  – Anatomy of a compiler
What is Computer Language Engineering

1. How to give instructions to a computer

2. How to make the computer carry out the instructions efficiently
Power of a Language

• Can use to describe any action
  – Not tied to a “context”

• Many ways to describe the same action
  – Flexible
How to instruct a computer

• How about natural languages?
  – English??
  - “Open the pod bay doors, Hal.”
  - “I am sorry Dave, I am afraid I cannot do that”
  – We are not there yet!!

• Natural Languages:
  – Same expression describes many possible actions
  – Ambiguous
How to instruct a computer

• How about natural languages?
  – English??
  – “Open the pod bay doors, Hal.”
  – “I am sorry Dave, I am afraid I cannot do that”
  – We are not there yet!!

• Use a programming language
  – Examples: Java, C, C++, Pascal, BASIC, Scheme
Programming Languages

• Properties
  – need to be precise
  – need to be concise
  – need to be expressive
  – need to be at a high-level (lot of abstractions)
High-level Abstract Description to Low-level Implementation Details

President

General

Sergeant

Foot Soldier

My poll ratings are low, let's invade a small nation

Cross the river and take defensive positions

Forward march, turn left. Stop!, Shoot

Figures by MIT OCW.
1. How to instruct the computer

- Write a program using a programming language
  - High-level Abstract Description
- Microprocessors talk in assembly language
  - Low-level Implementation Details

Program written in a Programming Languages

Compiler

Assembly Language Translation
1. How to instruct the computer

• Input: High-level programming language
• Output: Low-level assembly instructions

• Compiler has to:
  – Read and understand the program
  – Precisely determine what actions it require
  – Figure-out how to carry-out those actions
  – Instruct the computer to carry out those actions
Example (input program)

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

```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
stl $2,20($15)
ldl $0,20($15)
br $31,$33

$33:
bis $15,$15,$30
ldq $26,0($30)
ldq $15,8($30)
adq $30,32,$30
ret $31,($26),1
```
Efficient Execution of the Actions

Cross the river and take defensive positions

General

Sergeant

Foot Soldier

Figures by MIT OCW.
Efficient Execution of the Actions

General

Cross the river and take defensive positions

Sergeant

Where to cross the river? Use the bridge upstream or surprise the enemy by crossing downstream?
How do I minimize the casualties??

Foot Soldier

Figures by MIT OCW.
Efficient Execution of the Actions

President: My poll ratings are low, let's invade a small nation.

Russia or Bermuda? Or just stall for his poll numbers to go up?

General

Figures by MIT OCW.
Efficient Execution of the Actions

• Mapping from High to Low
  – Simple mapping of a program to assembly language produces inefficient execution
  – Higher the level of abstraction ⇒ more inefficiency
• If not efficient
  – High-level abstractions are useless
• Need to:
  – provide a high level abstraction
  – with performance of giving low-level instructions
Example

```c
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
        x = x + (4*a/b)*i + (i+1)*(i+1);
        x = x + b*y;
    }
    return x;
}
```
test:  
    subu $fp, 16  
    sw zero, 0($fp)  # x = 0  
    sw zero, 4($fp)  # y = 0  
    sw zero, 8($fp)  # i = 0  
lab1:  # for(i=0;i<N; i++)  
    mul $t0, $a0, 4  # a*4  
    div $t1, $t0, $a1  # a*4/b  
    lw $t2, 8($fp)  # i  
    mul $t3, $t1, $t2  # a*4/b*i  
    lw $t4, 8($fp)  # i  
    addiu$t4, $t4, 1  # i+1  
    lw $t5, 8($fp)  # i  
    addiu$t5, $t5, 1  # i+1  
    mul $t6, $t4, $t5  # (i+1)*(i+1)  
    add $t7, $t3, $t6  # a*4/b*i + (i+1)*(i+1)  
    lw $t8, 0($fp)  # x  
    add $t8, $t7, $t8  # x = x + a*4/b*i + (i+1)*(i+1)  
    sw $t8, 0($fp)  
    lw $t0, 4($fp)  # y  
    mul $t1, $t0, $a1  # b*y  
    lw $t2, 0($fp)  # x  
    add $t2, $t2, $t1  # x = x + b*y  
    sw $t2, 0($fp)  
    lw $t0, 8($fp)  # i  
    addui $t0, $t0, 1  # i+1  
    sw $t0, 8($fp)  
    ble $t0, $a3, lab1  
    lw $v0, 0($fp)  
    addu $fp, 16  
    b $ra
Lets Optimize...

```c
int sumcalc(int a, int b, int N) {
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
        x = x + (4*a/b)*i + (i+1)*(i+1);
        x = x + b*y;
    }
    return x;
}
```
Constant Propagation

```c
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
        x = x + (4*a/b)*i + (i+1)*(i+1);
        x = x + b*y;
    }
    return x;
}
```
Constant Propagation

```c
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
        x = x + (4*a/b)*i + (i+1)*(i+1);
        x = x + b*y;
    }
    return x;
}
```
Constant Propagation

```c
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
        x = x + (4*a/b)*i + (i+1)*(i+1);
    }
    return x;
}
```
Constant Propagation

```c
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
        x = x + (4*a/b)*i + (i+1)*(i+1);
        x = x + b*y;
    }
    return x;
}
```
Constant Propagation

```c
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
        x = x + (4*a/b)*i + (i+1)*(i+1);
        x = x + b*0;
    }
    return x;
}
```
Algebraic Simplification

```c
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
        x = x + (4*a/b)*i + (i+1)*(i+1);
        x = x + b*0;
    }
    return x;
}
```
int sumcalc(int a, int b, int N) {
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
        x = x + (4*a/b)*i + (i+1)*(i+1);
        x = x + b*0;
    }
    return x;
}
Algebraic Simplification

```c
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
        x = x + (4*a/b)*i + (i+1)*(i+1);
        x = x + 0;
    }
    return x;
}
```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
        x = x + (4*a/b)*i + (i+1)*(i+1);
        x = x + 0;
    }
    return x;
}
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
        x = x + (4*a/b)*i + (i+1)*(i+1);
        x = x + 0;
    }
    return x;
}
int sumcalc(int a, int b, int N) {
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
        x = x + (4*a/b)*i + (i+1)*(i+1);
        x = x;
    }
    return x;
}
Copy Propagation

```c
int sumcalc(int a, int b, int N) {
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
        x = x + (4*a/b)*i + (i+1)*(i+1);
        x = x;
    }
    return x;
}
```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
        x = x + (4*a/b)*i + (i+1)*(i+1);
    }
    return x;
}
Common Subexpression Elimination

```c
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
        x = x + (4*a/b)*i + (i+1)*(i+1);
    }
    return x;
}
```
Common Subexpression Elimination

```c
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
        x = x + (4*a/b)*i + (i+1)*(i+1);
    }
    return x;
}
```
int sumcalc(int a, int b, int N) 
{
    int i;
    int x, y, t;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
        t = i+1;
        x = x + (4*a/b)*i + (i+1)*(i+1);
    }
    return x;
}
Common Subexpression Elimination

```c
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y, t;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
        t = i+1;
        x = x + (4*a/b)*i + t*t;
    }
    return x;
}
```
Dead Code Elimination

```c
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y, t;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
        t = i+1;
        x = x + (4*a/b)*i + t*t;
    }
    return x;
}
```
Dead Code Elimination

```c
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y, t;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
        t = i+1;
        x = x + (4*a/b)*i + t*t;
    }
    return x;
}
```
Dead Code Elimination

```c
int sumcalc(int a, int b, int N) {
    int i;
    int x, y, t;
    x = 0;

    for(i = 0; i <= N; i++) {
        t = i + 1;
        x = x + (4*a/b)*i + t*t;
    }
    return x;
}
```
Dead Code Elimination

```c
int sumcalc(int a, int b, int N)
{
    int i;
    int x, t;
    x = 0;

    for(i = 0; i <= N; i++) {
        t = i+1;
        x = x + (4*a/b)*i + t*t;
    }
    return x;
}
```
Loop Invariant Removal

```c
int sumcalc(int a, int b, int N) {
    int i;
    int x, t;
    x = 0;

    for(i = 0; i <= N; i++) {
        t = i+1;
        x = x + (4*a/b)*i + t*t;
    }

    return x;
}
```
Loop Invariant Removal

```c
int sumcalc(int a, int b, int N) {
    int i;
    int x, t;
    x = 0;

    for(i = 0; i <= N; i++) {
        t = i+1;
        x = x + (4*a/b)*i + t*t;
    }
    return x;
}
```
Loop Invariant Removal

```c
int sumcalc(int a, int b, int N) {
    int i;
    int x, t, u;
    x = 0;
    u = (4*a/b);
    for(i = 0; i <= N; i++) {
        t = i+1;
        x = x + u*i + t*t;
    }
    return x;
}
```
**Strength Reduction**

```c
int sumcalc(int a, int b, int N)
{
    int i;
    int x, t, u;
    x = 0;
    u = (4*a/b);
    for(i = 0; i <= N; i++) {
        t = i+1;
        x = x + u*i + t*t;
    }
    return x;
}
```
Strength Reduction

```c
int sumcalc(int a, int b, int N)
{
    int i;
    int x, t, u;
    x = 0;
    u = (4*a/b);
    for(i = 0; i <= N; i++) {
        t = i+1;
        x = x + u*i + t*t;
    }
    return x;
}
```

```plaintext
u*0, v=0,
u*1, v=v+u,
u*2, v=v+u,
u*3, v=v+u,
u*4, v=v+u,
...
...
```
int sumcalc(int a, int b, int N) {
    int i;
    int x, t, u, v;
    x = 0;
    u = (4*a/b);
    v = 0;
    for(i = 0; i <= N; i++) {
        t = i+1;
        x = x + u*i + t*t;
        v = v + u;
    }
    return x;
}
Strength Reduction

int sumcalc(int a, int b, int N)
{
    int i;
    int x, t, u, v;
    x = 0;
    u = (4*a/b);
    v = 0;
    for(i = 0; i <= N; i++) {
        t = i+1;
        x = x + v + t*t;
        v = v + u;
    }
    return x;
}
int sumcalc(int a, int b, int N) {
    int i;
    int x, t, u, v;
    x = 0;
    u = (4*a/b);
    v = 0;
    for(i = 0; i <= N; i++) {
        t = i+1;
        x = x + v + t*t;
        v = v + u;
    }
    return x;
}
Strength Reduction

```c
int sumcalc(int a, int b, int N)
{
    int i;
    int x, t, u, v;
    x = 0;
    u = (4*a/b);
    v = 0;
    for(i = 0; i <= N; i++) {
        t = i+1;
        x = x + v + t*t;
        v = v + u;
    }
    return x;
}
```
Strength Reduction

```c
int sumcalc(int a, int b, int N)
{
    int i;
    int x, t, u, v;
    x = 0;
    u = ((a<<2)/b);
    v = 0;
    for(i = 0; i <= N; i++) {
        t = i+1;
        x = x + v + t*t;
        v = v + u;
    }
    return x;
}
```
Register Allocation

Local variable $X$
Local variable $Y$
Local variable $I$
Register Allocation

Local variable $\text{X}$
Local variable $\text{Y}$
Local variable $\text{I}$

$t9 = \text{X}$
$t8 = \text{t}$
$t7 = \text{u}$
$t6 = \text{v}$
$t5 = \text{i}$
int sumcalc(int a, int b, int N)
{
    int i;
    int x, t, u, v;
    x = 0;
    u = ((a<<2)/b);
    v = 0;
    for(i = 0; i <= N; i++) {
        t = i+1;
        x = x + v + t*t;
        v = v + u;
    }
    return x;
}
test:
    subu $fp, 16
    add $t9, zero, zero    # x = 0
    sll $t0, $a0, 2       # a<<2
    div $t7, $t0, $a1     # u = (a<<2)/b
    add $t6, zero, zero   # v = 0
    add $t5, zero, zero   # i = 0

lab1:     # for(i=0;i<N; i++)
    addui$t8, $t5, 1    # t = i+1
    mul $t0, $t8, $t8   # t*t
    addu $t1, $t0, $t6  # v + t*t
    addu $t9, t9, $t1   # x = x + v + t*t

    addu $6, $6, $7     # v = v + u

    addui$t5, $t5, 1    # i = i+1
    ble $t5, $a3, lab1

    addu $v0, $t9, zero
    addu $fp, 16
    b     $ra
Unoptimized Code

```
test:
  subu $fp, 16
  sw  zero, 0($fp)
  sw  zero, 4($fp)
  sw  zero, 8($fp)
lab1:
  mul $t0, $a0, 4
  div $t1, $t0, $a1
  lw  $t2, 8($fp)
  mul $t3, $t1, $t2
  lw  $t4, 8($fp)
  addui $t4, $t4, 1
  lw  $t5, 8($fp)
  addui $t5, $t5, 1
  mul $t6, $t4, $t5
  addu $t7, $t3, $t6
  lw  $t8, 0($fp)
  add $t8, $t7, $t8
  sw  $t8, 0($fp)
  lw  $t0, 4($fp)
  mul $t1, $t0, $a1
  lw  $t2, 0($fp)
  add $t2, $t2, $t1
  sw  $t2, 0($fp)
  lw  $t0, 8($fp)
  addui $t0, $t0, 1
  sw  $t0, 8($fp)
  ble $t0, $a3, lab1
```

Optimized Code

```
test:
  subu $fp, 16
  add $t9, zero, zero
  sll $t0, $a0, 2
  div $t7, $t0, $a1
  add $t6, zero, zero
  add $t5, zero, zero
lab1:
  addui $t8, $t5, 1
  mul $t0, $t8, $t8
  addu $t9, $t0, $t6
  addu $t9, $t9, $t1
  addu $6, $6, $7
  addui $t5, $t5, 1
  ble $t5, $a3, lab1
  addu $v0, $t9, zero
  addu $fp, 16
  b  $ra
```

$$4 \times \text{ld/st} + 2 \times \text{add/sub} + \text{br} + N \times (9 \times \text{ld/st} + 6 \times \text{add/sub} + 4 \times \text{mul} + \text{div} + \text{br}) = 7 + N \times 21$$

$$6 \times \text{add/sub} + \text{shift} + \text{div} + \text{br} + N \times (5 \times \text{add/sub} + \text{mul} + \text{br}) = 9 + N \times 7$$

Execution time = 43 sec  Execution time = 17 sec
Compilers Optimize Programs for...

- Performance/Speed
- Code Size
- Power Consumption
- Fast/Efficient Compilation
- Security/Reliability
- Debugging
Compilers help increase the level of abstraction

- Programming languages
  - From C to OO-languages with garbage collection
  - Even more abstract definitions

- Microprocessor
  - From simple CISC to RISC to VLIW to ….
Anatomy of a Computer

Program written in a Programming Languages

Compiler

Assembly Language Translation
Anatomy of a Computer

Program (character stream)

Lexical Analyzer (Scanner)

Token Stream
Lexical Analyzer (Scanner)

Num(234) mul_op lpar_op Num(11) add_op Num(-22) rpar_op

18..23 + val#ue
Lexical Analyzer (Scanner)

\[ 2 \ 3 \ 4 \ * \ ( \ 1 \ 1 \ + \ - \ 2 \ 2 \ ) \]

\[ \text{Num}(234) \ \text{mul}_\text{op} \ \text{lpar}_\text{op} \ \text{Num}(11) \ \text{add}_\text{op} \ \text{Num}(-22) \ \text{rpar}_\text{op} \]

18..23 + val#ue

Variable names cannot have ‘#’ character

Not a number
Anatomy of a Computer

Program (character stream)
Lexical Analyzer (Scanner)
Token Stream
Syntax Analyzer (Parser)
Parse Tree
Syntax Analyzer (Parser)

```
num '*' '(' num '+' num ')' 
```

Diagram:
```
  <expr><expr><expr><expr><expr>
    |      |      |      |
    expr  op    expr  op    expr
      |      |        |        |
    num  *    (  num  +  num )
```
Syntax Analyzer (Parser)

```c
int * foo(i, j, k))
    int i;
    int j;
{
    for(i=0; i j) {
    fi(i>j)
    return j;
    }
} Extra parentheses
Missing increment
Not an expression
Not a keyword
```
Anatomy of a Computer

- Program (character stream)
- Lexical Analyzer (Scanner)
- Token Stream
- Syntax Analyzer (Parser)
- Parse Tree
- Semantic Analyzer
- Intermediate Representation
Semantic Analyzer

```c
int * foo(i, j, k)
{
    int i;
    int j;
    int x;
    x = x + j + N;
    return j;
}
```

Type not declared
Mismatched return type
Uninitialized variable used
Undeclared variable
Anatomy of a Computer

- Program (character stream)
- Lexical Analyzer (Scanner)
- Token Stream
- Syntax Analyzer (Parser)
- Parse Tree
- Semantic Analyzer
- Intermediate Representation
- Code Optimizer
- Optimized Intermediate Representation
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
        x = x + 4*a/b*i + (i+1)*(i+1);
        x = x + b*y;
    }
    return x;
}

int sumcalc(int a, int b, int N)
{
    int i;
    int x, t, u, v;
    x = 0;
    u = ((a<<2)/b);
    v = 0;
    for(i = 0; i <= N; i++) {
        t = i+1;
        x = x + v + t*t;
        v = v + u;
    }
    return x;
}
Anatomy of a Computer

1. **Program (character stream)**
2. **Lexical Analyzer (Scanner)**
3. **Token Stream**
4. **Syntax Analyzer (Parser)**
5. **Parse Tree**
6. **Semantic Analyzer**
7. **Intermediate Representation**
8. **Code Optimizer**
9. **Optimized Intermediate Representation**
10. **Code Generator**
11. **Assembly code**
int sumcalc(int a, int b, int N)
{
    int i;
    int x, t, u, v;
    x = 0;
    u = ((a<<2)/b);
    v = 0;
    for(i = 0; i <= N; i++) {
        t = i+1;
        x = x + v + t*t;
        v = v + u;
    }
    return x;
}