Course: Compiler Construction

Week 10: Target Code Generation

Lecturer: Martha Gichuki

Course Description

- The course begins with a coverage of basic Principles in Compiler Design with an introduction to Formal programming language translation and compiler design concepts; compilers and interpreters.
- A coverage of Language theory, Parsing Context-Free Languages (Top - down and bottom - up parsing), translation specifications and machine- independent code optimization will then follow.

Course Description …

- The main phases of compilation i.e. Lexical, Syntax and Semantic analysis will be taught.
- Code generation and Optimization, Symbol table design, Program compilation, Loading and execution will then be covered.
- Compilation techniques, Optimization, Design of a simple complete compiler will culminate the course.

Learning outcomes Week 10: Target Code Generation

At the end of the lecture, you will be able to:

i. Describe Input IR, target program
ii. Describe the role of the Front-end and Back-end parts of a compiler
iii. Describe Symbol table information

Target Code Generation

- ✓**Code generation** is the final phase of a compiler.
- ✓The code generator maps the **Intermediate Representation(IR)** produced by the front end, along with relevant symbol table information or if there is a **code optimization** phase by the code optimizer, into the **target program[1].**

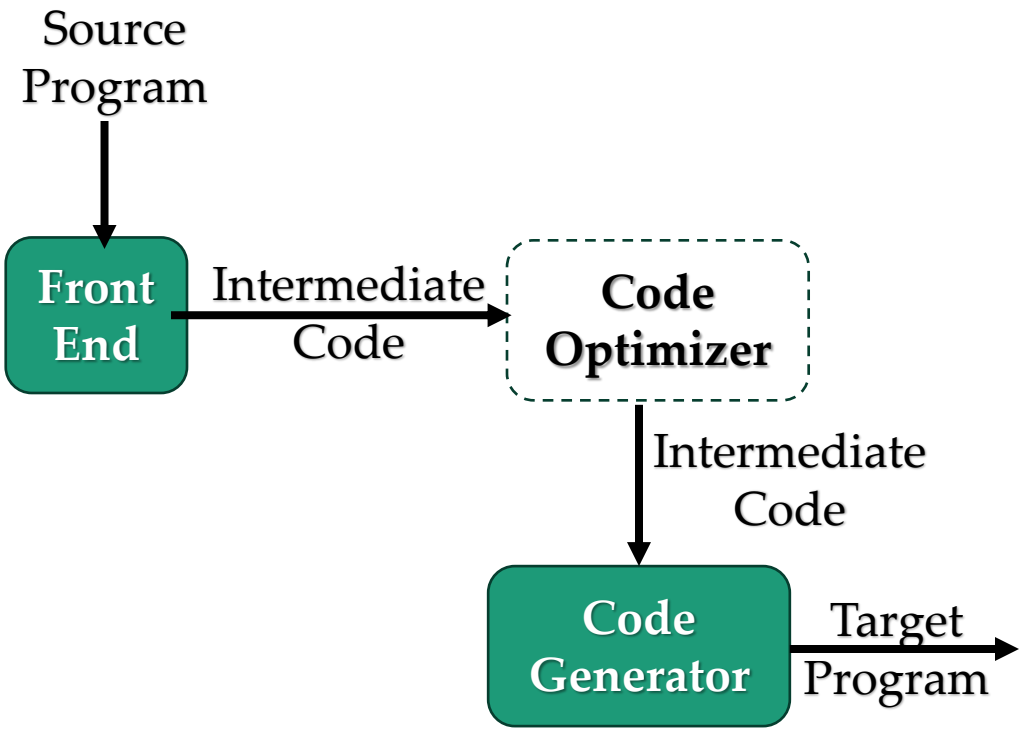
1. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 530

Code Generator Requirements[2]

i. Target program must *preserve the semantic meaning of the source program*
ii. Target program must be of *high quality*
iii. Target program must make *effective use of the available resources* of the target machine
iv. Code generator itself must *run efficiently*

2. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 530

- ✓ Compilers that need to produce efficient target programs include an optimization phase prior to code generation as shown below[3]



3. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 530

Two assumptions made in this lecture

I. The front end has scanned parsed and translated the source program into a relatively low-level IR.

✓ This ensures that values such as integers and floating point numbers) of the names appearing in the IR can be represented by quantities that can be manipulated directly by the target machine[4].

4. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 532

Two assumptions made in this lecture

II. All syntactic and static semantic errors have been detected, necessary type checking has taken place, type conversion operators have been inserted where necessary.

✓ Therefore, the code generator proceeds on the assumption that the input is free of these kinds of errors[5]

5. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 532

- ✓ The two main parts of a compiler are:-

i. **Analysis (Front-end)** and
ii. **Synthesis (Back-end)**[6]

6. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 530

Role of the Analysis (Front-end)

i. *Breaking up* the source program into constituent pieces and imposes a grammatical structure on them.

✓ It then uses this structure to create an **Intermediate Representation** of the source program.

ii. *Error reporting* - If the analysis part detects that the source program is either syntactically ill formed or semantically unsound, then it must provide informative messages (**error report**), so the user can take corrective action[7].

7. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 530

Role of the Analysis (Front-end) …

iii. Collecting information about the source program and storing it in a data structure called a **symbol table**, which is passed along with the IR to the synthesis part[8]

8. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 530

Role of the Synthesis (Back-end)

i. Constructing the desired target program using the IR and Symbol Table information.

✓ Generally, code-optimization and code-generation phases of a compiler, often referred to as the **back-end**, may make several passes over the IR before generating the target program[9]

9. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 530

- ✓ The optimizer maps the IR into IR from which more efficient code can be generated.
- ✓ In this lecture, we discuss techniques that can be used whether optimization phase occurs before code generation or not[10].

10. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 532

A code generator has three primary tasks:

i. **Instruction selection** is the process of choosing **target-language instructions** to implement each IR statement
ii. **Register allocation** is the process of deciding **which IR values** to keep in registers. Graph coloring is an effective technique for doing register allocation in compilers[11].

11. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 535

iii. **Register assignment** is the process of deciding **which** register should **hold a given IR value**[12].

12. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 536

Instruction Selection

- ✓ The code generator must map the IR program into a code sequence that can be executed by the target machine.
- ✓ The complexity of performing this mapping is determined by factors such as:-

i. **level** of the IR

ii. **nature** of the Instruction-Set Architecture

iii. the **desired quality** of the generated code[13].

13. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 533

- ✓The **nature of the instruction set** of the target machine has a strong effect on the **difficulty** of instruction selection.
- ✓For example, the **uniformity** and **completeness** of the instruction set are important factors[14].

14. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 534

- ✓If the target machine does not support each data type in a uniform manner, then each exception to the general rule requires **special handling**.
- ✓On some machines, for example, **floating-point operations** are done using **separate registers**

- ✓Instruction speeds and machine idioms are other important factors.
- ✓For each type of three-address statement, we can design a code skeleton that *defines the target code to be generated for that construct*[15].

15. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 28, 533/534

Example:-

- ✓Every three-address statement of the form **x=y + z**, where **x**, **y**, and **z** are **statically allocated**, can be translated into the following code sequence[16]:-

```
LD R0, y          //R0 = y (load y into register R0)
ADD R0, R0, z     //R0 = R0 + z (add z to R0)
ST x, R0          //x = R0 (store R0 into x)
```

16. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 534

✓This strategy often produces **redundant loads and stores**.

Example:-

✓Consider the following sequence of three-address statements[17]

$$a = b + c$$

$$d = a + e$$

17. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 534

These statements would be translated into[18]:-

a = b +c
d = a + e

```
LD R0, b          // R0 = b
ADD R0, R0, c     // R0 = R0 + c
ST a, R0          // a = R0
LD R0, a          // R0 = a
ADD R0, R0, e     // R0 = R0 + e
ST d, R0          // d = R0
```

✓ *Here, the fourth statement is* ***redundant*** *since it loads a value* ***a*** *that has just been stored, and so is the third if* ***a*** *is not subsequently used.*

18. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 534

Register Allocation

- ✓ A key problem in code generation is ***deciding what values to hold in what registers***.
- ✓ Registers are the **fastest computational units** on the target machine, but we usually do not have enough of them to hold all values[19].

19. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 535

Register Allocation…

- ✓ Values that are not held in **registers** need to reside in **memory**.
- ✓ Instructions which involve register operands are invariably **shorter** and **faster** than those involving operands in memory, so efficient utilization of registers is particularly important[20].

20. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 535

The use of registers is often subdivided into two sub-problems:

i. ***Register allocation***, during which we select the set of variables that will reside in registers at each point in the program.

ii. ***Register assignment***, during which we pick the specific register that a variable will reside in.

- ✓ Finding an optimal assignment of registers to variables is difficult, even with single-register machines[21].
- ✓ Mathematically, the problem is NP-complete.

21. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 535

- ✓ Consider the two three-address code sequences in which the only difference in (a) and (b) is the ***operator in the second statement***[22].

t = a + b	t = a + b
t = t * c	t = t + c
t = t / d	t = t / d
(a)	**(b)**

22. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 536

- ✓ The shortest assembly-code sequences for (a) and (b) are given as[23]:-

```
t = a  + b
t = t * c
t= t / d
```
(a)

```
t = a  + b
t = t + c
t= t / d
```
(b)

```
L R1,a
A R1,b
M R0,c
D R0,d
ST R1,t
```
(a)

```
L R0, a
A R0, b
A R0, c
SRDA R0, 32
D R0, d
ST R1, t
```
(b)

23. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 536

- ✓ $\mathbf{R_i}$ stands for register **i**.
- ✓ **SRDA** stands for **S**hift-**R**ight-**D**ouble-**A**rithmetic and
- ✓ **SRDA R0,32** shifts the dividend into **R1** and clears **R0** so all bits equal its sign bit.
- ✓ **L**, **ST**, and **A** stand for **load**, **store**, and **add**, respectively.
- ✓ **Note:** The optimal choice for the register into which **a** is to be loaded depends on what will ultimately happen to $\mathbf{t^{24}}$.

```
L R1,a          L R0, a
A R1,b          A R0, b
M R0,c          A R0, c
D R0,d          SRDA R0, 32
ST R1,t         D R0, d
                ST R1, t
```

24. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 536

Content Covered in Week 10: Target Code Generation

At the end of the lecture, we were able to:

i. Describe Input IR, target program
ii. Describe the role of the Front-end and Back-end parts of a compiler
iii. Describe Symbol table information

Course Text Books

1. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; *Addison- Wesley Pub Co,* ISBN: 0201100886 (2007))
2. Compiler Construction: Principles and practices; Kenneth C. Louden; *Cengage Learning*; 1st edition, ISBN-10 : 0534939724 (1997)
3. Basics of Compiler Design: Torben Mogensen; *DIKU University of Copenhagen Universitetsparken 1 DK-2100 Copenhagen DENMARK* (2007)
4. Engineering a Compiler: 2nd edition; Keith D. Cooper and Linda Torczon; *Morgan Kaufmann Publishers* ISBN: 978-0-12-088478-0 (2003)
5. Compiler Design: Santanu Chattopadhyay; *PHI Learning Publishers*, ISBN 812032725X, 9788120327252 (2005)