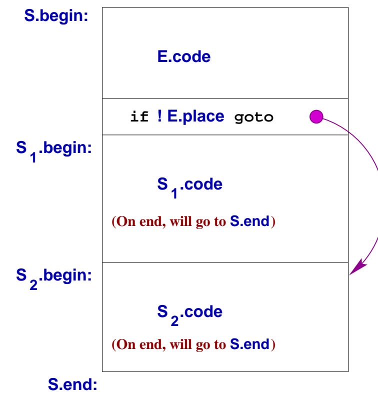
## If Statements: An Alternative

$S \longrightarrow$ if $E$ then $S_1$ else $S_2$






## Conditional Statements and Continuations

$S \longrightarrow$ if $E$ then $S_1$ else $S_2$ {

$S.begin = get_new_label();$

$S_1.end = S_2.end = S.end;$

$S.code = emit(S.begin\!:) \;\|$

$E.code \;\|$

$emit($`if` $\neg E.place$ goto $S_2.begin) \;\|$

$S_1.code \;\|$

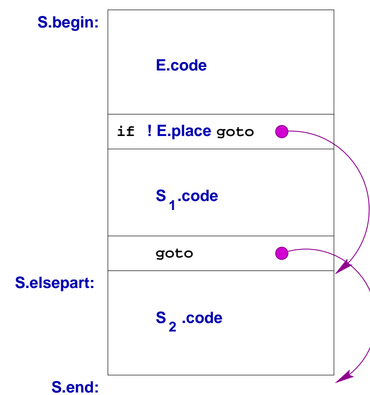
$S_2.code;$

}



## Code Generation for Statements

$S \longrightarrow$ if $E$ then $S_1$ else $S_2$






## Conditional Statements

$S \longrightarrow$ if $E$ then $S_1$ else $S_2$ {

$elselabel = get_new_label();$

$endlabel = get_new_label();$

$S.code = E.code \;\|$

$emit($`if` $\neg E.place$ goto $elselabel) \;\|$

$S_1.code \;\|$

$emit($`goto` $endlabel) \;\|$

$emit(elselabel\!:) \;\|$

$S_2.code \;\|$

$emit(endlabel\!:)$

}

## Generating Shortcircuit Code

Use two continuations for each boolean expression:

- $E.success$: where control will go when expression in $E$ evaluates to *true*.
- $E.fail$: where control will go when expression in $E$ evaluates to *false*.



## Shortcircuit Code for Boolean Expressions

$$
\begin{array}{lcll}
E & \longrightarrow & E_1 \;\&\&\; E_2 & \{ \\
 & & & E_1.fail = E.fail; \\
 & & & E_2.fail = E.fail; \\
 & & & E_1.success = get_new_label(); \\
 & & & E_2.success = E.success; \\
 & & & E.code = E_1.code \;\| \\
 & & & \qquad E_2.code \\
 & & & \} \\
E & \longrightarrow & !\; E_1 & \{ \\
 & & & E_1.fail = E.success; \\
 & & & E_1.success = E.fail; \\
 & & & \} \\
E & \longrightarrow & \mathsf{true} & E.code = emit(\texttt{goto}\ E.success)
\end{array}
$$



## Continuations

An attribute of a statement that specifies where control will flow to ***after*** the statement is executed.

- Analogous to the *follow* sets of grammar symbols.
- In deterministic languages, there is only one continuation for each statement.
- Can be generalized to include local variables whose values are needed to execute the following statements:

  Uniformly captures *call*, *return* and ***exceptions***.



## Code Generation for Boolean Expressions

$$
\begin{array}{lcll}
E & \longrightarrow & E_1 \;\&\&\; E_2 & \{ \\
 & & & E.code = E_1.code \;\| \\
 & & & \qquad E_2.code \;\| \\
 & & & \qquad emit(\texttt{and}) \\
 & & & \} \\
E & \longrightarrow & !\; E_1 & \{ \\
 & & & E.code = E_1.code \;\| \\
 & & & \qquad emit(\texttt{not}) \\
 & & & \} \\
E & \longrightarrow & \mathsf{true} & E.code = emit('1') \\
E & \longrightarrow & \mathsf{id} & E.code = emit(\mathsf{id}.place)
\end{array}
$$

## Code Generation

After intermediate code is generated,

- **Optimize** intermediate code using target machine-independent techniques.

  Examples:
  - constant propagation
  - loop-invariant code motion
  - dead-code elimination
  - strength reduction
- **Generate** final machine code

  Perform target machine-specific optimizations.



## Constant Propagation

Identify expressions that can be evaluated at compile time, and replace them with their values.

```
x = 5;              ==>   x = 5;              ==>   x = 5;
y = 2;                    y = 2;                    y = 2;
v = u + y;                v = u + y;                v = u + 2;
z = x * y;                z = x * y;                z = 10;
w = v + z + 2;            w = v + z + 2;            w = v + 12;
...                       ...                       ...
```



## Short-circuit code for Conditional Statements

$S \longrightarrow$ if $E$ then $S_1$ else $S_2$ {

$S.begin = get_new_label();$

$S_1.end = S_2.end = S.end;$

$E.success = S_1.begin;$

$E.fail = S_2.begin;$

$S.code = emit(S.begin{:}) \parallel$

$E.code \parallel$

$S_1.code \parallel$

$S_2.code;$

}



## Continuations and Code Generation

Continuation of a statement is an inherited attribute.

It is not an L-inherited attribute!

Code of statement is a synthesized attribute, but is dependent on its continuation.

***Backpatching:*** Make two passes to generate code.

1. Generate code, leaving "holes" where continuation values are needed.
2. Fill these holes on the next pass.

## Strength Reduction

Replace expensive operations with equivalent cheaper (more efficient) ones.

```
x = 5;                  ⟹   x = 5;
z = x * y;                   z = (y << 2) + y;
...                          ...
```



## Loop Invariant Code Motion

Move code whose effect is ***independent*** of the loop's iteration ***outside*** the loop.

```
for (i=0; i<N; i++) {        ⟹   for (i=0; i<N; i++) {
   for (j=0; j<N; i++) {            base = a + (i * dim1);
      ...  a[i][j] ...              for (j=0; j<N; i++) {
                                       ...  (base + j) ...
```