Optimization Overview

inefficient code

- You may have noticed that the code you are generating in your project seems very inefficient
- Lots of **redundant computation**

```
LA T1 <address of x>
LW T2 0(T1)
LA T3 <address of x>
LW T4 0(T3)
ADD T5 T2 T4
```

instead of

```
LA T1 <address of x>
LW T2 0(T1)
LW T4 0(T1)
ADD T5 T2 T4
```

inefficient code

- You may have noticed that the code you are generating in your project seems very inefficient

- Lots of **redundant computation**

```
LA T1 <address of x>
LW T2 0(T1)
LA T3 <address of x>
LW T4 0(T3)
ADD T5 T2 T4
```

instead of

```
LA T1 <address of x>
LW T2 0(T1)
LW T4 0(T1)
ADD T5 T2 T2
```

inefficient code

- You may have noticed that the code you are generating in your project seems very inefficient
- Lots of **instruction choice**

```
LI T1 10
LW T2 8(FP)
ADD T3 T1 T2
SW T3 -4(FP)
```

instead of

```
LW T2 8(FP)
ADDI T3 T2 10
SW T3 -4(FP)
```

inefficient code

- You may have noticed that the code you are generating in your project seems very inefficient
- Lots of **unnecessary loads and stores**

```
LA T1 <address of x>
LI T2, 10
SW T2, 0(T1)
LW T3, 0(T1)
ADDI T4, T3, 20
SW T4, 0(T1)
```

instead of

```
LA T1 <address of x>
LI T2, 10
ADDI T4, T2, 20
SW T4, 0(T1)
```

inefficient code

But code inefficiency goes beyond just small sequences of instructions

- What about redundant computation happening inside a loop?
- What about expensive operations happening inside a loop?
- What about code that access memory in a way that is bad for caching?

optimization

- Compilers can generate correct code relatively easily
- But generating *efficient* code is much harder
  - This is where a lot of programming languages research happens

- Can happen at multiple levels
  - At the source code level or AST level (e.g., restructure loops for better performance)
  - At the assembly level (e.g., replace some sequences of instructions with more efficient sequences)
  - At the **intermediate representation** level (e.g., remove redundant instructions)

intermediate representation

- We have already worked with one intermediate representation: abstract syntax trees
- Many compilers have another, lower-level intermediate representation that facilitates optimization
  - Closer to assembly, but no machine specific instructions, registers, etc.
  - Examples: LLVM bitcode, C# CIL, Java bytecode