جامعة بغداد
كلية التربية للعلوم الصرفة ابن الهيثم
قسم علوم الحاسبات

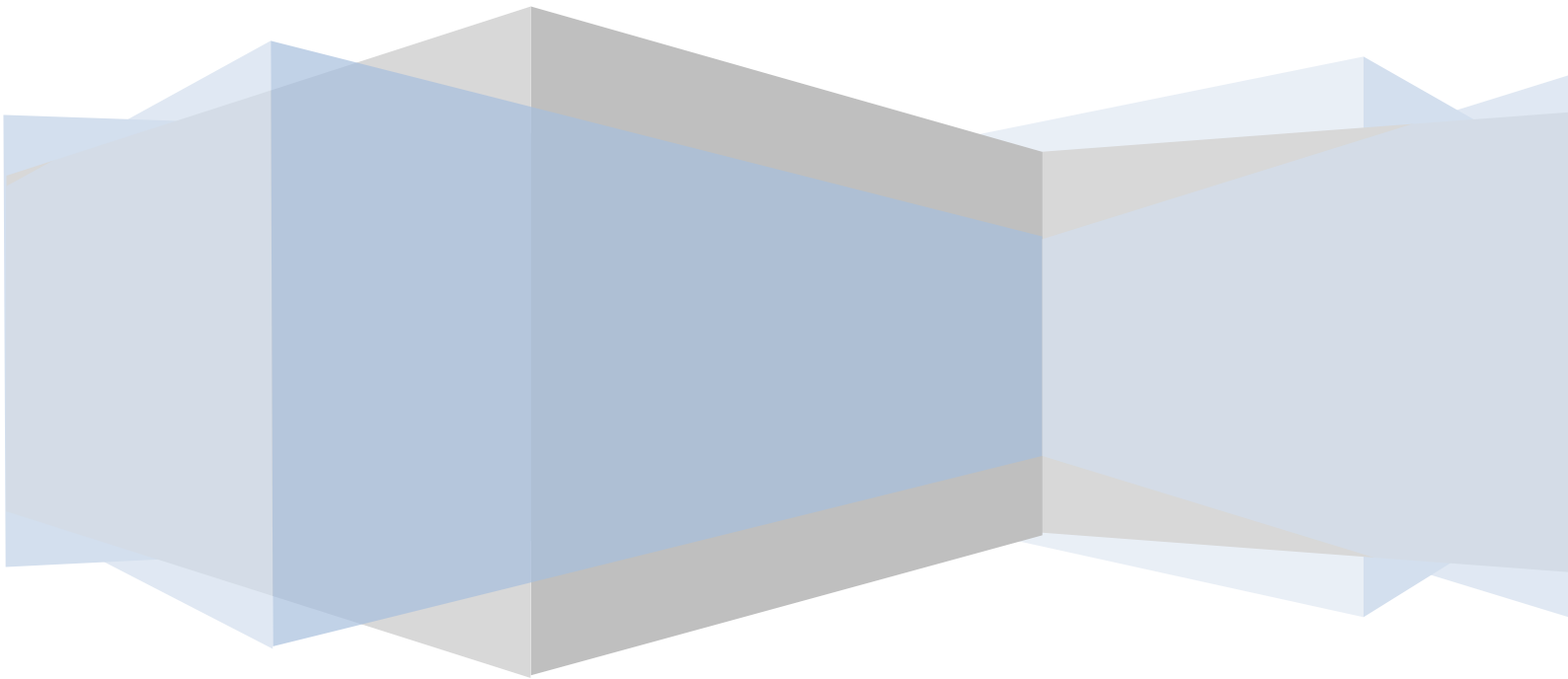
Compilers

Code Generation

Third stage

M.Sc. Ahmed Rafid

2016-2017



Code Generation

Code generation is the final phase of compiler phases, It takes input from the intermediate representation with information in symbol table of the source program and produces as output an equivalent target program (see Figure 1).

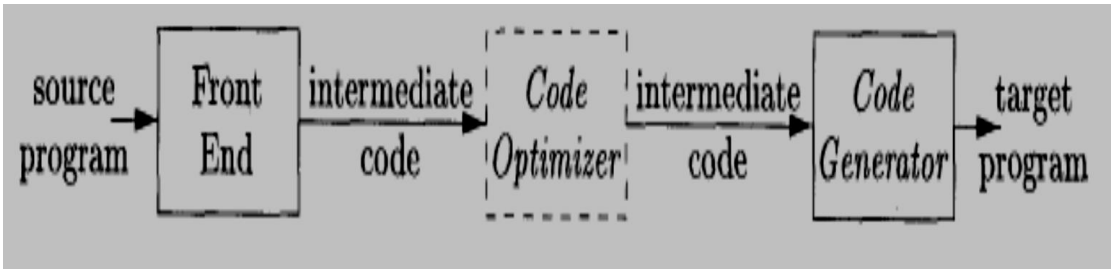


Figure 1: position of Code generation

Main Tasks of Code Generator

1- **Instruction selection**: choosing appropriate target-machine instructions to implement the IR statements.

The complexity of mapping IR program into code-sequence for target machine depends on:

– Level of IR (high-level or low-level)

– Nature of instruction set (data type support)

– Desired quality of generated code (speed and size)

2- **Registers allocation and assignment:** deciding what values to keep in which registers

3- **Instruction ordering:** deciding in what order to schedule the execution of instructions.

Issues in the design of code generator

1- Input to the code generator

- three-address presentations (quadruples, triples, …)
- Virtual machine presentations (bytecode, stack-machine, …)
- Linear presentation (postfix …)
- Graphical presentation (syntax trees, DAGs,…)

2- The target program

Instruction set architecture (RISC, CISC)

The instruction-set architecture of the target machine has a significant impact on the difficulty of constructing a good code generator that produces high-quality machine code. The most common target-machine architectures are RISC (reduced instruction set computer), CISC (complex instruction set computer), and stack based.

A RISC machine typically has many registers, three-address instructions, simple addressing modes, and a relatively simple instruction-set architecture.

In contrast, a CISC machine typically has few registers, two-address instructions, a variety of addressing modes, several register classes, variable-length instructions, and instructions with side effects.

In a stack-based machine, operations are done by pushing operands onto a stack and then performing the operations on the operands at the top of the stack. To achieve high performance the top of the stack is typically kept in registers. Stack-based machines almost disappeared because it was felt

that the stack organization was too limiting and required too many swap and copy operations.

Output may take variety of forms

1. Absolute machine language(executable code)
2. Relocatable machine language(object files for linker)
3. Assembly language(facilitates debugging)

Absolute machine language has advantage that it can be placed in a fixed location in memory and immediately executed.

Relocatable machine language program allows subprograms to be compiled separately.

Producing assembly language program as output makes the process of code generation somewhat easier.