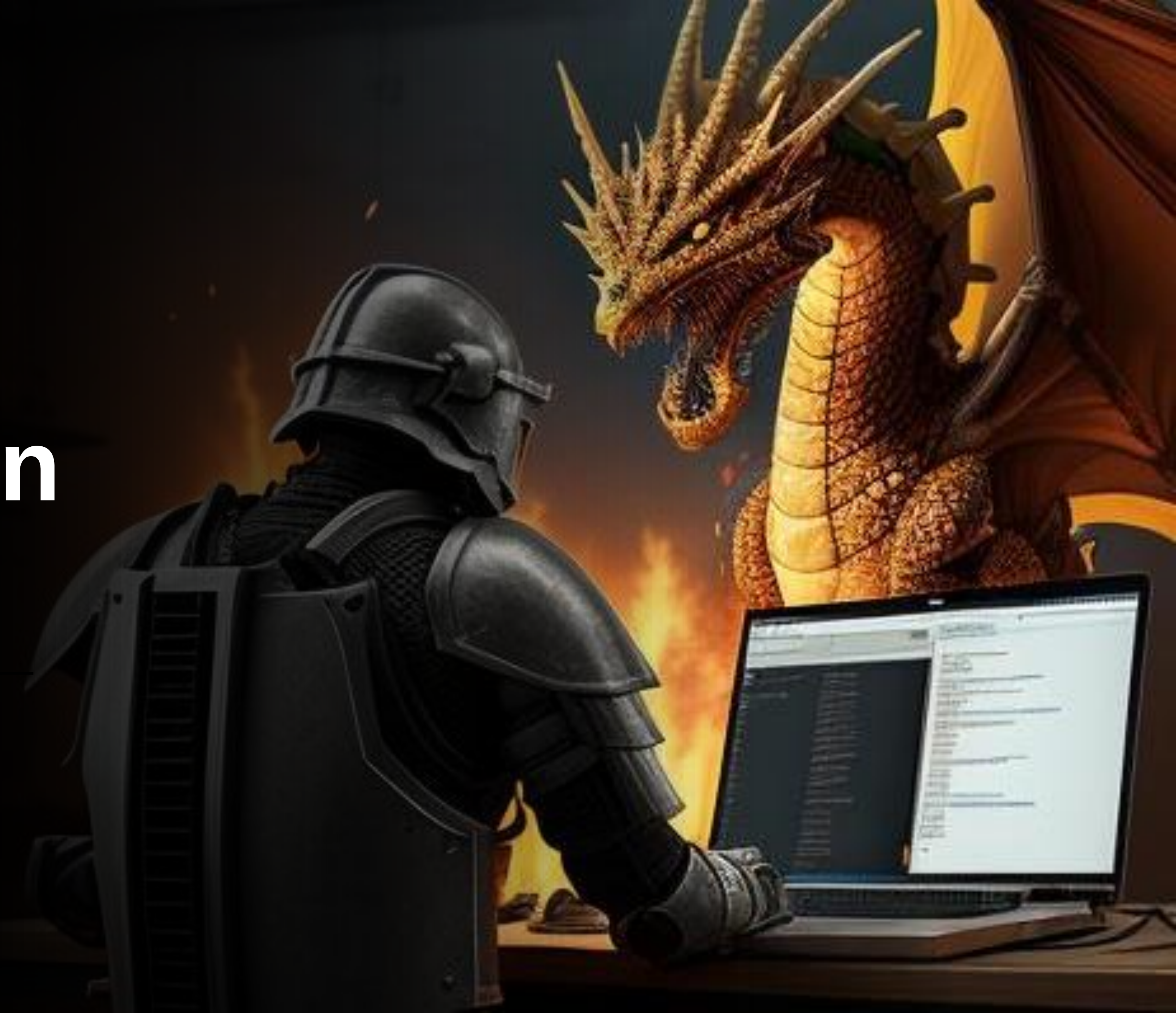
Peephole Optimization



Baivab Das

Lecturer

Department of CSE

University of Asia Pacific

Peephole Optimization

- A small moving window on the target program is a peephole
- We are not going to consider the entire code
- The peephole optimization replaces a short sequence of target instructions with a shorter or faster sequence
- Peephole optimization can be applied to both intermediate code and target machine code
- Peephole optimization is a machine-dependent optimization technique. This means that it is specific to the architecture of the machine on which the program will be executed.

Techniques

- Redundant Instruction Elimination
- Unreachable Code Elimination
- Flow of Control Optimization
- Algebraic Simplification
- Use of Machine Idioms

Redundant Instruction Elimination

```
1) MOV R0, x
2) MOV x, R0
```

- Here, the first one is a load instruction which loads the content of register R_0 into the memory location x
- Again, in the 2nd statement, we can observe that x is moved into R_0
- This is called as **redundant load and store** which is not necessary
- So, we can eliminate the 2nd statement as in the 1st statement copy operation is already performed

Redundant Instruction Elimination

```
    1) MOV R0, x
L1  2) MOV x, R0
    …   …
    …   …
    7) if x<0 GOTO L1
```

- But, if the redundant statement (here the 2nd statement) is preceded by any lable such as L1, L2 etc. then **we cannot eliminate that statement**
- As we cannot guarantee that the statement 1 is always executed before the 2nd statement
- Also, the 2nd statement can be called as some other statement also
- So, if the statement is preceded by any label, we cannot eliminate that statement

Example: Redundant Instruction Elimination

```
a=b+c
d=a+e
```

```
MOV b, R0
ADD c, R0
MOV R0,a
MOV a,R0
ADD e, R0
MOV R0,d
```

Code with redundant load and store

```
MOV b, R0
ADD c, R0
MOV R0,a
ADD e, R0
MOV R0,d
```

Code after Peephole Optimization

Unreachable/ Dead Code Elimination

```
x = 0;
if(x==1)
{
    a=b;
}
```

- The above C code snippet shows the example of unreachable code
- As the value of x is already defined, the if block will never execute. It will always become false
- So, the whole if block is unreachable

Intermediate Code for the Unreachable/ Dead Code

```
x=0
if x=1 GOTO L1
GOTO L2
L1: a=b
L2: (false part)
```

- Here, jump to L1 will never be performed
- Also, the false part is always getting executed
- So, **jump over jump** is getting performed here

```
x=0
if x!=1 GOTO L2
a=b
L2: (false part)
```

- The optimized intermediate code contains only 1 jump statement
- Instead of checking if `x=1`, it checks if x *not equals to* 1
- If x=1, then only it will execute **a=b**
- But we can observe that as **x=0** is already defined, the a=b statement will never be executed
- The false part (L2) is always executed
- So, we can simply **eliminate** the a=b statement

Flow of Control Optimization

```
    1) if a>b GOTO L1
    2) ……
L1: 3) GOTO L2
    4) ……
L2: 5) a=b
```

```
    1) if a>b GOTO L2
    2) ……
L1: 3) GOTO L2
    4) ……
L2: 5) a=b
```

- IC normally generates jumps to jumps
- We can use the peephole optimization to eliminate these unnecessary jumps
- In the example above, we can observe that the jump to L1 is unnecessary as it jumps to L2
- We can apply peephole optimization and directly make the jump to L2
- We can simply remove the L1 jump statement

Flow of Control Optimization

- But we need to keep in mind that if the statement is a **target of another Jump statement,** then we cannot remove it!

```
    1) if a>b GOTO L2
    2) ……
L1: 3) GOTO L2
    4) ……
L2: 5) a=b
    6) if a>b GOTO L1
```

Flow of Control Optimization: Conditional Jump Example

```
   1) if a<b GOTO L1
   …  ……
L1:5) GOTO L2
L2:…  …………
```

- Here, jump to L1 is performed and L1 again jumps to L2
- We can optimize by removing the L1

```
   1) if a<b GOTO L2
   …  ……
```
~~L1:3) GOTO L2~~
```
L2:…  …………
```

- We can simply remove the L1 statement
- But we need to keep in mind that if L1 is the target of another statement, we cannot remove it (see the previous example)

Algebraic Simplification

```
x=x+0
x=x*1
```

- In both of the statements, value of x is never changed in despite of the operations performed
- So, these redundant statements can be removed from the IC

Algebraic Simplification: Reduction in Strength

```
x^2 = x*x
2*x = x+x
x*2 = x<<1
x/2 = x>>1
```

x*2	=	x<<1
x*4	=	x<<2
x*8	=	x<<3
x/8	=	x>>3

- Reduction in strength means complex operations must be replaced by the simpler one
- We can also say, expensive operations must be replaced by the cheaper operations
- For example, `x^2` (exponent) requires a separate `pow(x,y)` function in C. Instead of doing that, we can perform the multiplication which is simpler than using a function
- Similarly, we can convert the multiplication by power of 2 into left shift. If `x*4` is needed to be performer, we can left shift twice as `x<<2`

Use of Machine Idioms

- Hardware instructions can be used for reducing the execution time
- To perform the addition operation `i=i+1`, we need 3 target instructions. First, we move the first i to a register, then we add 1 to that register. Finally, the content is moved to i.
- Instead of writing 3 target machine instruction, we can simply use the hardware `INC` (increment) instruction

```
i=i+1 → INC i
i=i-1 → DEC i
```