Outline

Parameter passing
   Exercise 1

Register Allocation
   Exercise 4

Local Optimisation
   Exercise 3
Parameter passing
Exercise 1

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

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

```c
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>a b/v</th>
<th>a b/r</th>
</tr>
</thead>
<tbody>
<tr>
<td>b b/v</td>
<td>val 6</td>
</tr>
<tr>
<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>a b/v</th>
<th>a b/r</th>
</tr>
</thead>
<tbody>
<tr>
<td>b b/v</td>
<td>val 6</td>
</tr>
<tr>
<td>b b/r</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 reference*, 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)
Register Allocation
Exercise 4

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 3

- 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?

\[
\begin{align*}
a & := b + c \\
z & := a \times 2 \\
x & := 0 \times b \\
y & := b + c \\
w & := y \times y \\
u & := x + 3 \\
v & := u + w
\end{align*}
\]
Any questions?

We’re done!

Good luck with the assignment.
Code Generation
Exercise 2

Write a code generation function:
\[
cgen(\text{cond } \langle p_1 \rangle \Rightarrow \langle e_1 \rangle; \ldots; \langle p_n \rangle \Rightarrow \langle e_n \rangle; 1 \Rightarrow \langle e_{n+1} \rangle \text{ dnoc})
\]
for this conditional expression.