جامعة بغداد
كلية التربية للعلوم الصرفة ابن الهيثم
قسم علوم الحاسبات

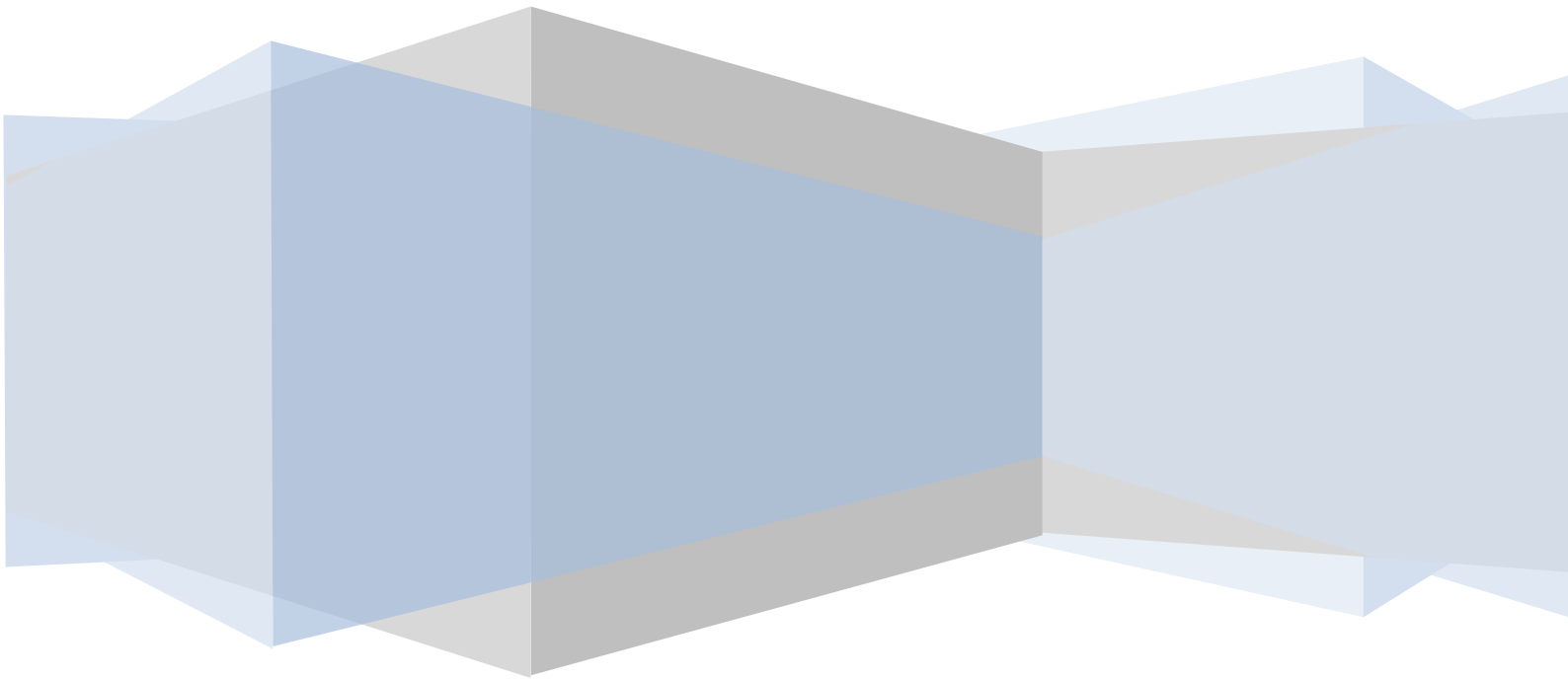
Compilers

Code optimizer

Third stage

M.Sc. Ahmed Rafid

2016-2017



Code Optimization

Optimization is a program transformation technique, which tries to improve the code by making it consume less resources (i.e. CPU, Memory) and deliver high speed.

In optimization, high-level general programming constructs are replaced by very efficient low-level programming codes. A code optimizing process must follow the three rules given below:

- The output code must not, in any way, change the meaning of the program.
- Optimization should increase the speed of the program and if possible.
- The program should demand less number of resources.

Efforts for an optimized code can be made at various levels of compiling the process.

- At the beginning, users can change/rearrange the code or use better algorithms to write the code.
- After generating intermediate code, the compiler can modify the intermediate code by address calculations and improving loops.
- While producing the target machine code, the compiler can make use of memory hierarchy and CPU registers.

Optimization can be categorized broadly into two types: machine independent and machine dependent.

1- Machine-independent Optimization

In this optimization, the compiler takes in the intermediate code and transforms a part of the code that does not involve any CPU registers and/or absolute memory locations. (Machine Independent improvements address the logic of the program)

For example:

```
do
{
item = 10;
  value = value + item;
}while(value<100);
```

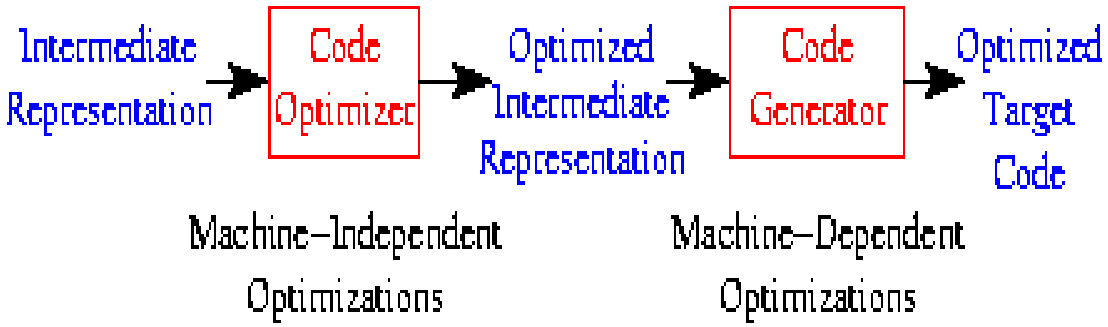
This code involves repeated assignment of the identifier item, which if we put this way:

```
Item = 10;
do
{
  value = value + item;
} while(value<100);
```

should not only save the CPU cycles, but can be used on any processor.

2- Machine-dependent Optimization

Machine-dependent optimization is done after the target code has been generated and when the code is transformed according to the target machine architecture. It involves CPU registers and may have absolute memory references rather than relative references. Machine-dependent optimizers put efforts to take maximum advantage of memory hierarchy.



Peephole optimization: - peephole optimization is a kind of optimization performed over a very small set of instructions in a segment of generated code. The set is called a "peephole" or a "window". It works by recognizing sets of instructions that can be replaced by shorter or faster sets of instructions.

Code Optimization has Two levels which are:-

1- Machine independent code Optimization

- Control Flow analysis
- Data Flow analysis
- Transformation

2- Machine dependent code- Optimization

- Register allocation
- Utilization of special instructions.

Code optimization can either be high level or low level:

- High level code optimizations.
- Low level code optimizations.
- Some optimization can be done in both levels.

Flow graph: - is a common intermediate representation for code optimization.

Basic Blocks

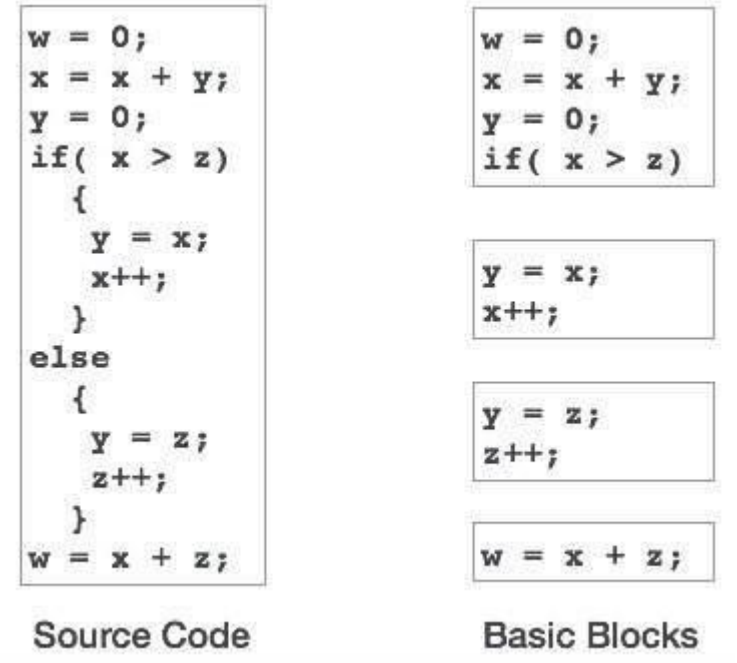
Source codes generally have a number of instructions, which are always executed in sequence and are considered as the basic blocks of the code. These basic blocks do not have any jump statements among them, i.e., when the first instruction is executed, all the instructions in the same basic block will be executed in their sequence of appearance without losing the flow control of the program.

A program can have various constructs as basic blocks, like IF-THEN-ELSE, SWITCH-CASE conditional statements and loops such as DO-WHILE, FOR, and REPEAT-UNTIL, etc.

Basic blocks are important concepts from both code generation and optimization point of view.

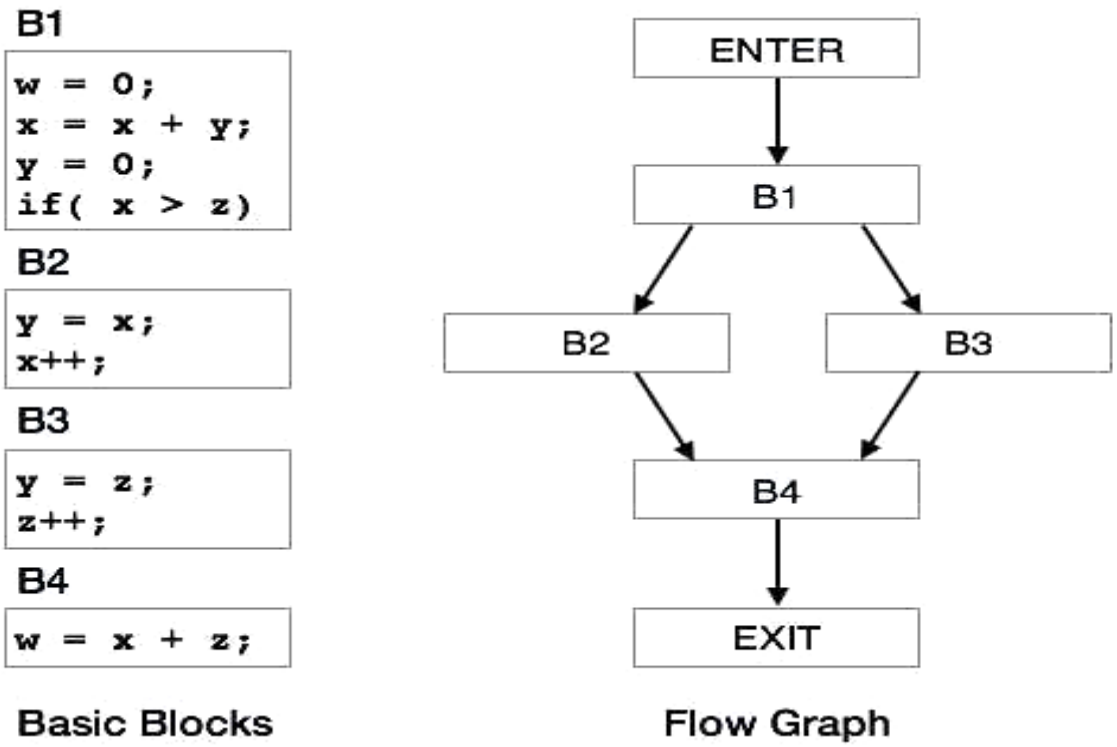
Local Optimizations are performed on basic blocks of code

Global Optimizations are performed on the whole code



Control Flow Graph

Basic blocks in a program can be represented by means of control flow graphs. A control flow graph depicts how the program control is being passed among the blocks. It is a useful tool that helps in optimization by help locating any unwanted loops in the program.



Global Data Flow Analysis

Compiler collect information about all program that needed for code optimizer phase, Collect information about the whole program and distribute the information to each block in the flow graph.

DFA provide information for global optimization about how execution program manipulate data.

- *Data flow information*: Information collected by data flow analysis.
- *Data flow equations*: A set of equations solved by data flow analysis to gather data flow information.

Criteria for code-improvement Transformations

1. Transformations must preserve the meaning of programs
2. A transformation must, on the average, speed up programs by a measurable amount
3. A transformation must be worth the effort.

Function Preserving Transformations

1. Common sub expression eliminations
2. Copy propagations
3. Dead and unreachable code elimination
4. Constant Folding