Register Allocation, i

Overview & spilling
\[ p := ((i \ldots) \ (\text{label} \ i \ \ldots) \ \ldots) \\
\text{i} := (x \leftarrow s) \\
\quad | (x \leftarrow (\text{mem} \ s \ n4)) \\
\quad | ((\text{mem} \ x \ n4) \leftarrow s) \\
\quad | (x \ \text{aop} = s) \\
\quad | (x \ \text{sop} = sx) \\
\quad | (x \ \text{sop} = \text{num}) \\
\quad | (cx \leftarrow s \ \text{cmp} \ s) \\
\quad | \text{label} \\
\quad | (\text{goto} \ \text{label}) \\
\quad | (\text{cjump} \ s \ \text{cmp} \ s \ \text{label} \ \text{label}) \\
\quad | (\text{call} \ s) \\
\quad | (\text{tail-call} \ s) \\
\quad | (\text{return}) \\
\quad | (eax \leftarrow (\text{print} \ s)) \\
\quad | (eax \leftarrow (\text{allocate} \ s \ s)) \\
\quad | (eax \leftarrow (\text{array-error} \ s \ s)) \\
\]

\[ \text{aop} ::= + | - | * | \& = \]
\[ \text{sop} ::= <\langle | \rangle\rangle = \]
\[ \text{cmp} ::= < | \leq | = \]

\[ s ::= x \ | \text{num} \ | \text{label} \]

\[ x, y ::= cx \ | \text{esi} \ | \text{edi} \ | \text{ebp} \ | \text{esp} \]
\[ cx ::= eax \ | \text{ecx} \ | \text{edx} \ | \text{ebx} \]
\[ sx ::= ecx \]
L2

\[
p := ((i \ldots) \ (\text{label } i \ldots) \ldots) \\
(\text{i ::= } (x <- s) \\
\quad | (x <- \ (\text{mem } s \ n4)) \\
\quad | ((\text{mem } x \ n4) <- s) \\
\quad | (x \ \text{aop= } s) \\
\quad | (x \ \text{sop=} \ sx) \\
\quad | (x \ \text{sop=} \ \text{num}) \\
\quad | (cx <- s \ \text{cmp } s) \\
\quad | \text{label} \\
\quad | (\text{goto } \text{label}) \\
\quad | (\text{cjump } s \ \text{cmp } s \ \text{label} \ \text{label}) \\
\quad | (\text{call } s) \\
\quad | (\text{tail-call } s) \\
\quad | (\text{return}) \\
\quad | (eax <- (\text{print } s)) \\
\quad | (eax <- (\text{allocate } s \ s)) \\
\quad | (eax <- (\text{array-error } s \ s))
\]

\[
aop ::= + \ | - \ | * \ | & \\
sop ::= \ll \ | \gg \\
cmp ::= < \ | \leq \ | = \\
s ::= x \ | \text{num} \ | \text{label}
\]

\[
x, y ::= \text{any-variable-at-all} \ | \ \text{reg} \\
cx ::= \text{any-variable-at-all} \ | \ \text{reg} \\
sx ::= \text{any-variable-at-all} \ | \ \text{reg} \\
\text{reg ::= eax} \ | \ \text{ecx} \ | \ \text{edx} \ | \ \text{ebx} \ | \ \text{esi} \ | \ \text{edi} \ | \ \text{ebp} \ | \ \text{esp}
\]
L2 semantics: variables

L2 behaves just like L1, except that non-reg variables are function local, e.g.,

\[
\begin{align*}
\text{(define } \ (f \ x) \ &\Rightarrow \ (\text{; :main}) \\
&\quad (+ (g \ x) \ 1)) \\
\text{(define } \ (g \ x) \ &\Rightarrow \ (\text{:f } (\text{temp }<- \ 1) \\
&\quad (\text{call } :g) \\
&\quad (\text{eax }+= \text{ temp}) \\
&\quad (\text{return})) \\
(f \ 10) \ &\Rightarrow \ (\text{:g } (\text{temp }<- \ 2) \\
&\quad (\text{eax }+= \text{ temp}) \\
&\quad (\text{return}) ))
\end{align*}
\]

The assignment to \texttt{temp} in \texttt{g} does not break \texttt{f}, but if \texttt{temp} were a register, it would.
L2 semantics: esp & ebp

L2 programs must use neither esp nor ebp. They are in L2 to facilitate register allocation only, not for the L3 \( \rightarrow \) L2 compiler’s use.
From L2 to L1

Register allocation, in three parts; for each function body we do:

• **Liveness analysis** ⇒ interference graph (nodes are variables; edges indicate “cannot be in the same register”)

• **Graph coloring** ⇒ register assignments

• **Spilling:** coping with too few registers

• Bonus part, **coalescing** eliminating redundant \((x \leftarrow y)\) instructions
Example Function

\[ \text{int } f(\text{int } x) = 2x^2 + 3x + 4 \]

: \text{f}
  
  (x2 <- eax)
  (x2 *= x2)
  (2x2 <- x2)
  (2x2 *= 2)
  (3x <- eax)
  (3x *= 3)
  (eax <- 2x2)
  (eax += 3x)
  (eax += 4)
  (return)
Example Function: live ranges

```c
int f(int x) = 2x^2 + 3x + 4

2x2 3x x2

: f
(x2 <- eax)
(x2 *= x2)
(2x2 <- x2)
(2x2 *= 2)
(3x <- eax)
(3x *= 3)
(eax <- 2x2)
(eax += 3x)
(eax += 4)
(return)
```
Example Function: live ranges

```
int f(int x) = 2x^2 + 3x + 4
```

```
: f
(x2 <- eax)
(x2 *= x2)
(2x2 <- x2)
(2x2 *= 2)
(3x <- eax)
(3x *= 3)
(eax <- 2x2)
(eax += 3x)
(eax += 4)
(return)
```
Example Function 2

```
int f(int x) = 4x (in a bad compiler)

:f
(a <- eax)
(b <- eax)
(c <- eax)
(d <- eax)
(eax <- a)
(eax += b)
(eax += c)
(eax += d)
(return)
```
No way to get all of \texttt{a, b, c, and d} into their own registers; so we need to \textit{spill} one of them.
Spilling

**Spilling** is a program rewrite to make it easier to allocate registers

- Pick a variable and a location on the stack for it
- Replace all writes to the variable with writes to the stack
- Replace all reads from the variable with reads from the stack

Sometimes that means introducing new temporaries
Spilling Example

Say we want to spill \( a \) to the location \((\text{mem ebp } -4)\).

Two easy cases:

\[
(a \leftarrow 1) \Rightarrow ((\text{mem ebp } -4) \leftarrow 1)
\]

\[
(x \leftarrow a) \Rightarrow (x \leftarrow (\text{mem ebp } -4))
\]
Example Function 2, need to spill

\[ \text{int} \ f(\text{int} \ x) = 4x \] (in a bad compiler)

:a b c d eax ebx ecx edi edx esi

::f
:(a <- eax)
:(b <- eax)
:(c <- eax)
:(d <- eax)
:(eax <- a)
:(eax += b)
:(eax += c)
:(eax += d)
:(return)
Example Function 2, spilling a

int f(int x) = 4x (in a bad compiler)

: f
((mem ebp -4) <- eax)
(b <- eax)
(c <- eax)
(d <- eax)
(eax <- (mem ebp -4))
(eax += b)
(eax += c)
(eax += d)
(return)
A trickier case:

\[(a \neq a) \Rightarrow (a_{\text{new}} \leftarrow (\text{mem ebp} - 4))\]
\[(a_{\text{new}} \neq a_{\text{new}})\]
\[((\text{mem ebp} - 4) \leftarrow a_{\text{new}})\]

In general, make up a new temporary for each instruction that uses the variable to be spilled

This makes for very short live ranges.
Example Function 2, spilling b

```plaintext
int f(int x) = 4x (in a bad compiler)

:f
(a <- eax)
((mem ebp -4) <- eax)
(c <- eax)
(d <- eax)
(eax <- a)
(s0 <- (mem ebp -4))
(eax += s0)
(eax += c)
(eax += d)
(return)
```
Example Function 2, spilling b

Even though we still have four temporaries, we can still allocate them to our three unused registers because the live ranges of \( s0 \) and \( a \) don’t overlap and so they can go into the same register.
Your job

Implement:

\[
\text{spill : (i ...) ;; original function}
\]
\[
\text{var ;; to spill}
\]
\[
\text{offset ;; multiple of 4}
\]
\[
\text{var ;; prefix for temporaries}
\]
\[
-> \text{(i ...)} ;; spilled version
\]