Compiler Design 1

Lesson 4

Magnus Lång
Kim-Ahn Tran
14th December 2017
Outline

Parameter passing
  Exercise 1

Activation Records and Scoping
  Exercise 2

Register Allocation
  Exercise 5

Local Optimisation
  Exercise 4
Parameter passing
Exercise 1

```c
int foo(int a) {
    int b = a++;
    return b * a;
}

int bar(int b) {
    return foo(b);
}

int main() {
    int c;
    c = 6;
    return bar(c);
}
```

For $a$ and $b$ in all combinations of *by value* and *by reference*:

What’s in the argument slots of the activation records...

- In the call to `bar` on line 13?

<table>
<thead>
<tr>
<th></th>
<th>$a$ b/v</th>
<th>$a$ b/r</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$</td>
<td>b b/v</td>
<td>val 6</td>
</tr>
<tr>
<td></td>
<td>b b/r</td>
<td>ptr to main’s c</td>
</tr>
</tbody>
</table>

- In the call to `foo` on line 7?

<table>
<thead>
<tr>
<th></th>
<th>$a$ b/v</th>
<th>$a$ b/r</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$</td>
<td>b b/v</td>
<td>val 6</td>
</tr>
<tr>
<td></td>
<td>b b/r</td>
<td>ptr to main’s c</td>
</tr>
<tr>
<td>$b$</td>
<td>val 6</td>
<td>ptr to main’s c</td>
</tr>
</tbody>
</table>

(assume the compiler allocates all the variables on the stack)
Exercise 1

```c
int foo(int a) {
    int b = a++;
    return b * a
}

int bar(int b) {
    return foo(b);
}

int main() {
    int c;
    c = 6;
    return bar(c);
}
```

For `a` and `b` in all combinations of *by value* and *by reference*:

What’s assembly is generated to retrieve `a`’s value on line 2?

- When `a` is call *by value*, it is directly loaded:
  ```assembly
  lw $a0, OFFSET($FP)
  ```
- When `a` is call *by value*, the pointer in the activation record is also dereferenced:
  ```assembly
  lw $t0, OFFSET($FP)
  lw $a0, 0($t0)
  ```

(assume the compiler allocates all the variables on the stack)
Activation Records and Scoping
Stack layout

- previous frame
- parameters
- controll link
- access link
- return address
- local vars

FP

SP
Exercise 2

```c
int spam(int a, int b) {
    int egg(int x) {
        return x*a mod b;
    }
    return egg(3);
}
```

- What does spam(7, 13) return?
- What does the stack contain just before egg returns?
- Where are \( a \) and \( x \) in the frame?
- What code is generated to access them?

# x is loaded directly
```
lw t0, -4(fp)
```

# a uses the access link
```
lw t0, -c(fp)
lw t0, -8(t0)
```
Register Allocation
Exercise 5

L0:  e := 0
    b := 1
    d := 2
L1:  a := b + 2
    c := d + 5
    e := e + c
    f := a * a
    if f < c goto L3
L2:  e := e + f
    goto L4
L3:  e := e + 2
L4:  d := d + 4
    b := b - 4
    if b != d goto L1
L5:

- Draw the control-flow graph
- Compute sets of live vars between all statements. Only e is live at L5.
- Draw the interference graph
Local Optimisation
Exercise 4

- Apply, in order
  1. algebraic simplification
  2. common sub-expression elimination
  3. copy propagation
  4. constant folding
  5. dead code elimination

- What optimisations, in what order, can optimise this further?
Any questions?

We’re done!

Good luck with the assignment.
Code Generation
cond

<p1> => <e1>;  
<p2> => <e2>;  
...  
<pn> => <en>;  
1 => <en{n+1}>  

dnoc

- Write a code generation function:
cgen(cond <p1> => <e1>; ...; <pn> => <en>; 1 => <en{n+1}> dnoc) for this conditional expression.