UNIT - V

UNIT – V
CODE GENERATION

PEEPHOLE OPTIMIZATION

- A statement-by-statement code-generations strategy often produce target code that contains redundant instructions and suboptimal constructs .The quality of such target code can be improved by applying "optimizing" transformations to the target program.

- A simple but effective technique for improving the target code is peephole optimization, a method for trying to improving the performance of the target program by examining a short sequence of target instructions (called the peephole) and replacing these instructions by a shorter or faster sequence, whenever possible.

- The peephole is a small, moving window on the target program. The code in the peephole need not contiguous, although some implementations do require this.it is characteristic of peephole optimization that each improvement may spawn opportunities for additional improvements.

- We shall give the following examples of program transformations that are characteristic of peephole optimizations:

- Redundant-instructions elimination
- Flow-of-control optimizations
- Algebraic simplifications
- Use of machine idioms
- Unreachable Code

Redundant Loads And Stores:

If we see the instructions sequence

(1) MOV R0,a
(2) MOV a,R0

we can delete instructions (2) because whenever (2) is executed. (1) will ensure that the value of **a** is already in register R0.If (2) had a label we could not be sure that (1) was always executed immediately before (2) and so we could not remove (2).

Unreachable Code:

- Another opportunity for peephole optimizations is the removal of unreachable instructions. An unlabeled instruction immediately following an unconditional jump may be removed. This operation can be repeated to eliminate a sequence of instructions. For example, for debugging purposes, a large program may have within it certain segments that are executed only if a variable **debug** is 1. In C, the source code might look like:

```
#define debug
0 ….
If ( debug ) {

Print debugging information

}
```

In the intermediate representations the if-statement may be translated as:

```
debug =1 goto L2

goto L2

L1: print debugging information

L2:.....(a)
```

- One obvious peephole optimization is to eliminate jumps over jumps .Thus no matter what the value of **debug**; (a) can be replaced by:

```
If debug ≠1 goto L2

Print debugging information

L2:.....(b)
```

- As the argument of the statement of (b) evaluates to a constant **true** it can be replaced by

```
If debug ≠0 goto L2

Print debugging information

L2:.....(c)
```

- As the argument of the first statement of (c) evaluates to a constant true, it can be replaced by goto L2. Then all the statement that print debugging aids are manifestly

unreachable and can be eliminated one at a time.

Flows-Of-Control Optimizations:

- The unnecessary jumps can be eliminated in either the intermediate code or the target code by the following types of peephole optimizations. We can replace the jump sequence

  goto L1
  ….
  L1: gotoL2 by the sequence
  goto L2
  ….
  L1: goto L2

- If there are now no jumps to L1, then it may be possible to eliminate the statement L1:goto L2 provided it is preceded by an unconditional jump .Similarly, the sequence

  if a < b goto L1
  ….
  L1: goto L2
  can be replaced by Ifa < b goto L2
  ….
  L1: goto L2

- Finally, suppose there is only one jump to L1 and L1 is preceded by an unconditional goto. Then the sequence

  goto L1

  ……..
  L1: if a <b goto L2

  L3:..... (1)

- Maybe replaced by Ifa<b goto L2

  goto L3
  …….