You can see where it came from, though.

This is obviously not the best code for our program.

```assembly
addl $4, %esp

push %eax

addl %ebx, %eax

call input

.text

Among other things, the target code generator must decide how to use the
what ranges of values might be held in a given variable
which recently computed values are still "live"

Annotations created in the middle end might include
or something very close

- For unused values, we may also keep track of whether an error message
  - If it isn't there already, we add it.

If we traverse this tree left-to-right (given the calculator's simple
write A             C := A + B
- types of expressions

In practice, construction of the AST is often interleaved with parsing,
- and so on
- labels on the arms of a switch (case) statement must be disjoint

expressed in the CFG.  These are STATIC semantics.

mo ->  * | /

write sum / 2
sum := A + B

input program to print the sum and average of two numbers:
```

MORE ON THE VARIOUS PHASES:

Phases within a pass may not be clearly differentiated.
(+) for the "middle end" [the big part]
(better target code

INPUT PROGRAM AGAINST TARGET LANGUAGE

MODIFIED INTERMEDIATE FORM                                        --------

Phases within a pass may not be clearly differentiated.
(+) for the "middle end" [the big part]