SNS COLLEGE OF TECHNOLOGY

(Autonomous)

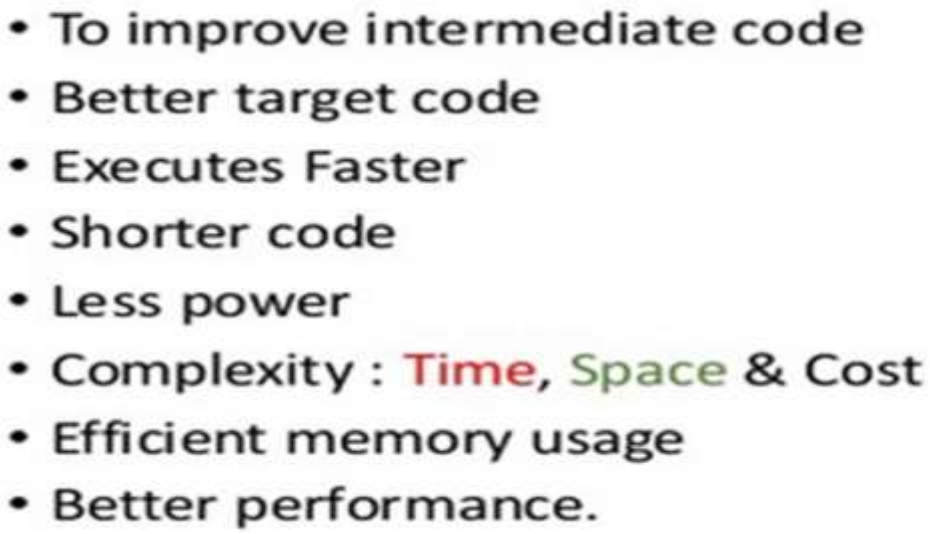
COIMBATORE-35

Introduction to code optimization Techniques&Principle Sources of Optimization

Introduction to code optimization Techniques

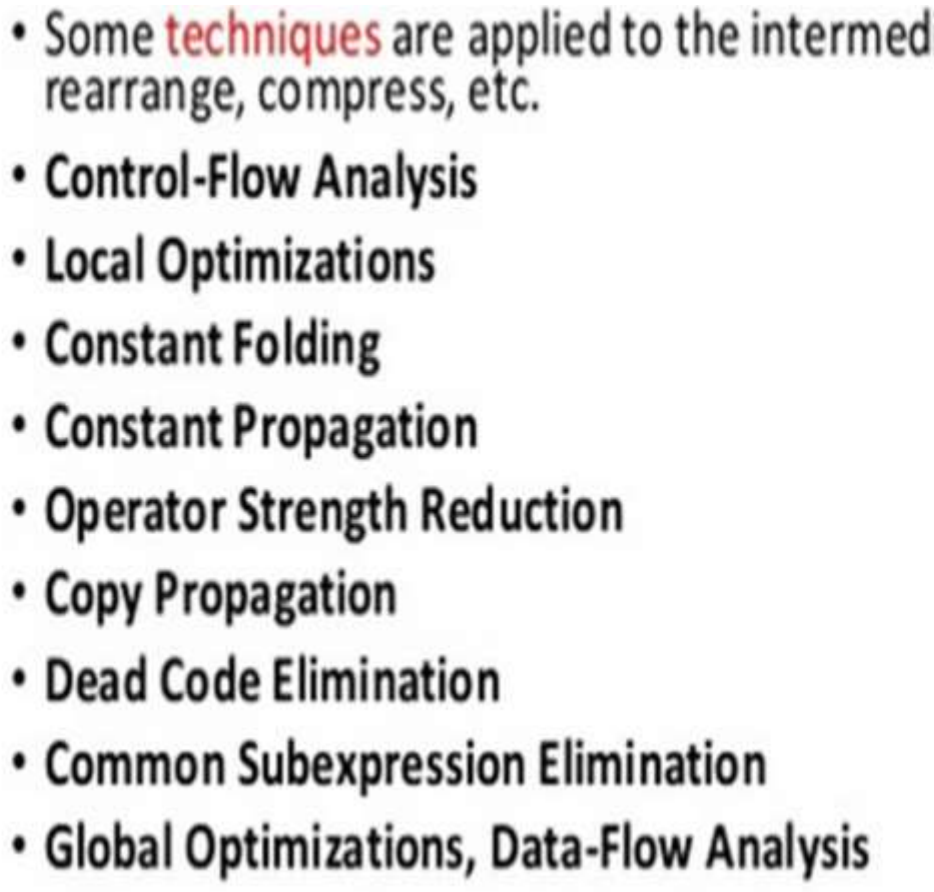
- ***Optimization*** is the process of transforming a piece of code to make more efficient (either in terms of tome or space)without changing its output or side effects

Why Optimization is needed



- To improve intermediate code
- Better target code
- Executes Faster
- Shorter code
- Less power
- Complexity : Time, Space & Cost
- Efficient memory usage
- Better performance.

Introduction to code optimization Techniques

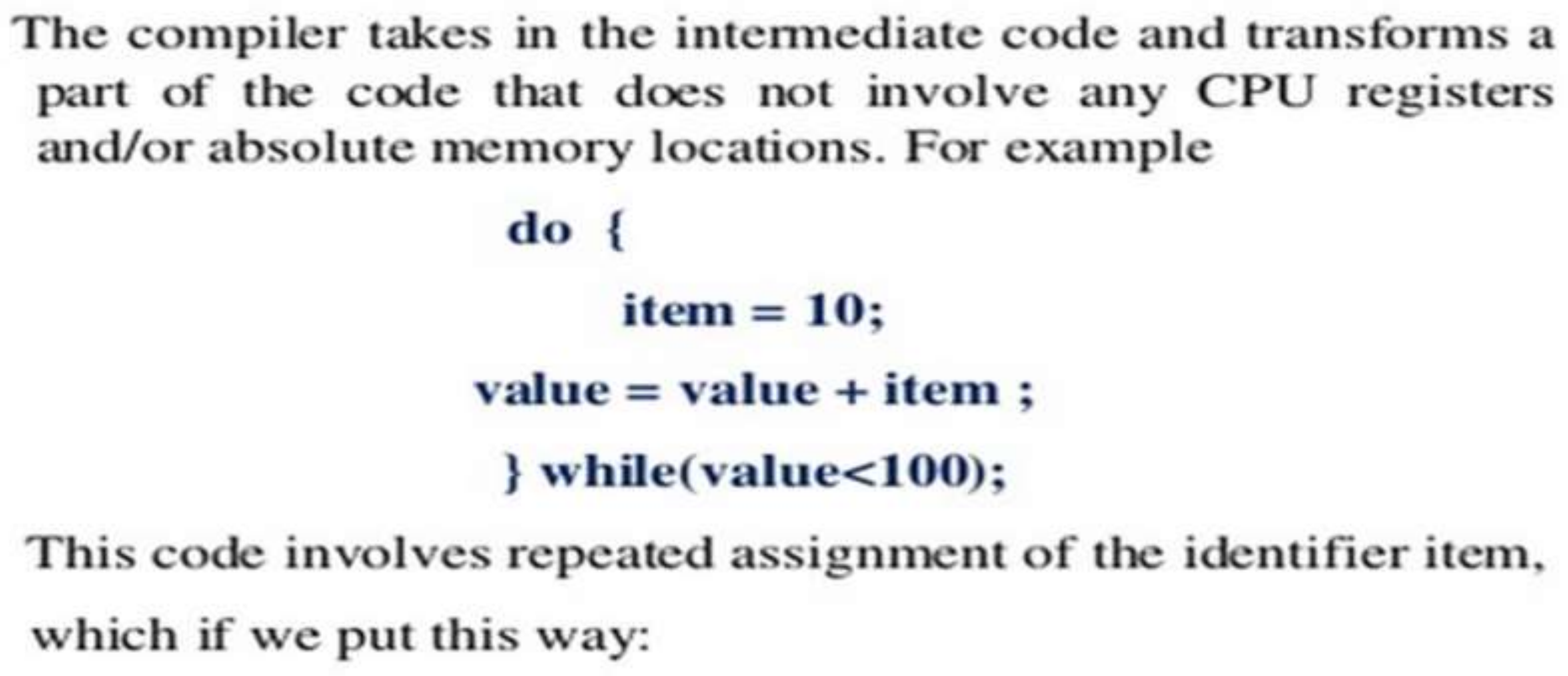


- Some techniques are applied to the intermediate code, to streamline, rearrange, compress, etc.
- Control-Flow Analysis
- Local Optimizations
- Constant Folding
- Constant Propagation
- Operator Strength Reduction
- Copy Propagation
- Dead Code Elimination
- Common Subexpression Elimination
- Global Optimizations, Data-Flow Analysis

Introduction to code optimization Techniques

- Optimization can be categorized into two types
  - ➢ Machine Independent &Machine dependent

Machine Independent



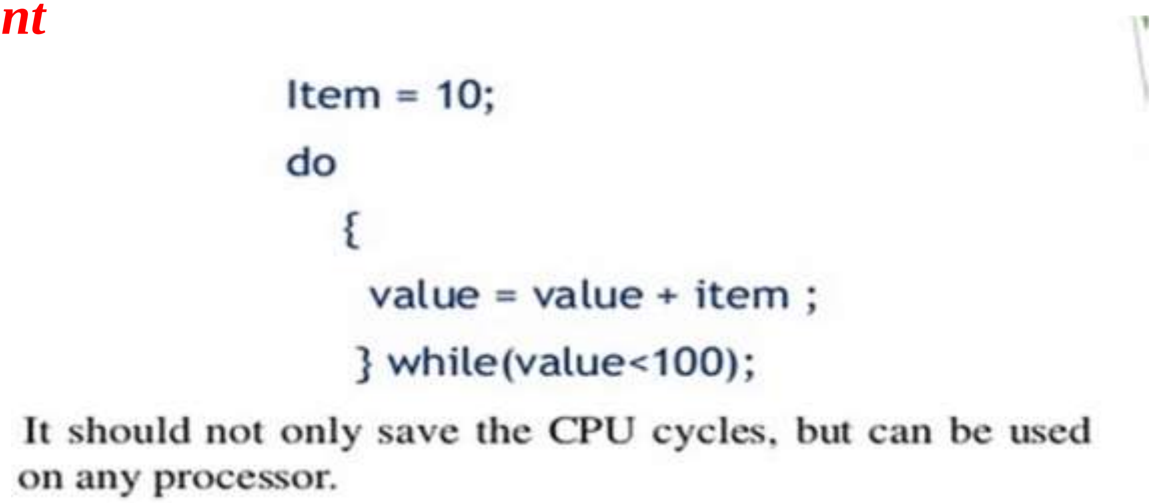
The compiler takes in the intermediate code and transforms a part of the code that does not involve any CPU registers and/or absolute memory locations. For example

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

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

Introduction to code optimization Techniques

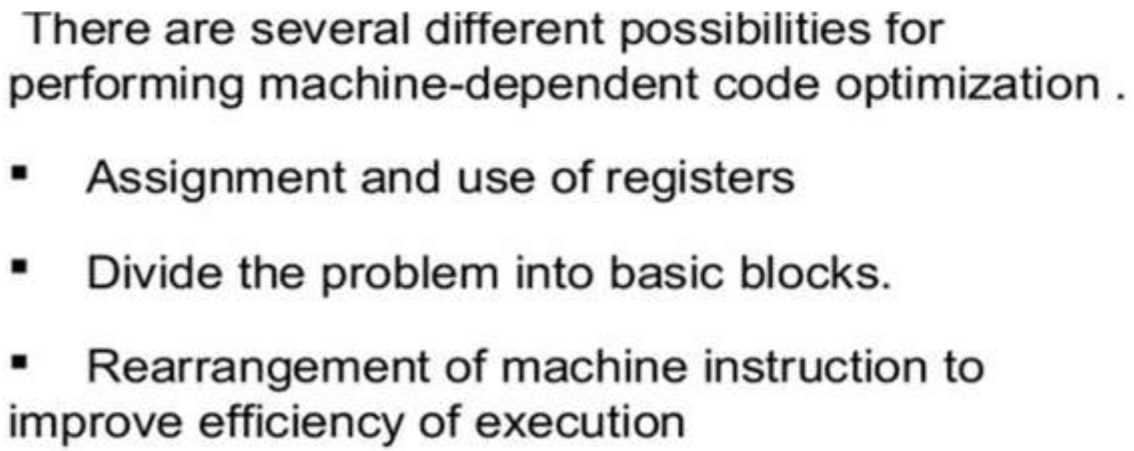
Machine Independent



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

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

Machine Dependent

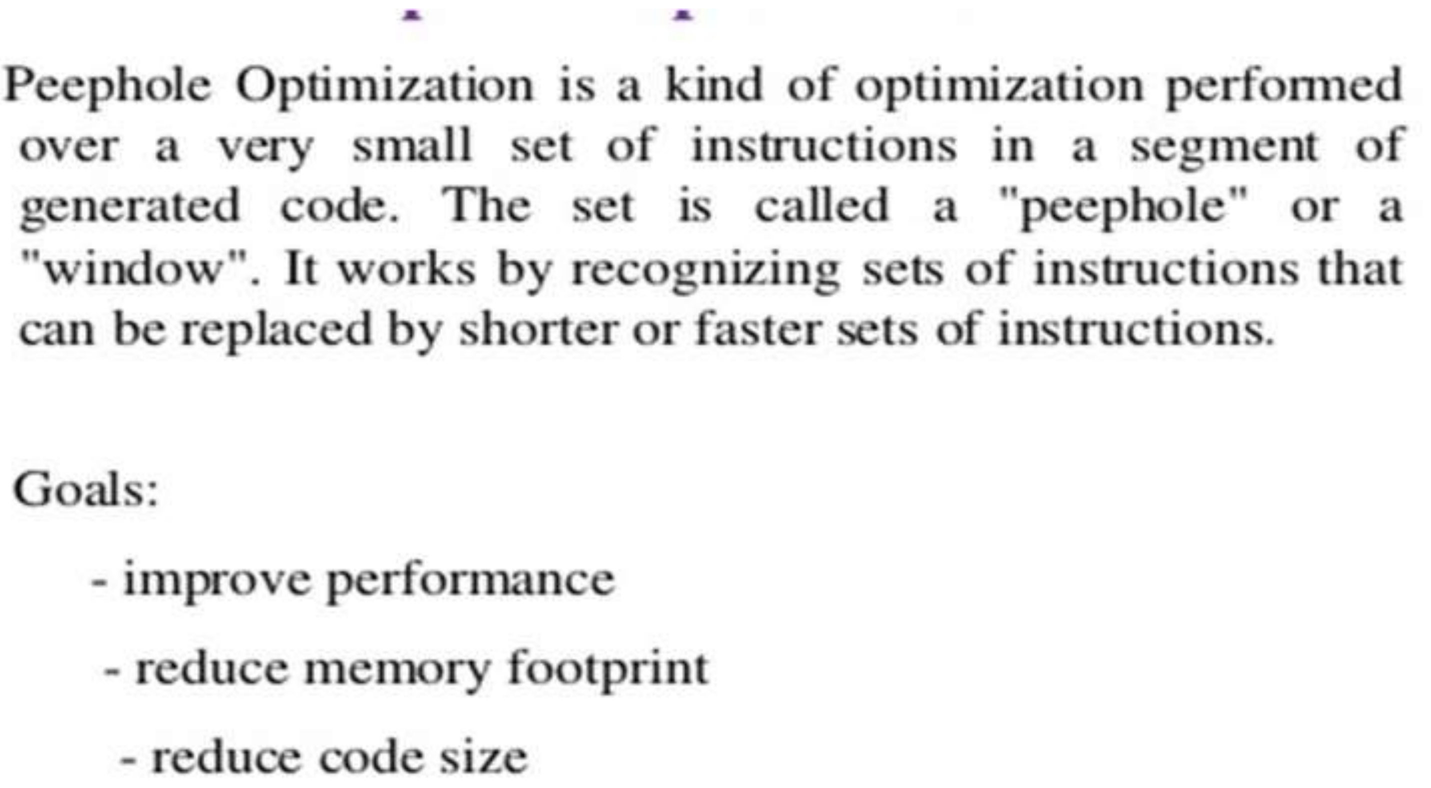


There are several different possibilities for performing machine-dependent code optimization .

- Assignment and use of registers
- Divide the problem into basic blocks.
- Rearrangement of machine instruction to improve efficiency of execution

Introduction to code optimization Techniques

Peephole Optimization

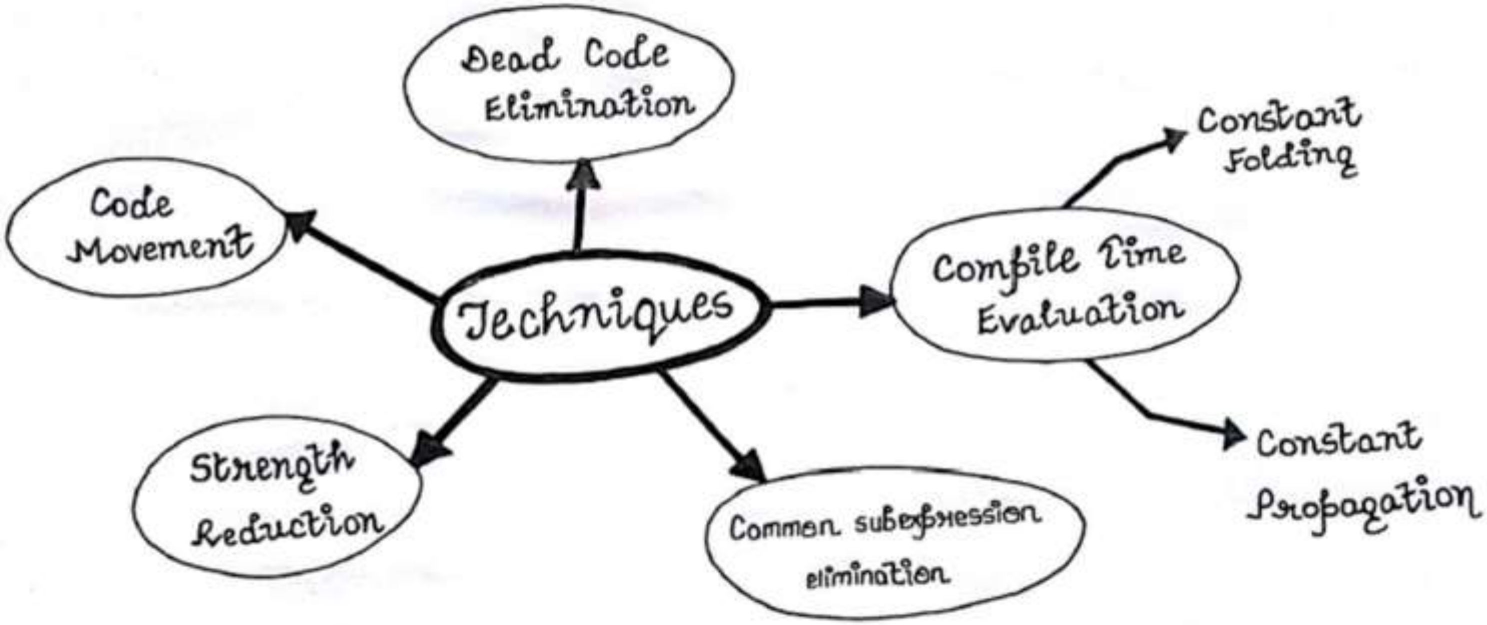


Peephole Optimization is a kind of optimization performed over a very small set of instructions in a segment of generated code. The set is called a "peephole" or a "window". It works by recognizing sets of instructions that can be replaced by shorter or faster sets of instructions.

Goals:

- improve performance
- reduce memory footprint
- reduce code size

Principles Sources of Optimization



Principles Sources of Optimization

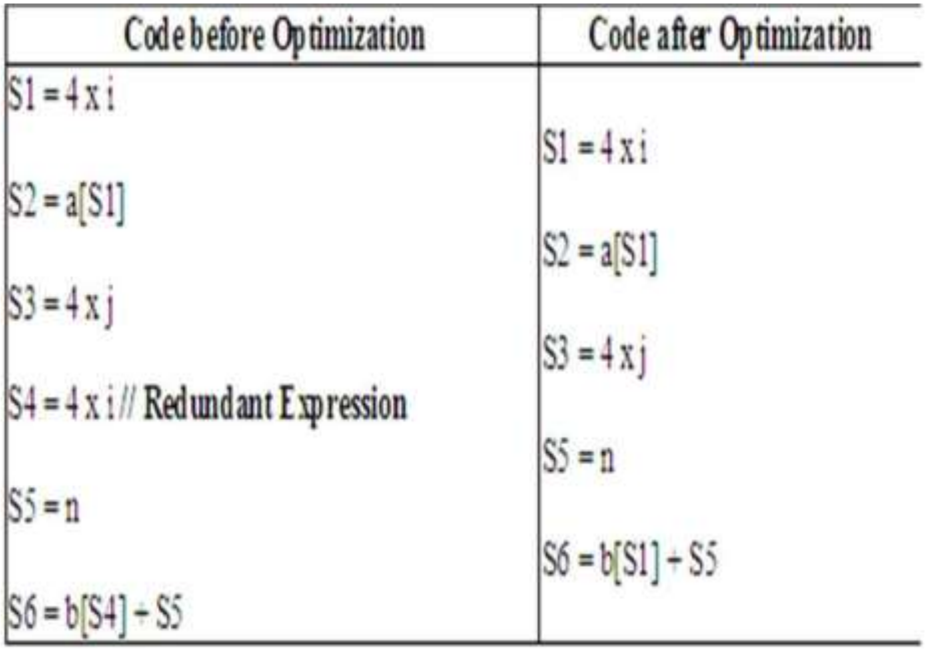
- Compile Time Evaluation
- Common sub-expression elimination
- Dead Code Elimination
- Code Movement
- Strength Reduction

Principles Sources of Optimization

- Compile Time Evaluation
- **A) Constant folding-**
- Circumference of circle = (22/7) x Diameter
- **Constant Propagation-**
- **Example-**
- pi = 3.14
- radius = 10
- Area of circle = pi x radius x radius

Principles Sources of Optimization

- **Common sub-expression**

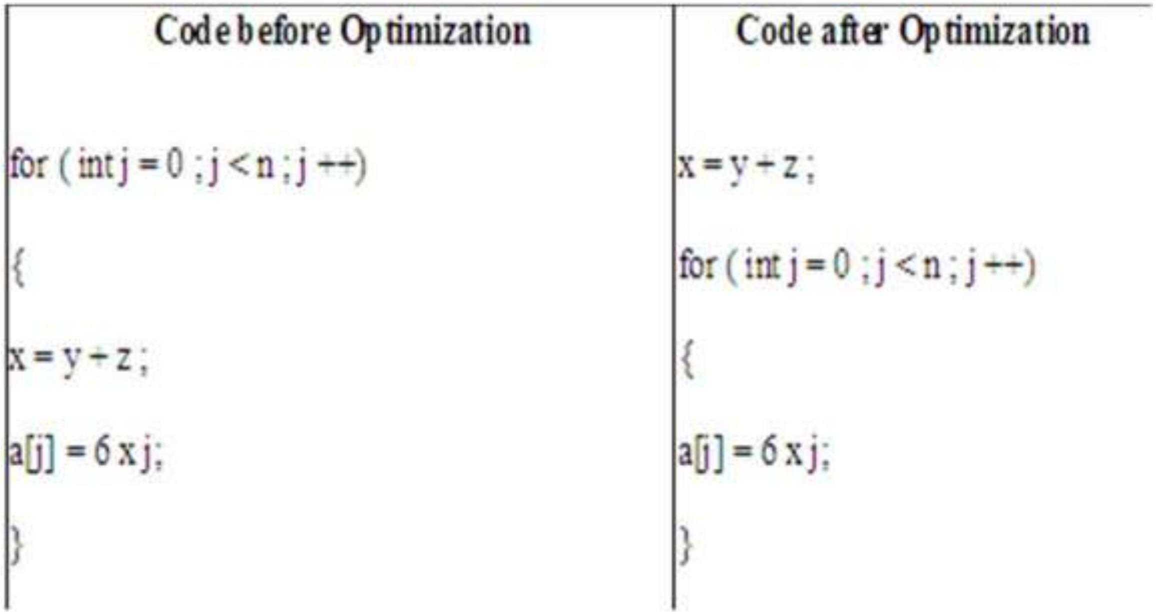


Code before Optimization	Code after Optimization

S1 = 4 x i
S2 = a[S1]
S3 = 4 x j
S4 = 4 x i // Redundant Expression
S5 = n
S6 = b[S4] + S5
	S1 = 4 x i
	S2 = a[S1]
	S3 = 4 x j
	S5 = n
	S6 = b[S1] + S5

Principles Sources of Optimization

- **Code Movement**



Code before Optimization	Code after Optimization

for (int j = 0 ; j < n ; j ++)
{
x = y + z ;
a[j] = 6 x j;
}
	x = y + z ;
	for (int j = 0 ; j < n ; j ++)
	{
	a[j] = 6 x j;
	}

Principles Sources of Optimization

- **Dead code elimination**

Code before Optimization

```
i = 0 ;
if (i == 1)
{
a = x + 5 ;
}
```

Code after Optimization

```
i = 0 ;
```

Principles Sources of Optimization

- **Strength reduction-**

Code before Optimization	Code after Optimization
B = A x 2	B = A + A

Summarization