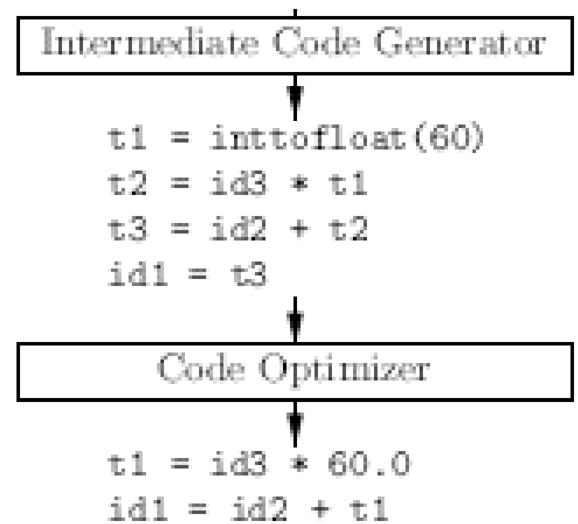
Unit V
Intermediate Code Optimization

- Program transformation technique
- Improves code – consume less resources
- Transforms the code to make it more efficient
- Output is not changed
- Intermediate code → optimization → code generation is made easier



Code Optimization

- Optimization
  - *Machine Independent Optimization*
  - takes in the intermediate code and transforms a part of the code that does not involve any CPU registers
  - 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:

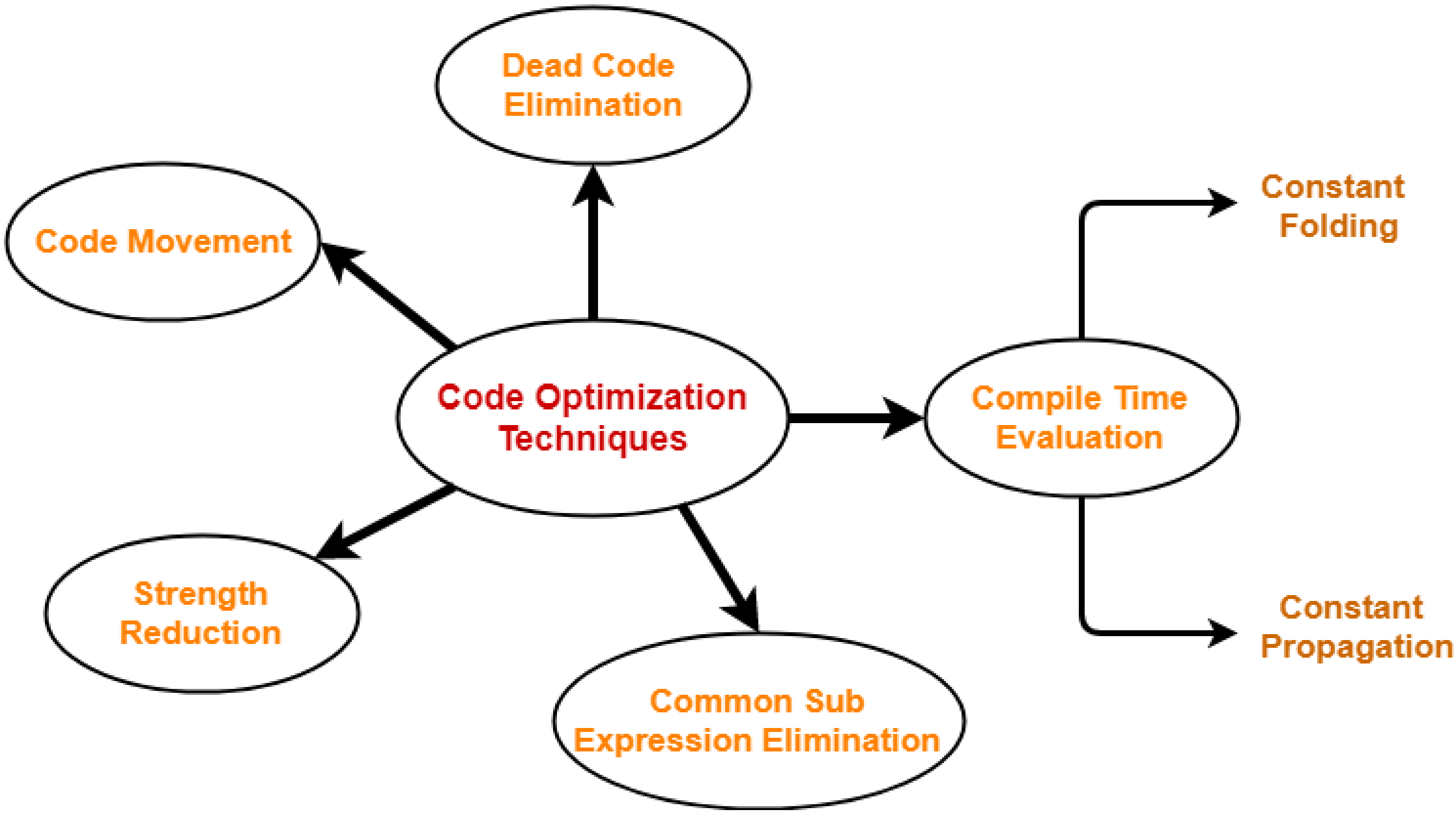
```
Item = 10;
do
{
   value = value + item;
} while(value<100);
```

  - *Machine Dependent optimization*
    - Target code
      - Rearrangement of machine instructions to improve the efficiency of the code
      - Divide the code into basic blocks

Peephole Optimization

- Optimization – eliminates the redundant instruction from a small area of code
- Set of code – peephole / window
- Goals :
  - Improves performance
  - Reduce memory footprint
  - Reduce code size

Principle sources of optimization



Compile Time Evaluation

- ***Constant Folding***
  - Folding the constants
  - The expressions that contain the operands having constant values at compile time are evaluated.
  - ***Example:***
  - return (3+5); → return 8;
  - Cir=(22/7)*diameter → cir = 3.14*diameter
- ***Constant Propagation***
  - If some variable has been assigned some constant value, then it replaces that variable with its constant value in the further program during compilation.
  - ***Example:***
    - radius =10,pi=3.14
    - area=pi*radius*radius; → area=3.14*10*10;

Common Sub Expression

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

Code Movement

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

Dead Code Elimination

- Eliminates the dead code

Code before Optimization

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

Code after Optimization

```
i = 0 ;
```

Strength Reduction

- Reduces the strength of expressions
- Replaces expensive operators with cheaper one
- ***<u>Example</u>***
  - B=A*2 →B=A+A
  - Cost of multiplication is higher than the addition