

CS443: Compiler Construction

Lecture 22: Instruction Selection

Stefan Muller

Easy but inefficient way to compile instructions:
move all operands to regs, then perform op.

```
%d = add i32 v1, v2
```

- For each operand:
 - If in register, keep
 - If in memory or a value, load into temp (x5, x6)
- If the destination is a register, do nothing
- If the destination is in memory, store the result in a temp (x7), then move
 - If a register, do nothing
 - If in memory, m

Easy but inefficient way to compile instructions:
move all operands to regs, then perform op.

```
%d = add i32 %v, 42
```

d -> stack 1, v -> a0

```
addi t0, zero, 42  
add t2, a0, t0  
sw t2, -4(fp)
```

- For each operand:
 - If in register, keep
 - If in memory or a value, load into temp ($x5=t0$, $x6=t1$)
- If the destination is a register, do nothing
- If the destination is in memory, store the result in a temp ($x7=t2$), then move

More efficient to special-case, especially
when immediates are involved

```
%d = add i32 %v, 42
```

```
addi t0, zero, 42
add t2, a0, t0
sw t2, -4(fp)
```

```
addi t2, a0, 42
sw t2, -4(fp)
```

Recall: Maximal Munch

Choose RISC-V Pattern that captures the most LLVM instructions

```
%t3 = icmp lt i32 %a0, %a1  
br i1 %x1, label ltrue, label lfalse
```

```
blt a0, a1, ltrue  
j lfalse
```

(Assuming we don't
need %t3 later)

Can also compile `icmp` directly

<code>%t2 = icmp lt i32 %t0, %t1</code>	<code>slt t2, t0, t1</code>
<code>%t2 = icmp gt i32 %t0, %t1</code>	<code>slt t2, t1, t0</code>
<code>%t2 = icmp eq i32 %t0, %t1</code>	<code>xor t2, t0, t1</code>
	<code>sltiu t2, t2, 1</code>
<code>%t2 = icmp le i32 %t0, %t1</code>	<code>slt t2, t0, t1</code>
	<code>xori t2, t1, 1</code>
<code>%t2 = icmp ge i32 %t0, %t1</code>	<code>slt t2, t1, t0</code>
	<code>xori t2, t1, 1</code>
<code>%t2 = icmp ne i32 %t0, %t1</code>	<code>xor t2, t0, t1</code>
	<code>sltiu t2, t2, 1</code>
	<code>xori t2, t1, 1</code>

getelementptr: a lot of math

```
%d = getelementptr ty, ty* %ptr, ty1 v1, ..., tyN vN
```

```
d = p + f(v1);
```

```
d += f(v2);
```

```
..
```

```
d += f(vN)
```

getelementptr: a lot of math

```
%d = getelementptr ty, ty* %ptr, ty1 v1, ..., tyN vN
```

For arrays of type ty': $d += v1 * \text{sizeof}(ty')$

```
addi t1, x0, sizeof(ty')
```

```
mul t1, t1, rv1
```

```
add rd, rd, t1
```

May have to bring in one index
at a time from memory since
we need t1

getelementptr example

```
%Tstruct = {i32 i32}  
%d = getelementptr %Tstruct, %Tstruct* %ptr,  
           i32 %i, i32 1
```

```
lw t2, 4(sp)    # t2 = ptr  
lw t0, 8(sp)    # t0 = i  
addi t1, x0, 8 # t1 = 8  
mul t0, t0, t1 # t0 = i * 8  
add t2, t2, t0 # t2 += i * 8  
addi t0, x0, 4 # t0 = 4 - prev. t0 no longer live  
add t2, t2, t0 # t2 += 4
```

ptr: 4(sp)
i: 8(sp)

alloca: Decrement sp, return new sp

```
%rd = alloca ty, i32 n
```

```
subi sp, sp, n * sizeof(ty)  
add rd, x0, sp
```

Example

startloop:

```
%last = alloc i32
store i32 %n, i32* %last
%temp = load i32, i32* %last
%tst = icmp le i32 %temp, 1
br i1 %tst, label %end, label %body
```

body:

```
%rem2 = urem i32 %temp, 2
%tst2 = icmp eq i32 %rem2, 0
br i1 %tst2, label %even, label %odd
```

even:

```
%n = udiv i32, %temp, 2
br startloop
```

odd:

```
%temp2 = mul i32 %n, 3
%n = add i32 %temp2, 1
br startloop
```