Register Allocation, i

Overview & spilling
\( p ::= ( (i \ldots) \ (\text{label} \ i \ldots) \ \ldots) \)
\( i ::= (x <- s) \)
\( \ | (x <- (\text{mem} \ x \ n4)) \)
\( \ | ((\text{mem} \ x \ n4) <- s) \)
\( \ | (x \ \text{aop} = t) \)
\( \ | (x \ \text{sop} = sx) \)
\( \ | (x \ \text{sop} = \text{num}) \)
\( \ | (cx <- t \ \text{cmp} \ t) \)
\( \ | \text{label} \)
\( \ | (\text{goto} \ \text{label}) \)
\( \ | (\text{cjump} \ t \ \text{cmp} \ t \ \text{label} \ \text{label}) \)
\( \ | (\text{call} \ u) \)
\( \ | (\text{tail-call} \ u) \)
\( \ | (\text{return}) \)
\( \ | (eax <- (\text{print} \ t)) \)
\( \ | (eax <- (\text{allocate} \ t \ t)) \)
\( \ | (eax <- (\text{array-error} \ t \ t)) \)
\( \ \text{aop} ::= += | -= | *= | &= \)
\( \ \text{sop} ::= <<= | >>= \)
\( \ \text{cmp} ::= < | <= | = \)
\( \ \text{s} ::= x | \text{num} | \text{label} \)
\( \ \text{t} ::= x | \text{num} \)
\( \ \text{u} ::= x | \text{label} \)
\( \ \text{x, y} ::= \text{cx} | \text{esi} | \text{edi} | \text{ebp} | \text{esp} \)
\( \ \text{cx} ::= \text{eax} | \text{ecx} | \text{edx} | \text{ebx} \)
\( \ \text{sx} ::= \text{ecx} \)
\[ p := ((i \ldots) \ (label \ i \ldots) \ldots) \]
\[ i := (x <- s) \]
\[ \mid (x <- (mem \ x \ n4)) \]
\[ \mid ((mem \ x \ n4) <- s) \]
\[ \mid (x aop= t) \]
\[ \mid (x sop= sx) \]
\[ \mid (x sop= num) \]
\[ \mid (cx <- t cmp t) \]
\[ \mid label \]
\[ \mid (goto \ label) \]
\[ \mid (cjump t cmp t label label) \]
\[ \mid (call \ u) \]
\[ \mid (tail-call \ u) \]
\[ \mid (return) \]
\[ \mid (eax <- (print t)) \]
\[ \mid (eax <- (allocate \ t \ t)) \]
\[ \mid (eax <- (array-error \ t \ t)) \]
\[ aop:=::=\oplus | -\mid \star | \&= \]
\[ sop:=::=\langle\langle | >>= \]
\[ cmp:=::=\prec | \preceq | = \]
\[ s::=x | num | label \]
\[ t::=x | num \]
\[ u::=x | label \]
\[ x, y::=cx | esi | edi | ebp | esp \]
\[ cx::=eax | ecx | edx | ebx | var \]
\[ sx::=ecx | var \]
\[ var::=variable \ matching \ regexp \ ^{a-zA-Z_} [a-zA-Z0-9-]*$ \]
\[ \text{except \ registers \ and \ keywords \ (e.g., \ print, \ call, \ cjump, \ ...)} \]
L2 semantics: variables

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

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

The assignment to \textbf{temp} in \textbf{g} does not break \textbf{f}, but if \textbf{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 → 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

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

:f
(x2 <- eax)
(x2 *= x2)
(dx2 <- x2)
(dx2 *= 2)
(tx <- eax)
(tx *= 3)
(eax <- dx2)
(eax += tx)
(eax += 4)
(return)
Example Function: live ranges

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

dx2 tx x2

: f
  (x2 <- eax)
  (x2 *= x2)
  (dx2 <- x2)
  (dx2 *= 2)
  (tx <- eax)
  (tx *= 3)
  (eax <- dx2)
  (eax += tx)
  (eax += 4)
  (return)
Example Function: live ranges

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

\[
:\text{f} \\
(\text{x2} \leftarrow \text{eax}) \\
(\text{x2} \ast= \text{x2}) \\
(\text{dx2} \leftarrow \text{x2}) \\
(\text{dx2} \ast= 2) \\
(\text{tx} \leftarrow \text{eax}) \\
(\text{tx} \ast= 3) \\
(\text{eax} \leftarrow \text{dx2}) \\
(\text{eax} += \text{tx}) \\
(\text{eax} += 4) \\
(\text{return})
\]
Example Function 2

\[
\text{int } f(\text{int } x) = x + x + x + x \quad \text{(in a stupid compiler)}
\]

:a b c d eax ebx ecx edi edx esi

:\text{f} \\
(a \leftarrow \text{eax}) \\
(b \leftarrow \text{eax}) \\
(c \leftarrow \text{eax}) \\
(d \leftarrow \text{eax}) \\
(eax \leftarrow a) \\
(eax += b) \\
(eax += c) \\
(eax += d) \\
(\text{return})
No way to get all of a, b, c, and d into their own registers; so we need to *spill* one of them.
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 \texttt{a} to the location \texttt{(mem ebp -4)}. Two easy cases:

\[
\begin{align*}
(a &< -1) \Rightarrow ((\text{mem ebp } -4) &< -1) \\
(x &< -a) \Rightarrow (x &< (\text{mem ebp } -4))
\end{align*}
\]
Example Function 2, need to spill

\[
\text{int } f(\text{int } x) = x + x + x + x \quad \text{(in a stupid compiler)}
\]

: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

```plaintext
int f(int x) = x+x+x+x (in a stupid compiler)

:f
  ((mem ebp -4) <- eax)
  (b <- eax)
  (c <- eax)
  (d <- eax)
  (eax <- (mem ebp -4))
  (eax += b)
  (eax += c)
  (eax += d)
  (return)
```
Spilling Example

A trickier case:

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

\[(a_{\text{new}} \* a_{\text{new}})\]

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

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) = x+x+x+x  (in a stupid 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:

```
spill : (i ...) ;; original function
  var ;; to spill
  offset ;; multiple of 4
  var ;; prefix for temporaries
-> (i ...) ;; spilled version
```
Here’s how to two example spilled functions from the earlier slides would look like as calls to spill:

\[
  \text{(spill «the original program»}
  \text{ 'a}
  \text{ -4}
  \text{ 's)}
\]

\[
  \text{(spill «the original program»}
  \text{ 'b}
  \text{ -4}
  \text{ 's)}
\]

See the assignment handout for more details on the precise spec for test cases and your spill function’s interface