

תורת הקומפילציה

תורת הקומפילציה
אהרון נץ

מבוסס על השקפים של עומר ביהם
שמבוססים על שקפי הרצאה מהקורס תורת הקומפילציה בטכניון
ד"ר שירלי הלוי

1

תורת הקומפילציה

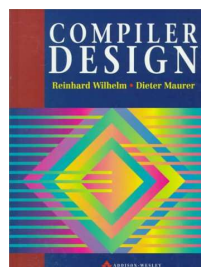
מידע כללי ועדכונים שוטפים באתר הקורס:
<http://cs.haifa.ac.il/courses/compilers/>

2

ספרות

■ ספר עיקרי

R. Wilhelm, and D. Maurer – “Compiler Design”, Addison-Wesley, 1995



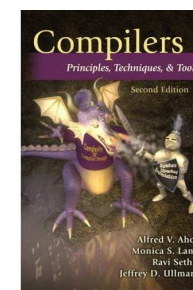
3

ספרות

■ ספר משני

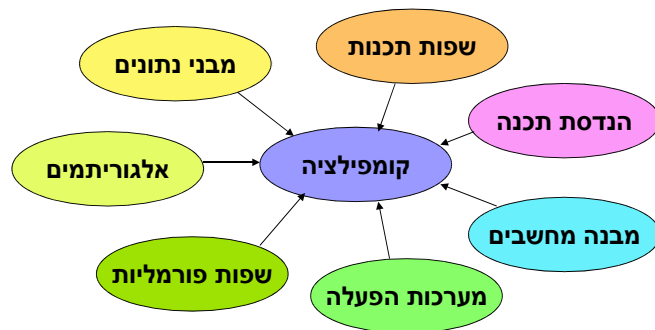
by Alfred V. Aho (Author), Monica S. Lam (Author), Ravi Sethi (Author), Jeffrey D. Ullman (Author)

“Compilers: Principles, Techniques, and Tools (2nd Edition) “



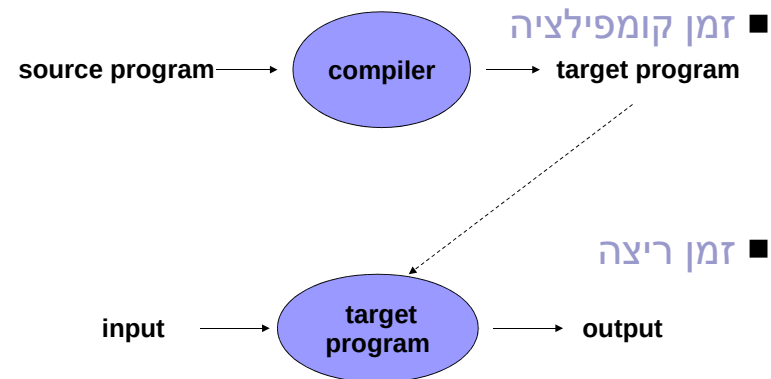
4

הידור – נושא מורכב



5

הידור – מושגי יסוד



6

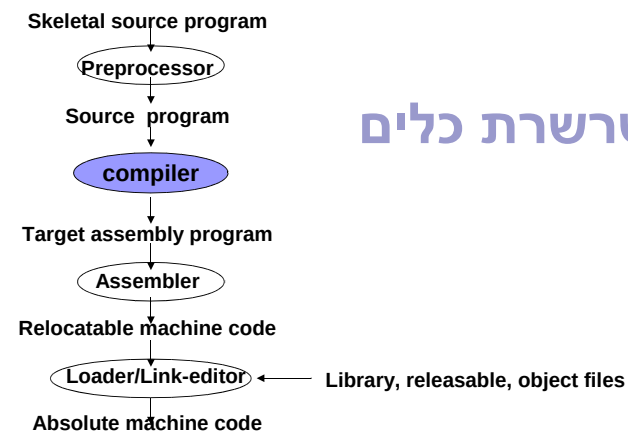
שיטות הידור – שימושים

קלט	יעד
שפות תכנות:	יקוד מכונה:
C, Pascal, Assembler,...	SPARC, P690, IA32
שפות ייצוג מסמכים:	יקוד עבור אינטרפרטר:
TeX, HTML	Java VM, P-Code, ...
שפות scripting:	שפות ייצוג מסמכים:
C-shell, perl	PostScript, PDF
שפות שאילתה לעבוד נתונים (SQL)	
שפות לתאור חומרה (VHDL)	
שפות בקרה	

7

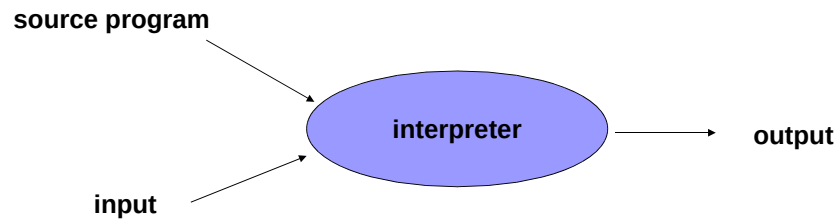
ההקשר הרחב – דוגמא

שרשרת כלים



8

אינטרפרטציה = פרשנות



9

הידור / אינטרפרטציה – הכללות

- Just In Time -- תוך כדי פרשנות התכנית, ה-`interpreter` מבצע קומפילציה של חלקי תוכנית על מנת שהמשך הביצוע יהיה מהיר יותר. דוגמא: `Java`

■ Source to Source

`C ++ program` → `translator` → `C program`

■ Virtual Machine

`Java program` → `compiler` → `Java bytecode`

■ Pre-processors

`program with embedded pre-processing statements (e.g., #include, macros)` → `preprocessor` → "pure" program

10

קומפילציה – חשיבות התחום

- לפיתוח שמושים מתקדמים
 - ניצול של כלים לפיתוח קומפיילרים להקלת מאמץ הפיתוח
 - כל מה שצריך לקרוא קובץ קלט (קבצי קונפיגורציה ואילך)
- למפתחי תוכנה
 - הבנה מעמיקה של האבחנה בין זמן קומפילציה לזמן ריצה
 - הבנה מעמיקה מה יעשה קומפיילר עם התוכנה שלכם (שיפור ריצתה, התראת שגיאותיה)
 - שימוש נכון במבנים שונים של שפות התכנות
 - ניצול נכון של ארכיטקטורת המחשב

11

קומפילציה – חשיבות התחום

- לאנשי שפות תכנות
 - תמיכה יעילה בשפה חדשה
- לאנשי ארכיטקטורה של מחשבים
 - הבנה טובה של האיזון העדין שבין חומרה לתכנה
 - הבנה מעמיקה מה יעשה קומפיילר (כלומר: מה תעשה כל תוכנה) עם החומרה שלכם
- לסטודנטים בפקולטה למדעי המחשב
 - חובה להשלמת התואר

12

תורת הקומפילציה – תכנים עיקריים

- עקרונות
- מבנה הקומפיילר
- ניתוח מילוני (lexical analysis)
- ניתוח תחבירי (parsing)
- ניתוח סמנטי
- יצירת קוד
- נושאים מתקדמים:
 - אופטימיזציה
 - ניתוח סטטי
 - Data-flow analysis
 - קומפיילרים Just-In-Time ו-Virtual Machines
 - קומפיילרים "פתוחים"

13

מבנה הקומפיילר – תמונה כללית

- Wilhelm and Maurer – Chapter 6
- Aho, Sethi, and Ullman – Chapter 1
- Cooper and Torczon – Chapter 1

14

קומפיילר – כקופסא שחורה

```
int a, b;  
a = 2;  
b = a*2 + 1;
```

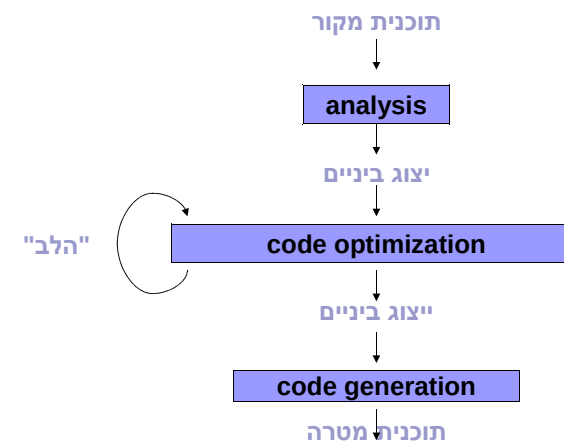
source
code

target code

```
SET   R1, 2  
STORE #0, R1  
SHIFT R1, 1  
STORE #1, R1  
ADD   R1, 1  
STORE #2, R1
```

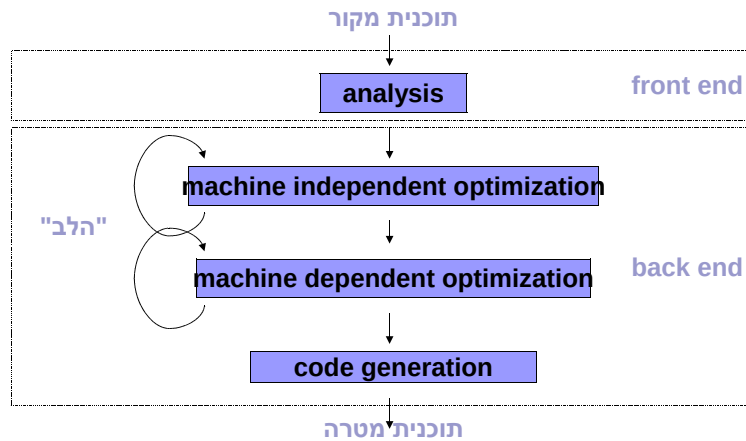
15

קומפיילר – מבנה סכמתי



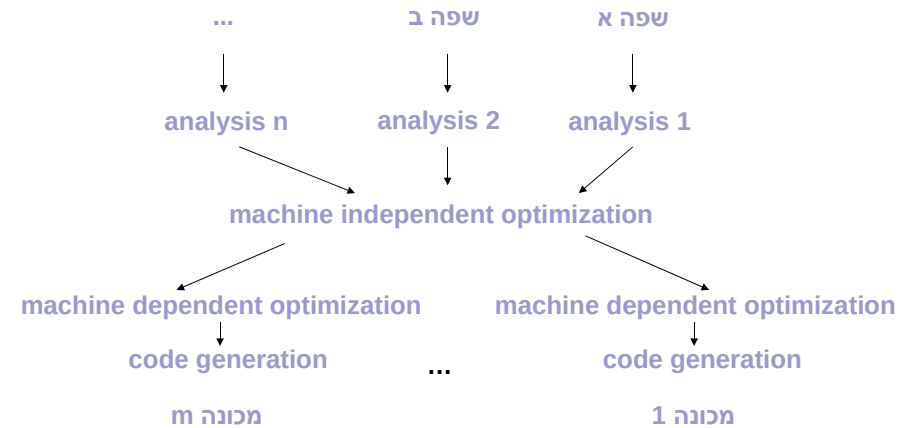
16

קומפיילר – מבנה סכמתי



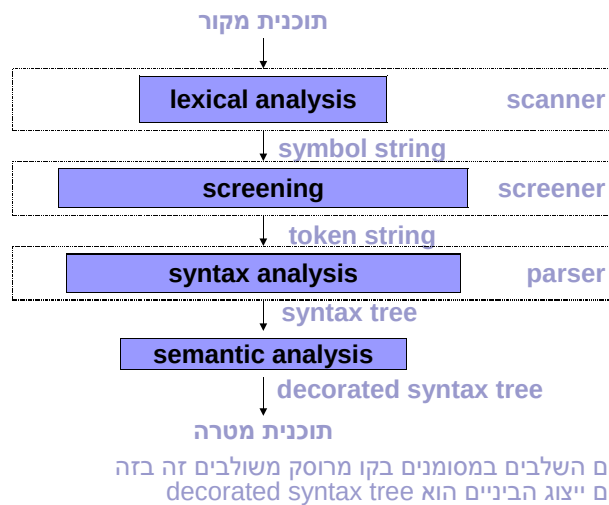
17

שימוש במרכיבי הקומפיילר



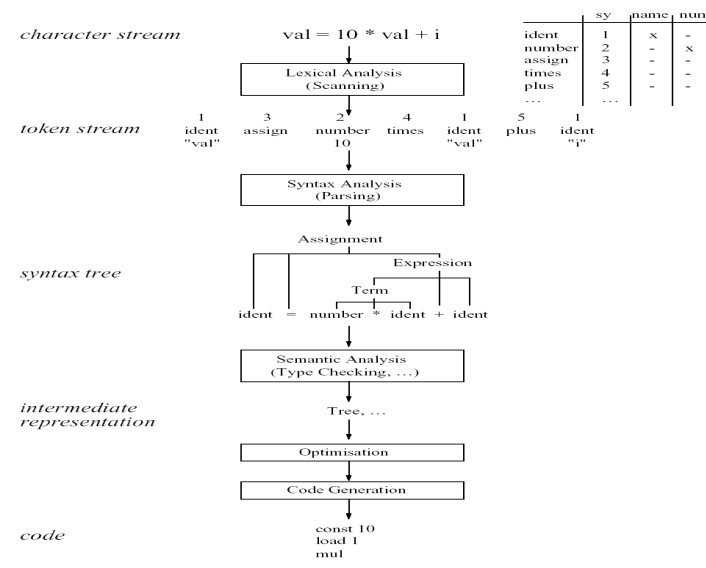
18

front end – שלב הניתוח



19

Dynamic Structure of a Compiler



20

אנליזה של ביטוי

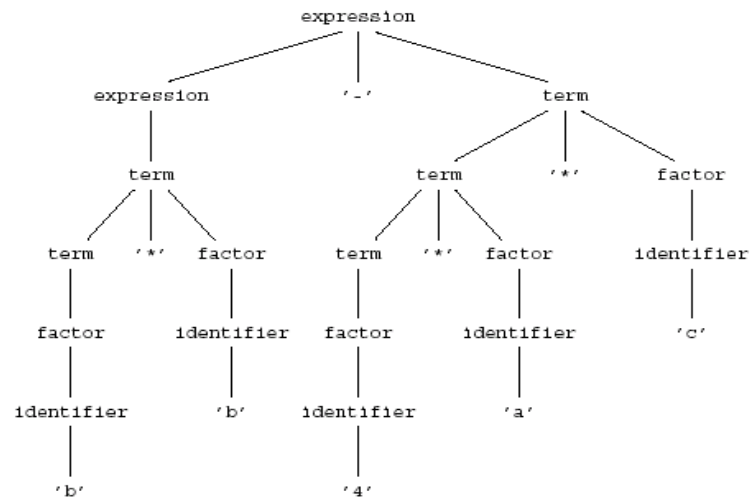


Figure 1.5 The expression $b*b - 4*a*c$ as a parse tree.

21

Abstract Syntax Tree – AST

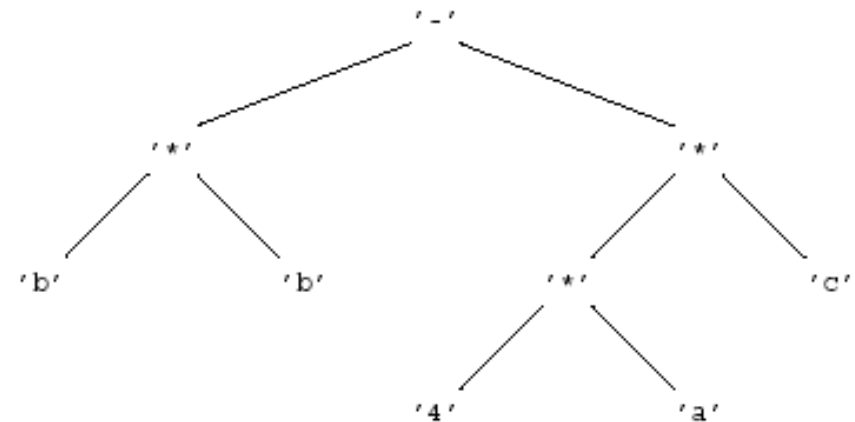


Figure 1.6 The expression $b*b - 4*a*c$ as an AST.

22

Decorated/Annotated AST

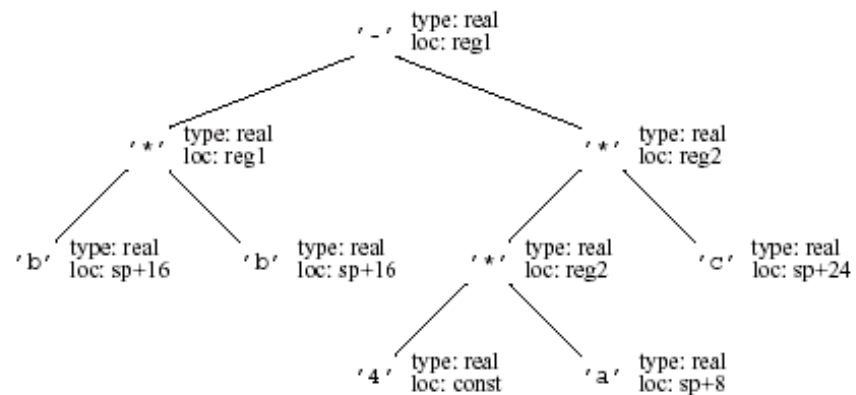


Figure 1.7 The expression $b*b - 4*a*c$ as an annotated AST.

23

מבנה הקומפיילר

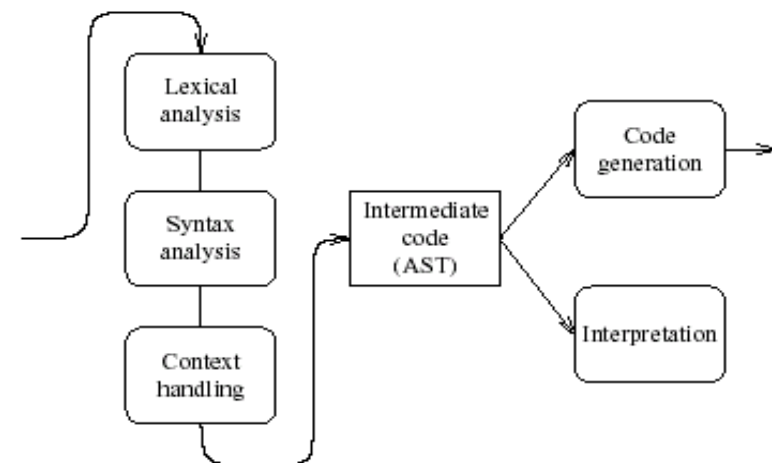
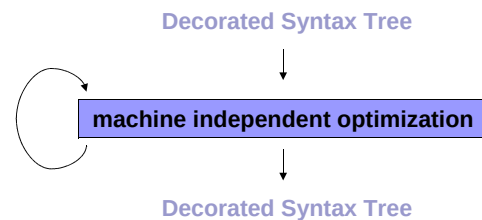


Figure 1.8 Structure of the demo compiler/interpreter.

שלב האופטימיזציה

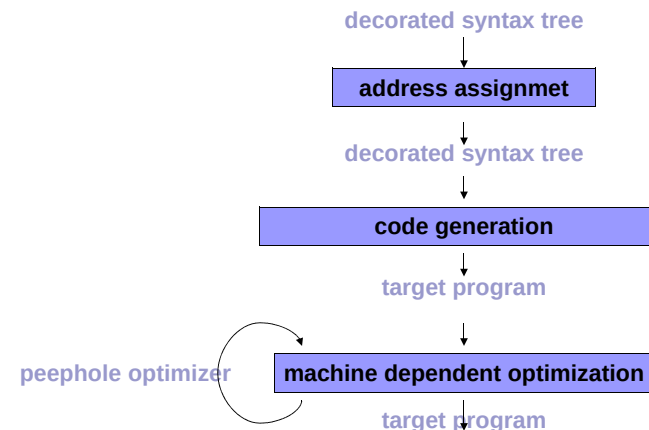


דוגמאות

- constant propagation ☐
- common subexpressions ☐
- dead code elimination ☐

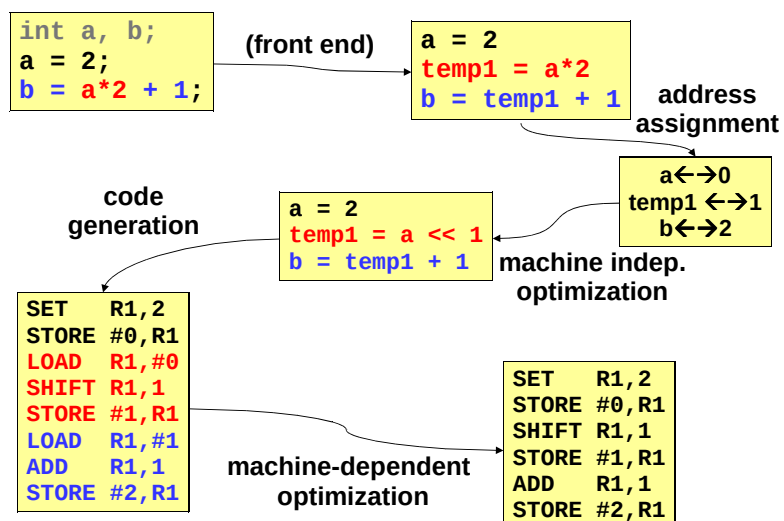
25

שלב הסינתזה (back-end)



26

ה-back-end, דוגמא



27

אופטימיזציה - דוגמא

- Example taken from
- 6.035 Computer Language Engineering - Fall 2007
- <http://web.mit.edu/6.035/www/>

- ☐ Constant/Copy propagation
- ☐ Algebraic Simplification
- ☐ common subexpressions
- ☐ dead code elimination
- ☐ Loop Invariant Removal
- ☐ Strength Reduction
- ☐ Register Allocation
- ☐ benchmarking

28

...Compilers Optimize Programs for

- Performance/Speed
- Code Size
- Power Consumption
- Fast/Efficient Compilation
- Security/Reliability
- Debugging

29

Optimization Example

```
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;
}
```

30

```
pushq   %rbp
movq    %rsp, %rbp
movl    %edi, -4(%rbp)

movl    %esi, -8(%rbp)
movl    %edx, -12(%rbp)
movl    $0, -20(%rbp)
movl    $0, -24(%rbp)
.L2:    movl    $0, -16(%rbp)
        movl    -16(%rbp), %eax
        cmpl    -12(%rbp), %eax
        jg      .L3
        movl    -4(%rbp), %eax
        leal    0(,%rax,4), %edx
        leaq    -8(%rbp), %rax
        movq    %rax, -40(%rbp)
        movl    %edx, %eax
        movq    -40(%rbp), %rcx
        cltd

        idivl   (%rcx)
        movl    %eax, -28(%rbp)
        movl    -28(%rbp), %edx
        imull   -16(%rbp), %edx
        movl    -16(%rbp), %eax
        incl    %eax
        imull   %eax, %eax
        addl    %eax, %edx
        leaq    -20(%rbp), %rax
        addl    %edx, (%rax)
        movl    -8(%rbp), %eax
        movl    %eax, %edx
        imull   -24(%rbp), %edx
        leaq    -20(%rbp), %rax
        addl    %edx, (%rax)
        leaq    -16(%rbp), %rax
        incl    (%rax)
        jmp     .L2
.L3:    movl    -20(%rbp), %eax
        leave   %eax
        ret
```

31

Lets Optimize...

```
int sumcalc(int a, int b, int N)
{
    int i, 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;
}
```

32

Constant Propagation

```
int sumcalc(int a, int b, int N)
{
    int i, 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;
}
```

33

Constant Propagation

```
int sumcalc(int a, int b, int N)
{
    int i, 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;
}
```

34

Constant Propagation

```
int sumcalc(int a, int b, int N)
{
    int i, 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;
}
```

35

Algebraic Simplification

```
int sumcalc(int a, int b, int N)
{
    int i, 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;
}
```

36

Algebraic Simplification

```
int sumcalc(int a, int b, int N)
{
    int i, 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;
}
```

37

Algebraic Simplification

```
int sumcalc(int a, int b, int N)
{
    int i, 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;
}
```

38

Copy Propagation

```
int sumcalc(int a, int b, int N)
{
    int i, 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;
}
```

39

Copy Propagation

```
int sumcalc(int a, int b, int N)
{
    int i, 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;
}
```

40

Copy Propagation

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

41

Common Subexpression Elimination

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

42

Common Subexpression Elimination

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

43

Common Subexpression Elimination

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

44

Dead Code Elimination

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

45

Dead Code Elimination

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

46

Dead Code Elimination

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

47

Loop Invariant Removal

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

48

Loop Invariant Removal

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

49

Loop Invariant Removal

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

50

Strength Reduction

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

51

Strength Reduction

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

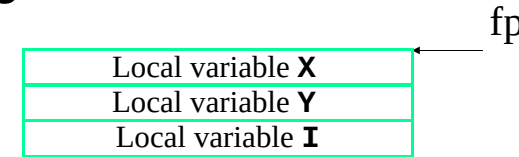
52

Strength Reduction

```
int sumcalc(int a, int b, int N)
{
    int i, x, y, 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 = u*i
    }
    return x;
}
```

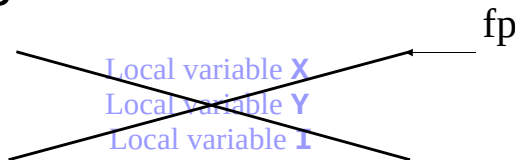
53

Register Allocation



54

Register Allocation



\$r8d = X
\$r9d = t
\$r10d = u
\$ebx = v
\$ecx = i

55

Optimized Example

```
int sumcalc(int a, int b, int N)
{
    int i, x, y, 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 = u*i
    }
    return x;
}
```

56

Unoptimized Code

Optimized Code

```

pushq %rbp
movq %rbp, %rbp
movl %edi, -4(%rbp)
movl %esi, -8(%rbp)
movl %edx, -12(%rbp)
movl $0, -20(%rbp)
movl $0, -24(%rbp)
movl $0, -16(%rbp)
.L2: movl -16(%rbp), %eax
      cmpl -12(%rbp), %eax
      jg .L3
      movl -4(%rbp), %eax
      leal 0(,%rax,4), %edx
      leaq -8(%rbp), %rax
      movq %rax, -40(%rbp)
      movl %edx, %eax
      movq -40(%rbp), %rcx
      cld
      idivl (%rcx)
      movl %eax, -28(%rbp)
      movl -28(%rbp), %edx
      imull -16(%rbp), %edx
      movl -16(%rbp), %eax
      incl %eax
      imull %eax, %eax
      addl %eax, %edx
      leaq -20(%rbp), %rax
      addl %edx, (%rax)
      movl -8(%rbp), %eax
      movl %eax, %edx
      imull -24(%rbp), %edx
      leaq -20(%rbp), %rax
      addl %edx, (%rax)
      leaq -16(%rbp), %rax
      incl (%rax)
      jmp .L2
.L3: movl -20(%rbp), %eax
      leave
      ret

```

```

xorl %r8d, %r8d
xorl %ecx, %ecx
movl %edx, %r9d
cmpl %edx, %r8d
jg .L7
sall $2, %edi
.L5: movl %edi, %eax
      cld
      idivl %esi
      leal 1(%rcx), %edx
      movl %eax, %r10d
      imull %ecx, %r10d
      movl %edx, %ecx
      imull %edx, %ecx
      leal (%r10,%rcx), %eax
      movl %edx, %ecx
      addl %eax, %r8d
      cmpl %r9d, %edx
      jle .L5
.L7: movl %r8d, %eax
      ret

```

Inner Loop:

10*mov + 5*lea + 5*add/inc

+ 4*div/mul + 5*cmp/br/jmp

= 29 instructions

Execution time = 43 sec

4*mov + 2*lea + 1*add/inc+

3*div/mul + 2*cmp/br/jmp

= 12 instructions

Execution time = 17 sec