Week 7.2

Tuesday, May 14, 2024 6:02 PM

x86-64

inst. operands : immediate, registers, memory locations

notation: "imm" immediate

R register

[R] value inside R

<x> value at memory location x

OPERAND	VALUE
$imm	constant
imm	<imm>
R	[R]
imm(R)	<imm + [R]>

inst

'q' suffix

IR optimization

LIR optimization
mem. mgmt
register allocation

---

optimization

- reduce the amount of computation
- need to guarantee that the optimized program "does the same thing"

optimization = analysis + transformation

this class

assume no pointers, structs, globals, or calls

$copy $arith $cmp $jump $branch $ret

Safety & profitability

|

preserve behavior as user-observable events
independent of time

[illegible] behavior
independent of time

"undefined behavior"

```
#include <limits.h>
#include <stdio.h>

int main() {
    printf("%d\n", (INT_MAX+1) < 0);
    printf("%d\n", (INT_MAX+1) < 0);
    return 0
}
```

```
gcc test.c -o test
./test
1   0   0   1
1   0   1   0
```

```
int dumb(int a) { return (a+1) > a; }
```

Case 1: a is not INT_MAX

↳ answer = 1

Case 2: a is INT_MAX

↳ answer = 1

```
int dumb(int a) { return 1;}
```

-Wall

---

function inlining

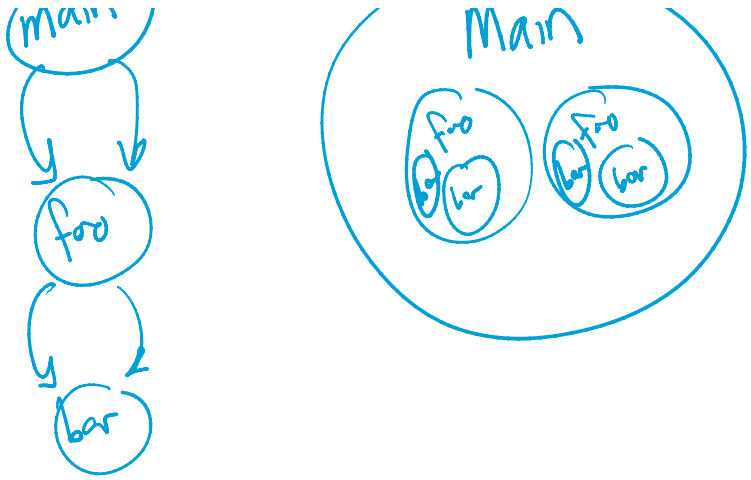
```
fn foo(a:int)→ int { return a + 1;}

fn main() → int {
    let x:int = foo(2);
    return x;
}
```

"inline"

```
fn main() → int {
    let x:int = 2 + 1;
    return x;
}
```

```
fn main()→ int { return 3;}
```



---

local opts

(per basic block)

constant folding & arith. identities

- if all operands are constants, evaluate
- if some operands are constants, apply arithmetic identity

local value numbering