PEEPHOLE OPTIMIZATION

Peephole is a small, moving window on the target program.

Peephole optimization, a method for trying to improve the performance of the target program by taking short sequence of the program, apply transformations over it to produce the shorter and faster sequence.

Following are the program transformations that are the characteristics of peephole optimization

* redundant instruction elimination
* flow of control optimization
* algebraic simplification
* use of machine idioms

1. Redundant stores and loads

1. MOV R0, a
2. MOV a, R0

We can delete instruction (2) because whenever (2) is executed, it ensures that the value of a is already in the register R0.

2. Unreachable code Elimination

A statement which is never executed in the program is called unreachable code.

Ex1:
An unlabeled instruction following the unconditional statement is one example of unreachable code.

```
L1  …..
    …..
    goto L1
    printf(“hello”);
```

print(“hello”) statement is unreachable. We can remove that statement.

Ex2:

```
#define debug 0
…..
…..
```

```
if(debug)
{
printf("hello");
}
```

debug value is not updated after the definition statement and before the condition statement. So printf statement in the condition statement will never get executed. So we can remove the condition and printf statement.

3. Flow of control optimization

Intermediate code frequently produces jumps to jumps, jumps to conditional jumps and conditional jumps to jumps. These unnecessary jumps has to be eliminated.

Ex1: jumps to jumps

```
      goto L1
      …..
      …..
L1: goto L2
```

can be transformed into the following by avoiding unnecessary jump to L1.

```
      goto L2
      …..
      …..
L1: goto L2
```

Ex2: jumps to conditional jumps

```
      goto L1
      …..
      …..
L1: if a<b goto L2
```

can be transformed into the following if L1 is preceded by unconditional goto.

```
     if a<b goto L2
     goto L3
     ……..
     ……..
L3:
```

Ex3: conditional jumps to jumps

```
      if a<b goto L1
      …..
```

…..
L1: goto L2

can be transformed into the following by avoiding unnecessary jump to L1.

if a<b goto L2
…..
…..
L1: goto L2

4. Algebraic simplification

Algebraic simplification such as addition with 0 (x=x+0), multiplication by 1 (x=x*1) can be removed to optimize the target code.

5. Reduction in strength

Reduction in strength replaces the expensive operations by the equivalent cheaper ones on the target machine. For the computation of x^2, the operation x*x is cheaper than calling the function pow(x,2).

6. Use of machine idioms

Target machine have some specific instruction to implement certain operations efficiently. Detecting such situation and using that instruction reduces the execution time greatly. Instructions which uses the auto increment or auto decrement addressing modes for the operands may be used for some suitable example.