

CS406: Compilers

Spring 2021

Week 9: IR code for if- statement, loops, switch,
functions

Evaluation Scheme (online mode)

Assessment Component	Weight (%)
Short Quiz	25
Class Participation	5
Assignments	40
Midsem	15
Endsem	15
Total	100

Evaluation Scheme (assessed so far)

Assessment Component	Weight (%)	So far (%)
Short Quiz	25	16
Class Participation	5	2.5
Assignments	40	10
Midsem	15	25
Endsem	15	-
Total	100	53.5

Evaluation Scheme (Revised)

Assessment Component	Weight (%)	So far (%)	Revised
Short Quiz	25	16	Best 11*
Class Participation	5	2.5	2.5
Assignments	40	10	21.5
Midsem	15	25	25
Endsem	15	-	40
Total	100	53.5	100

*Based on
 $\text{ceil}(25/38*16)$

Remaining:

- *two assignments worth 16 points* $\min(16, 11.5)$
- *one endsem worth 40 points. Up to 4.5 points* $(\max(16, 11.5) - 11.5)$ points serve as *cushion* for endsem.

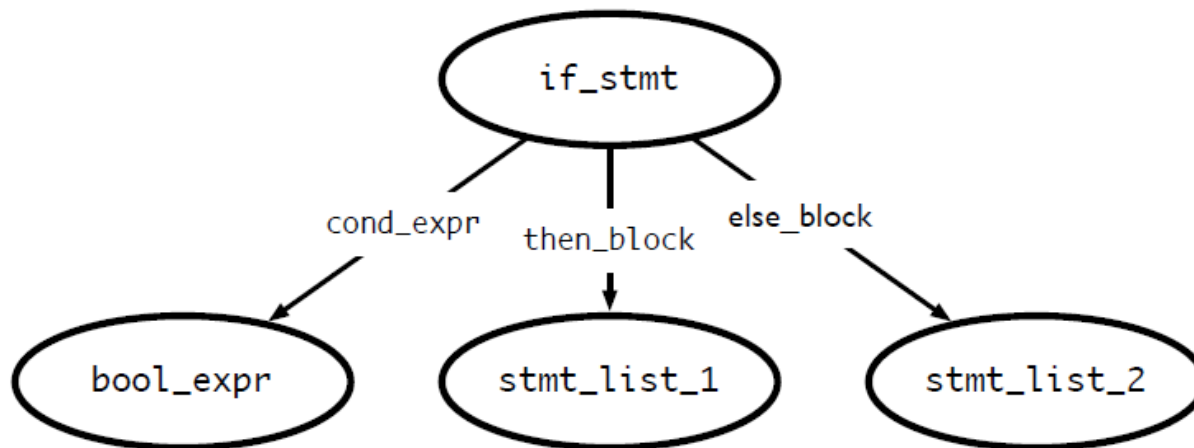
When are you available for CS316? ('17 Batch only)

1. Monday: 2:15 PM to 5:20 PM
2. Tuesday: 3:20 PM to 5:20 PM
3. Wednesday: 2:25 PM to 3:10 PM
and 4:25 PM to 5:20 PM
4. Thursday: 2:15 PM to 5:20 PM
5. Friday: 2:15 PM to 5:20 PM

If statements

```
if <bool_expr_1>  
    <stmt_list_1>  
else  
    <stmt_list_2>  
endif
```

If statements



Code-generation – if-statement

Program text

3AC

INT a, b;

Code-generation – if-statement

Program text

3AC

INT a, b;

Make entries in the
symbol table

Code-generation – if-statement

Program text

3AC

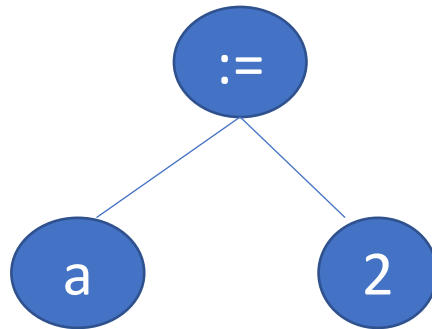
```
INT a, b;  
a := 2;
```

Code-generation – if-statement

Program text

3AC

```
INT a, b;  
a := 2;
```



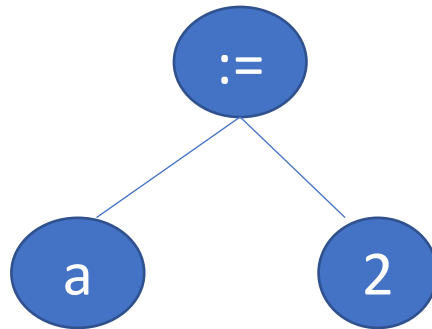
1. “a” is left-child, type=l-val. No code generated. *Return an object containing identifier details after verifying that “a” is present in the symbol table.*

Code-generation – if-statement

Program text

3AC

INT a, b;
a := 2;



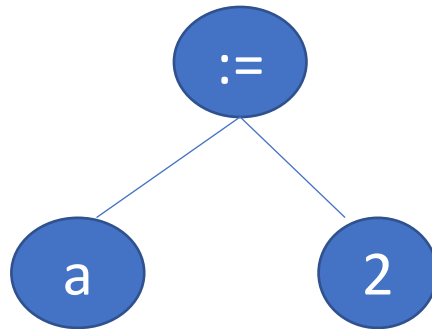
1. “a” is left-child, type=l-val. No code generated. Pass up the identifier.
2. “2” is right-child, type=const. No code generated.

Code-generation – if-statement

Program text

3AC

INT a, b;
a := 2;



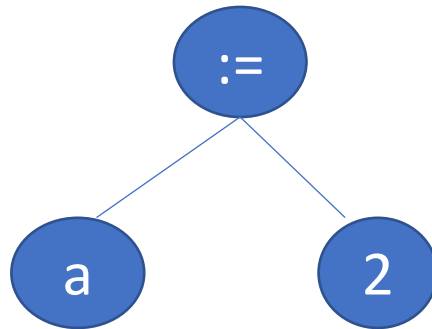
1. “a” is left-child, type=l-val. No code generated. Pass up the identifier.
2. “2” is right-child, type=const. No code generated.
3. Create a temporary T1 to store the result of the expression

Code-generation – if-statement

Program text

3AC

```
INT a, b;  
a := 2;
```

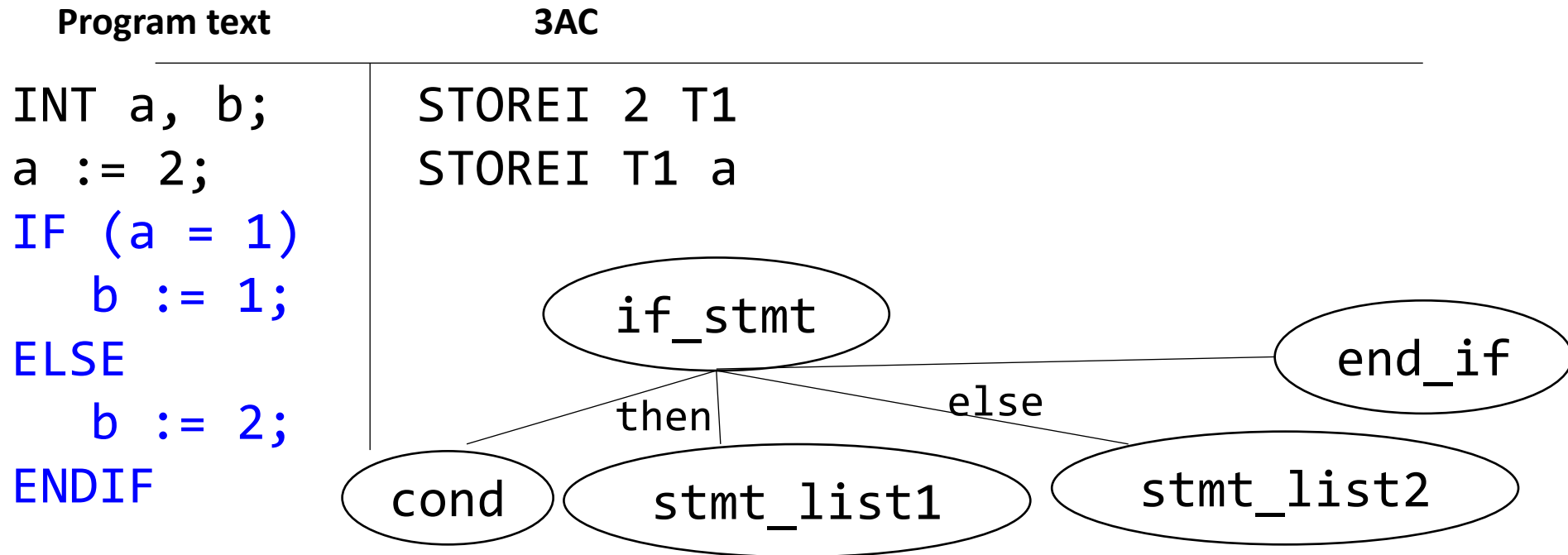


1. “a” is left-child, type=l-val. No code generated. Pass up the identifier.
2. “2” is right-child, type=const. No code generated.
3. Create a temporary T1 to store the result of the expression
 - Current node stores the op ‘:=’. A call to `process_op` stores the RHS data in LHS

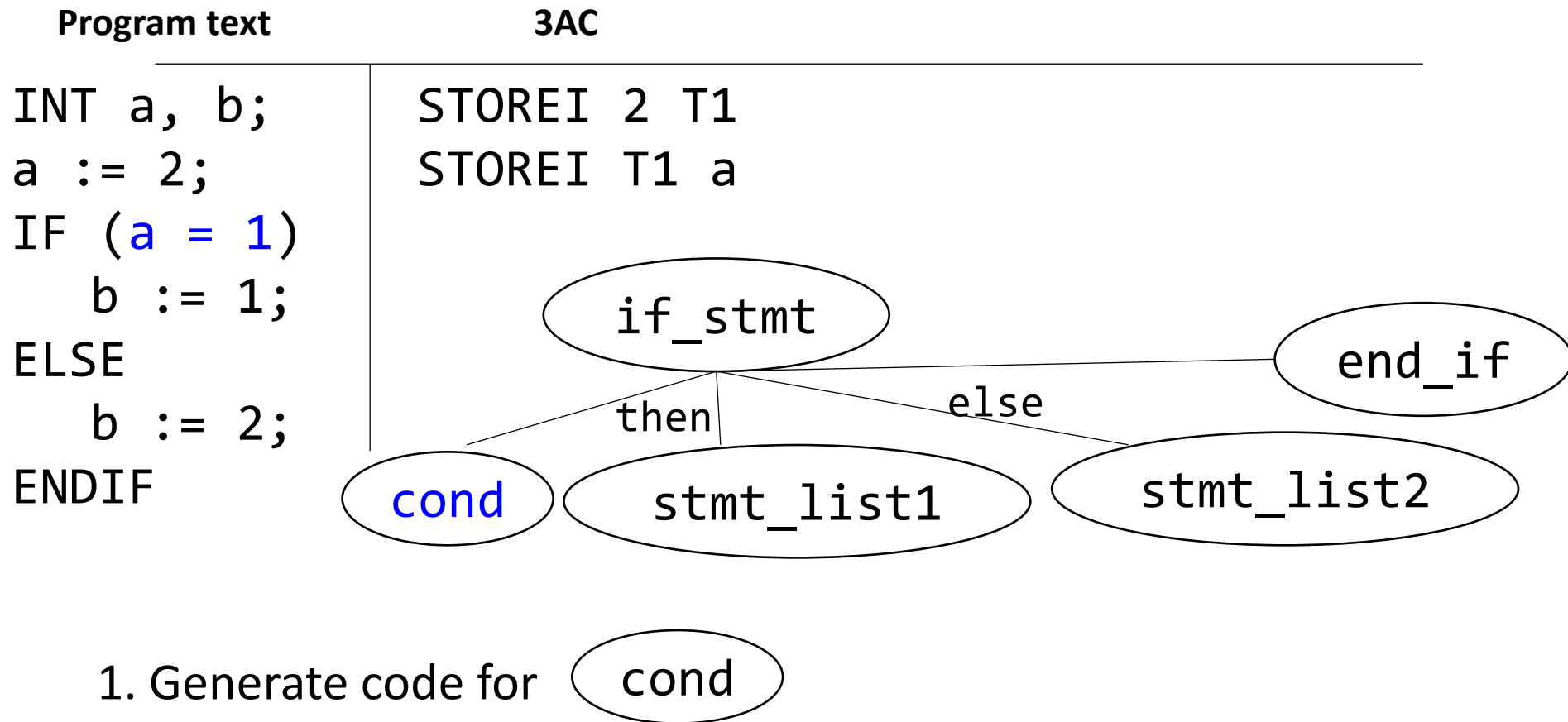
Code-generation – if-statement

Program text	3AC
INT a, b; a := 2;	STOREI 2 T1 STOREI T1 a

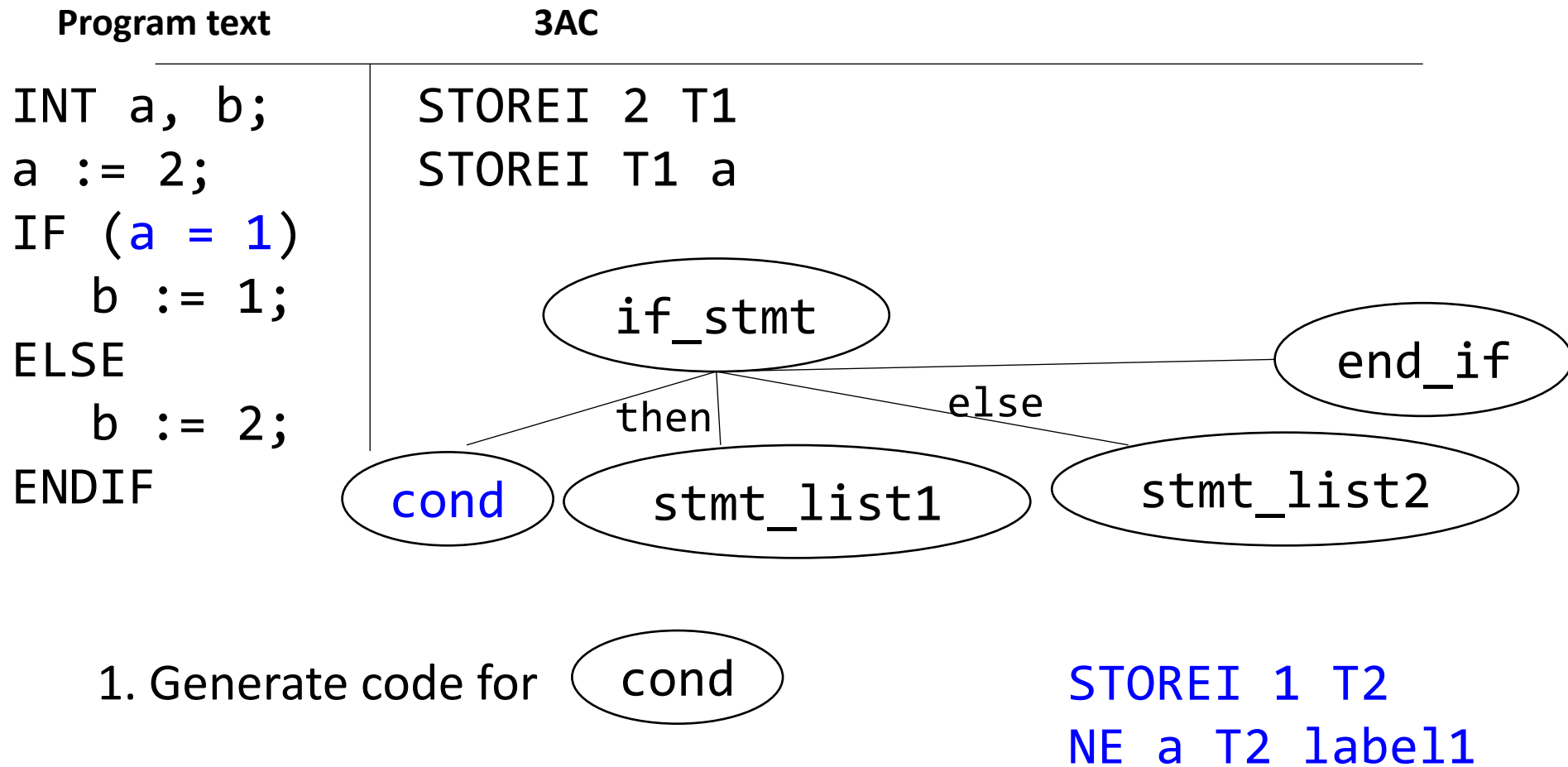
Code-generation – if-statement



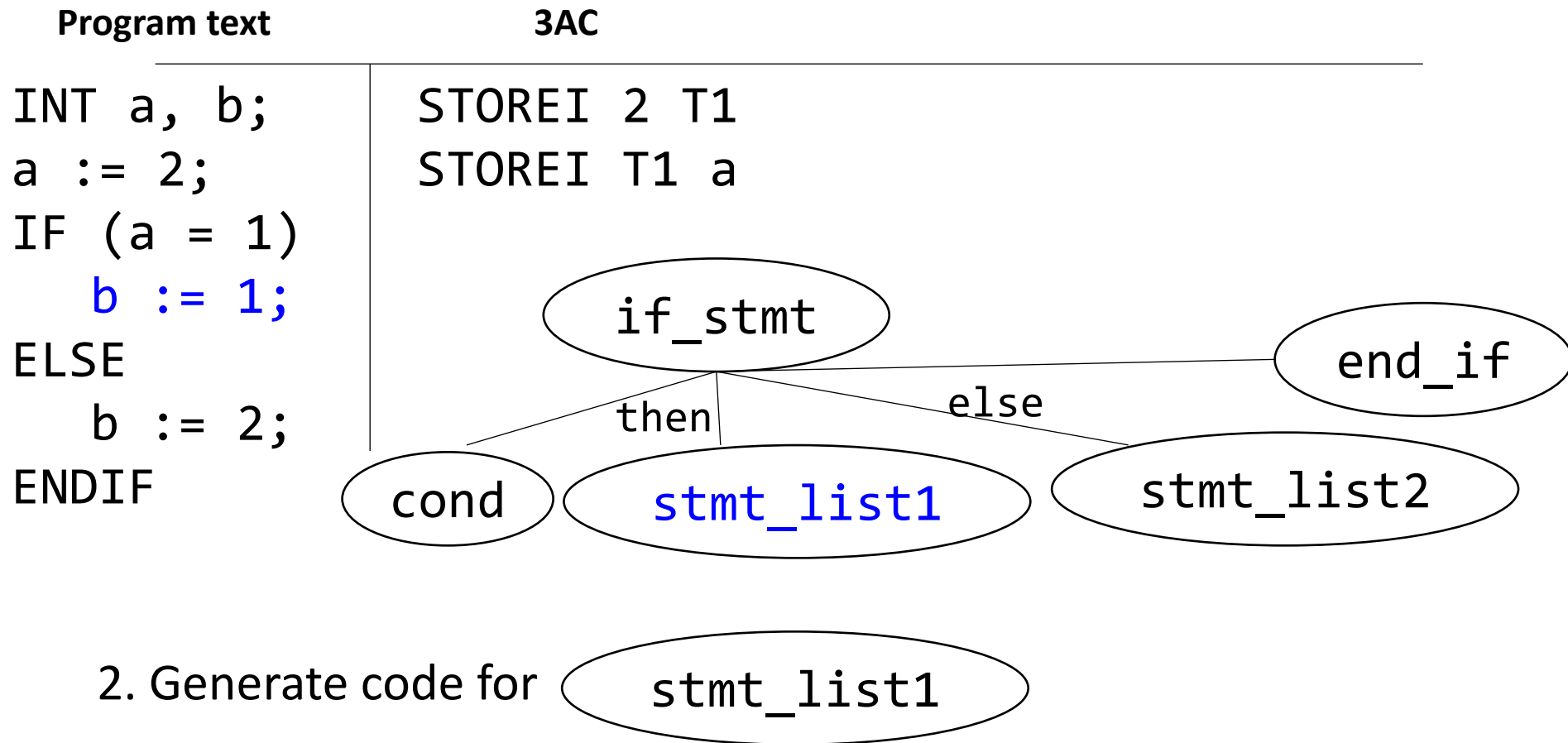
Code-generation – if-statement



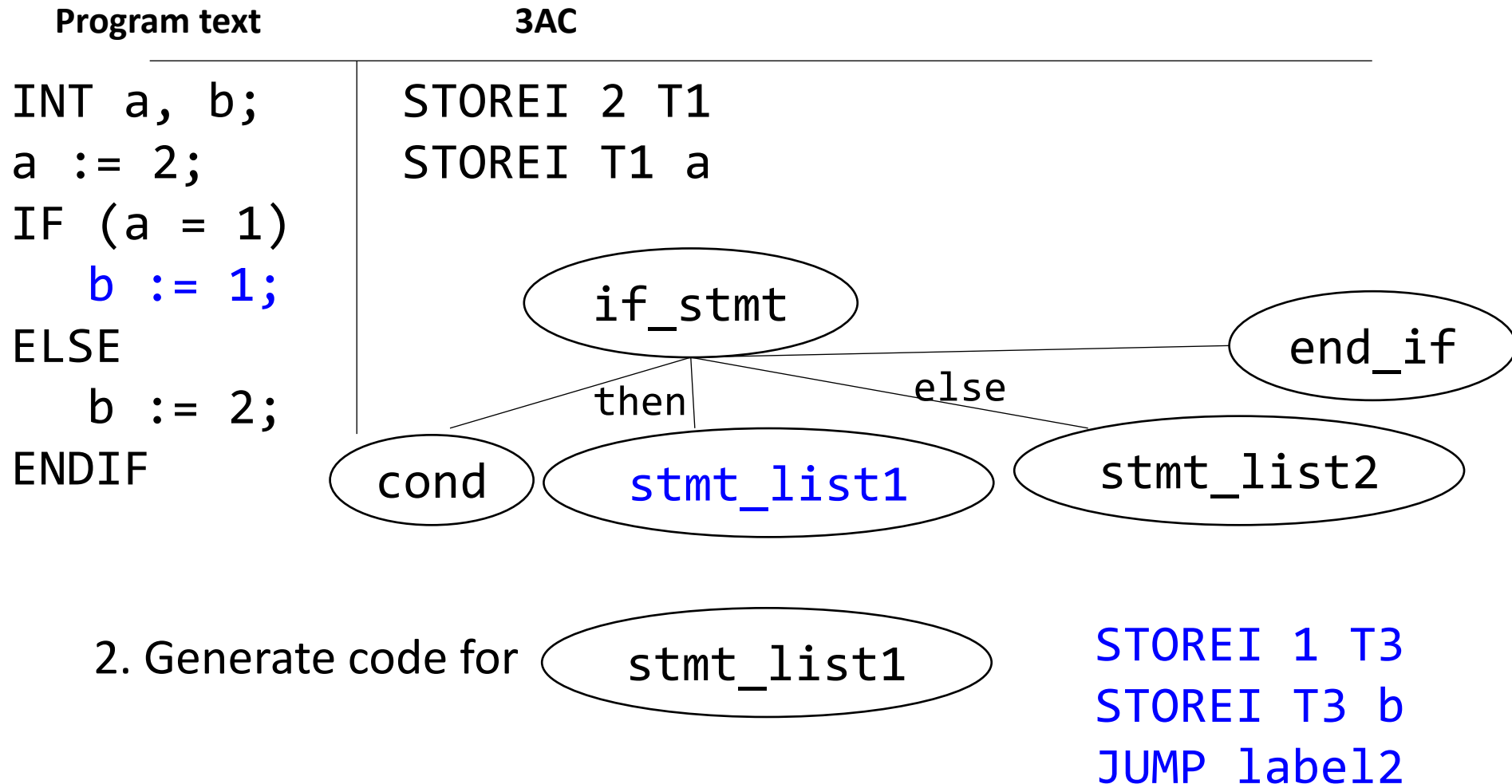
Code-generation – if-statement



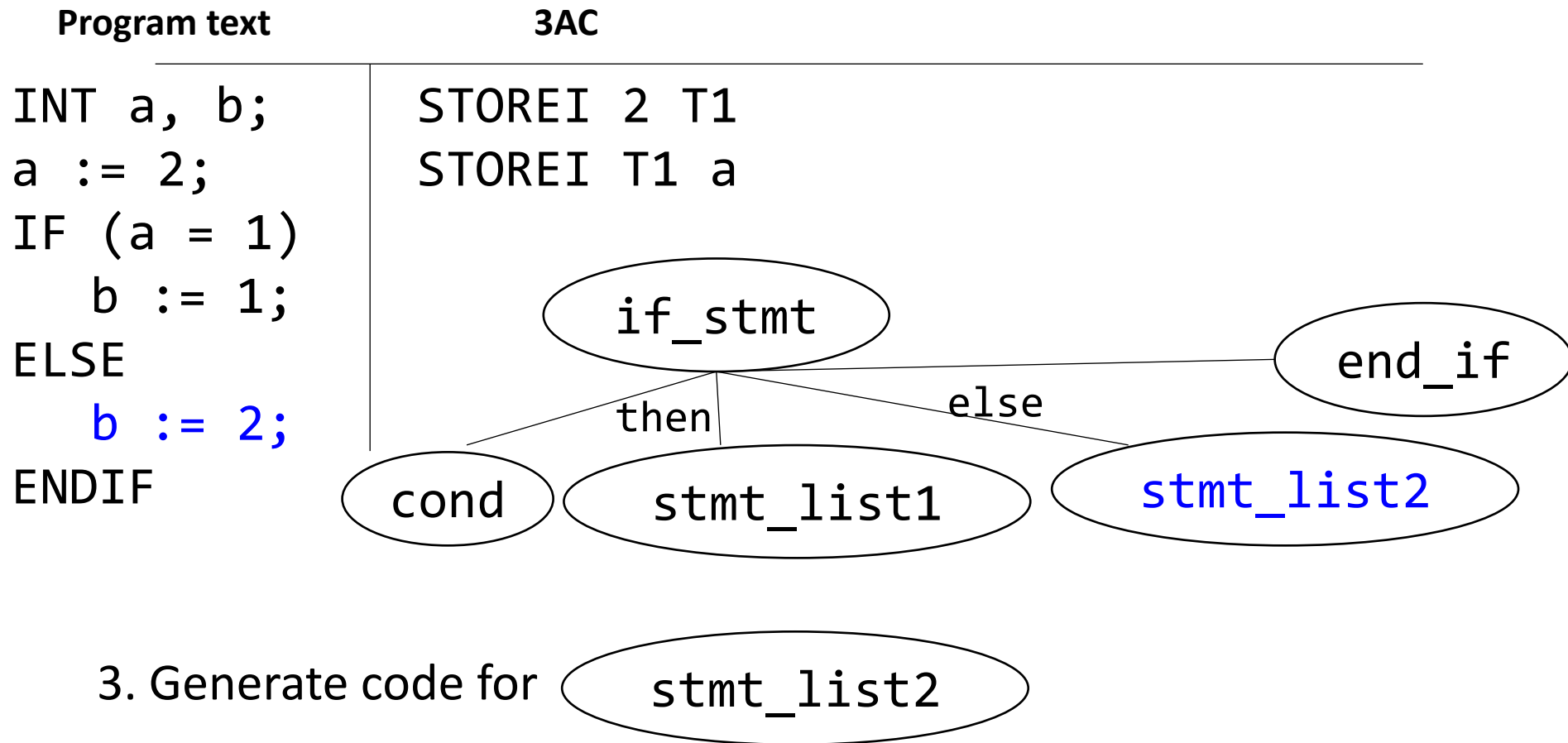
Code-generation – if-statement



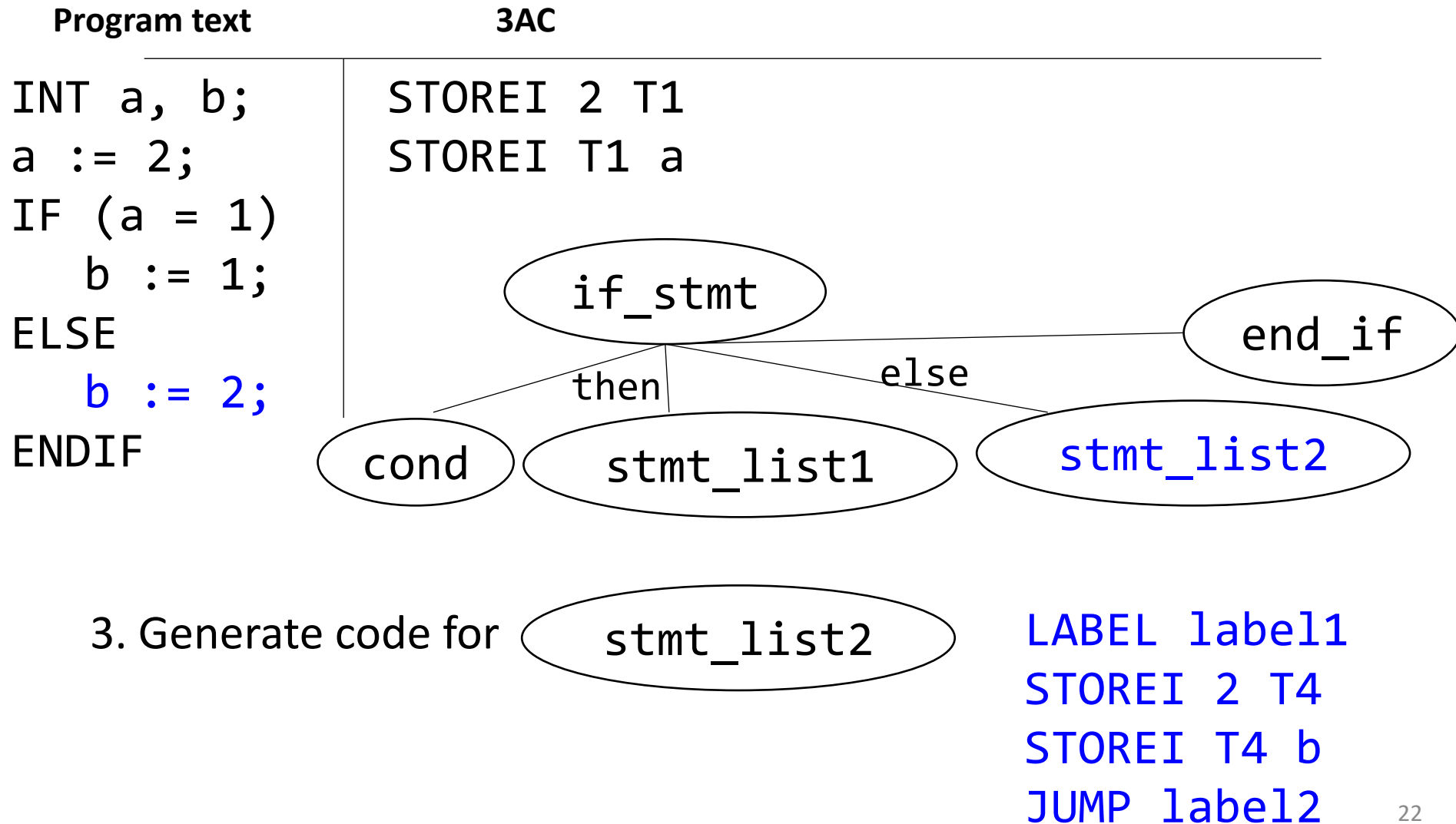
Code-generation – if-statement



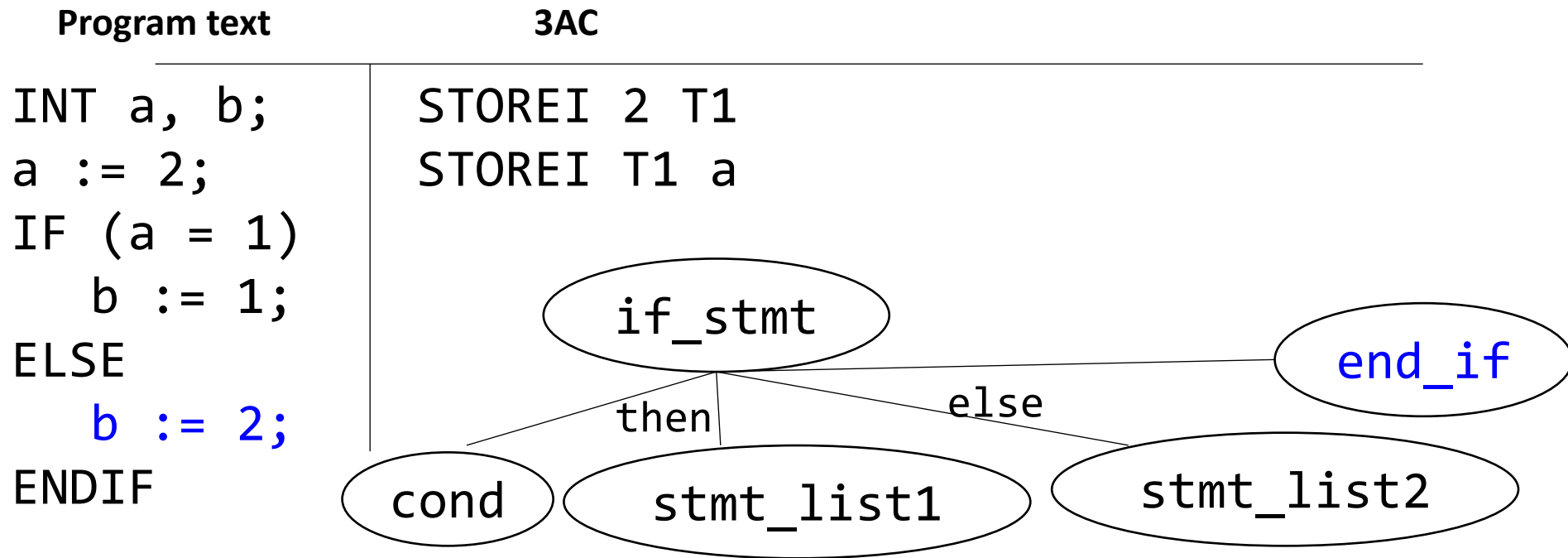
Code-generation – if-statement



Code-generation – if-statement



Code-generation – if-statement



4. Generate code for end_if

LABEL label2

Code-generation – if-statement

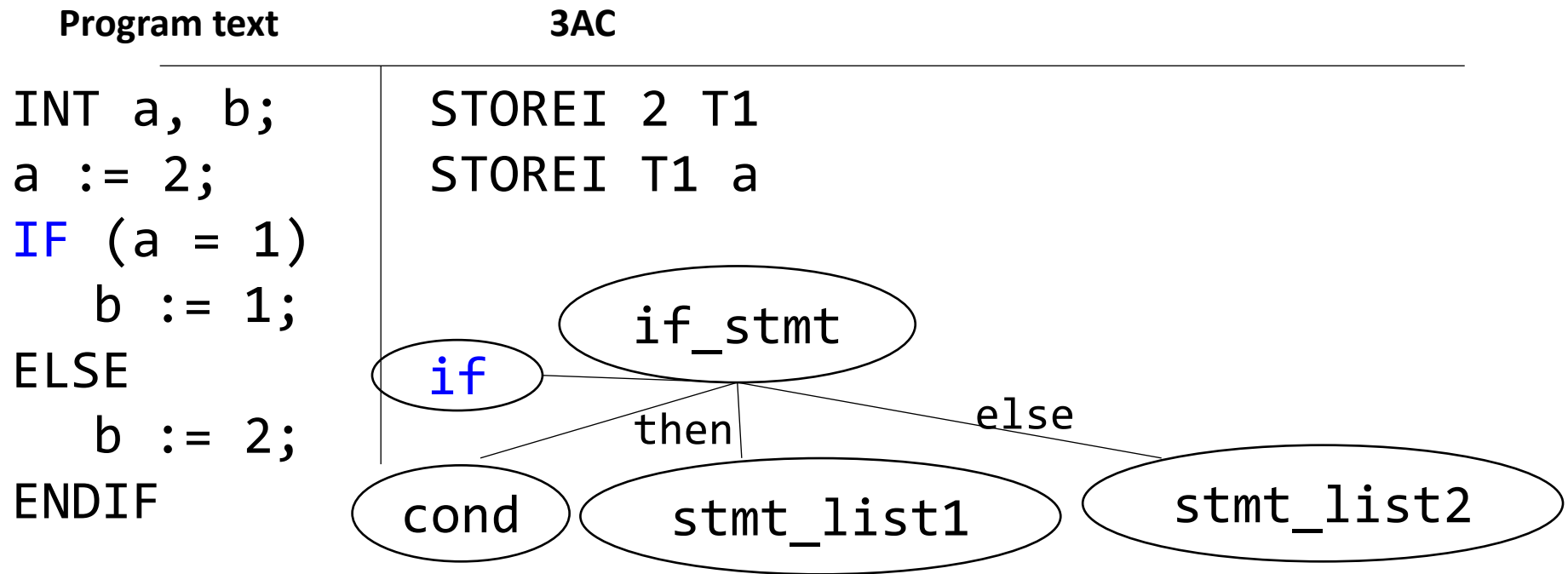
Program text	3AC
INT a, b;	STOREI 2 T1 //a := 2
a := 2;	STOREI T1 a
IF (a = 1)	STOREI 1 T2 //a = 1?
b := 1;	NE a T2 label1
ELSE	STOREI 1 T3 //b := 1
b := 2;	STOREI T3 b
ENDIF	JUMP label2 //to out label
	LABEL label1 //else label
	STOREI 2 T4 //b := 2
	STOREI T4 b
	JUMP label2 //jump to out label
	LABEL label2 //out label

Jumps and Labels?

- Who will generate labels?
- When will the labels be generated?
- To what addresses will the labels be associated with?

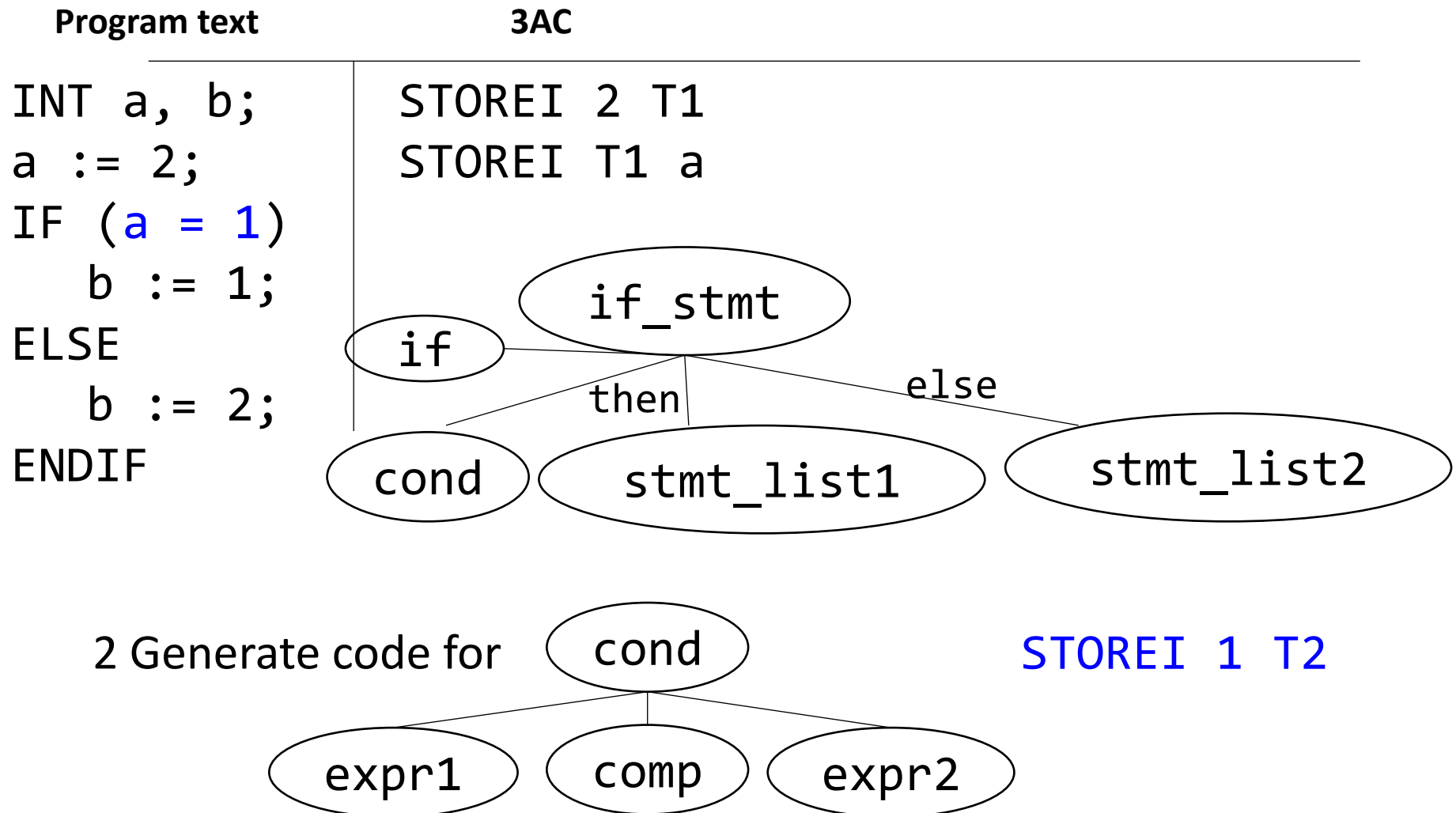
How are targets of jumps decided?

Code-generation – if-statement

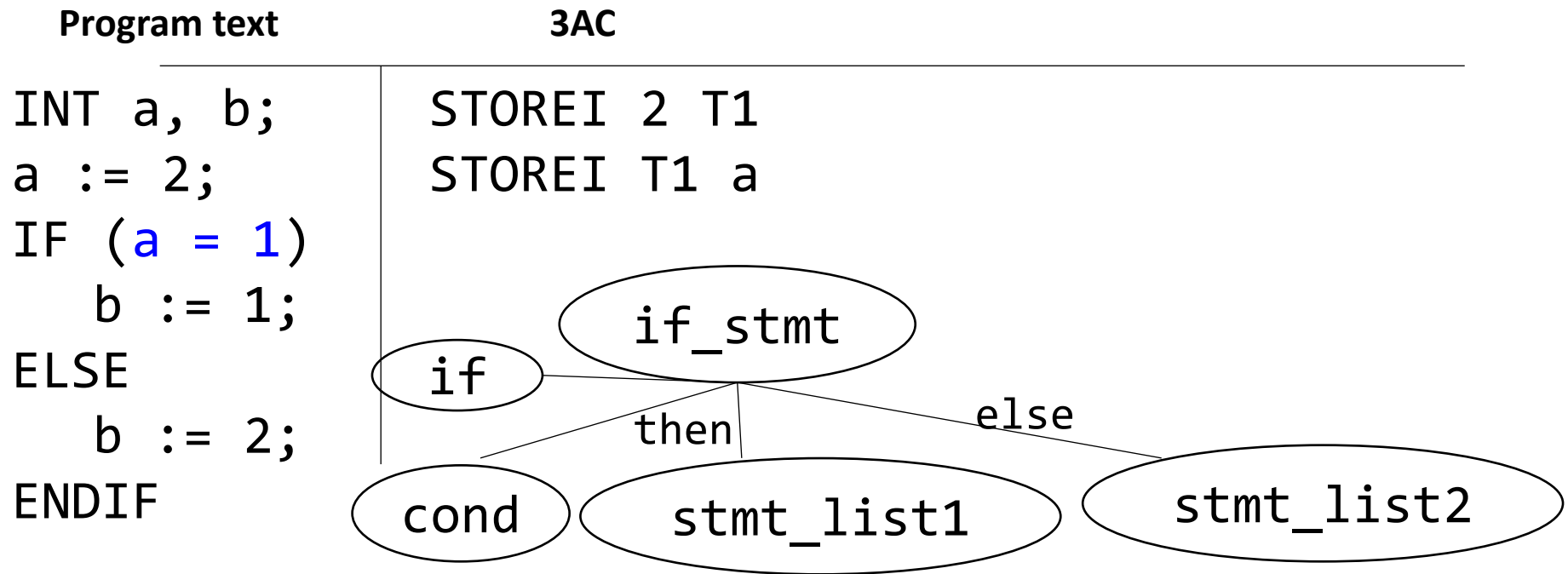


- 1 Generate out label and store it in semantic record of if_stmt (label12)

Code-generation – if-statement

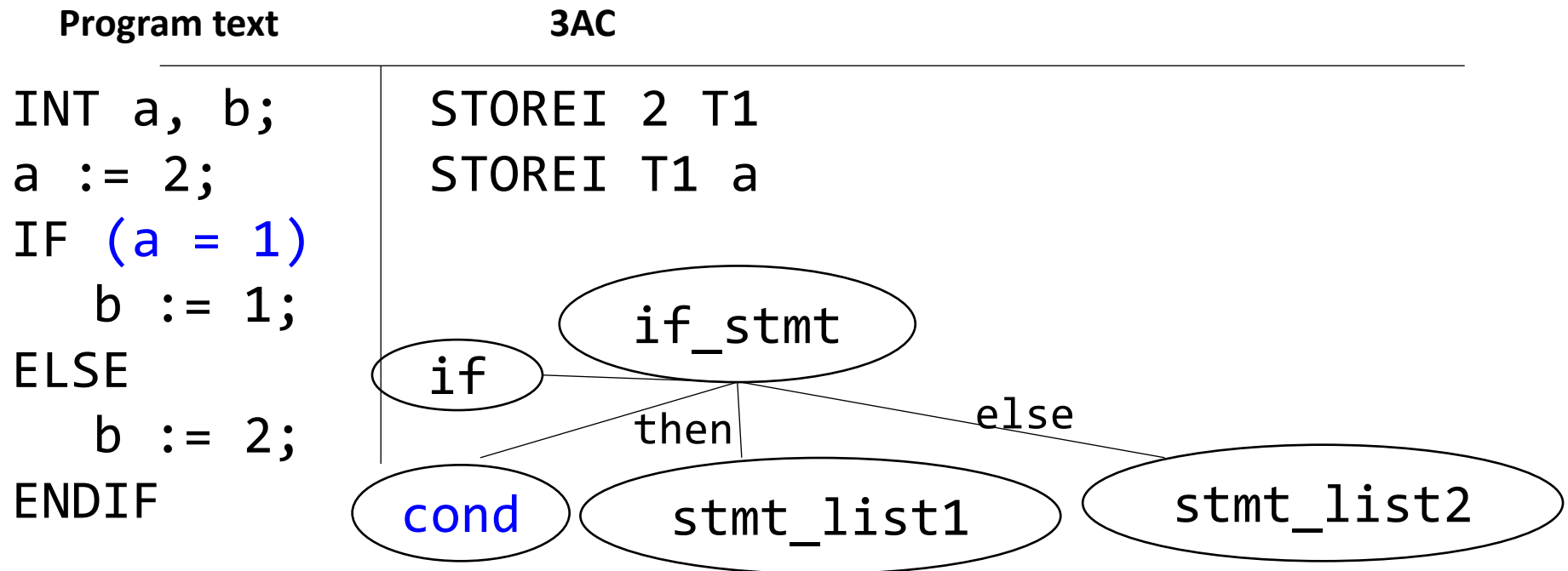


Code-generation – if-statement



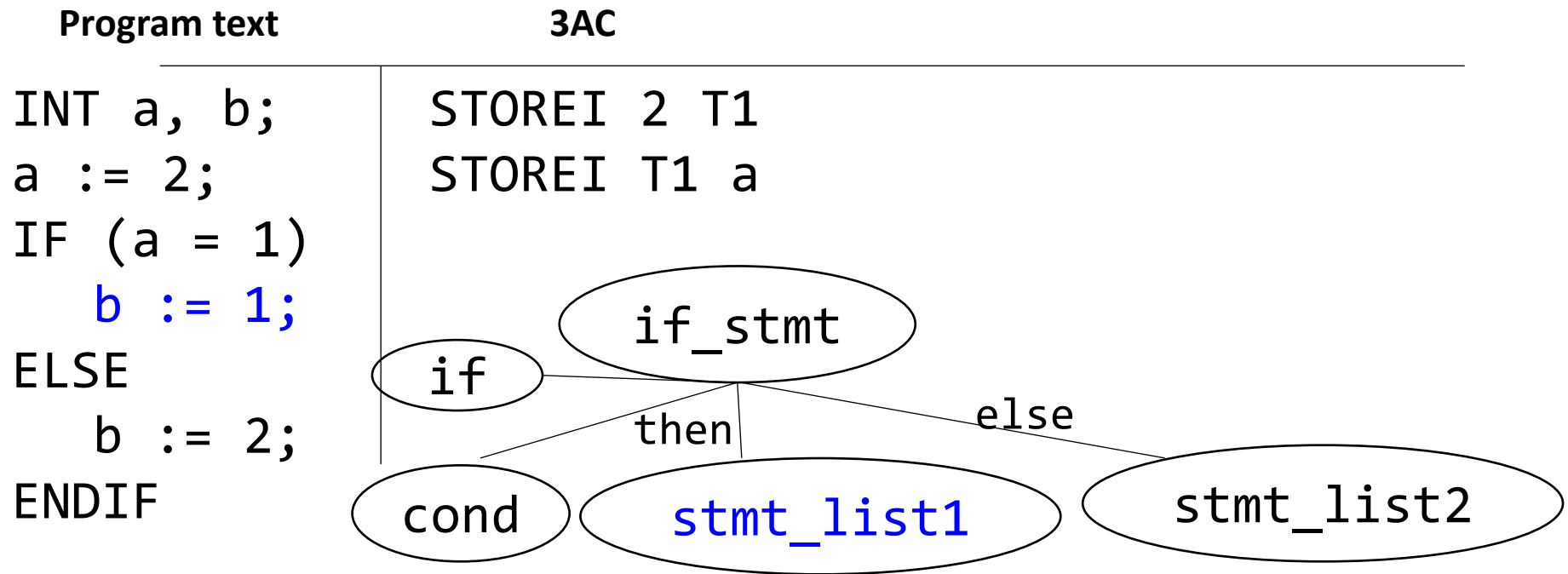
2. Store the result of calling process_op, **STOREI 1 T2**
where op is "=", in the node cond
(bool_expr1=false)

Code-generation – if-statement



2. Cond has been matched. a) Generate label for else part (label11) b) generate statement: JUMP0 bool_expr1 label11
The generated statement conditionally jumps to the else part if cond is false.

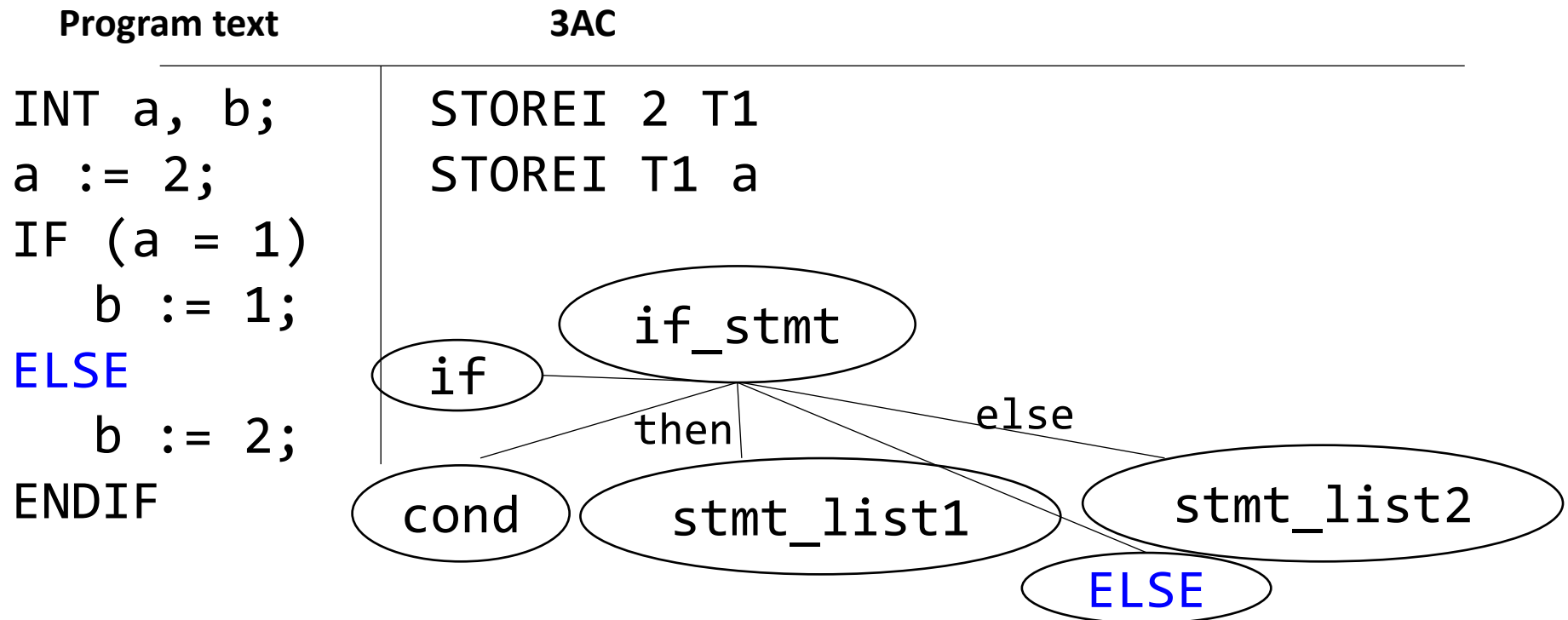
Code-generation – if-statement



3. Generate code for `stmt_list1`

(STOREI 1 T3
STOREI T3 b)

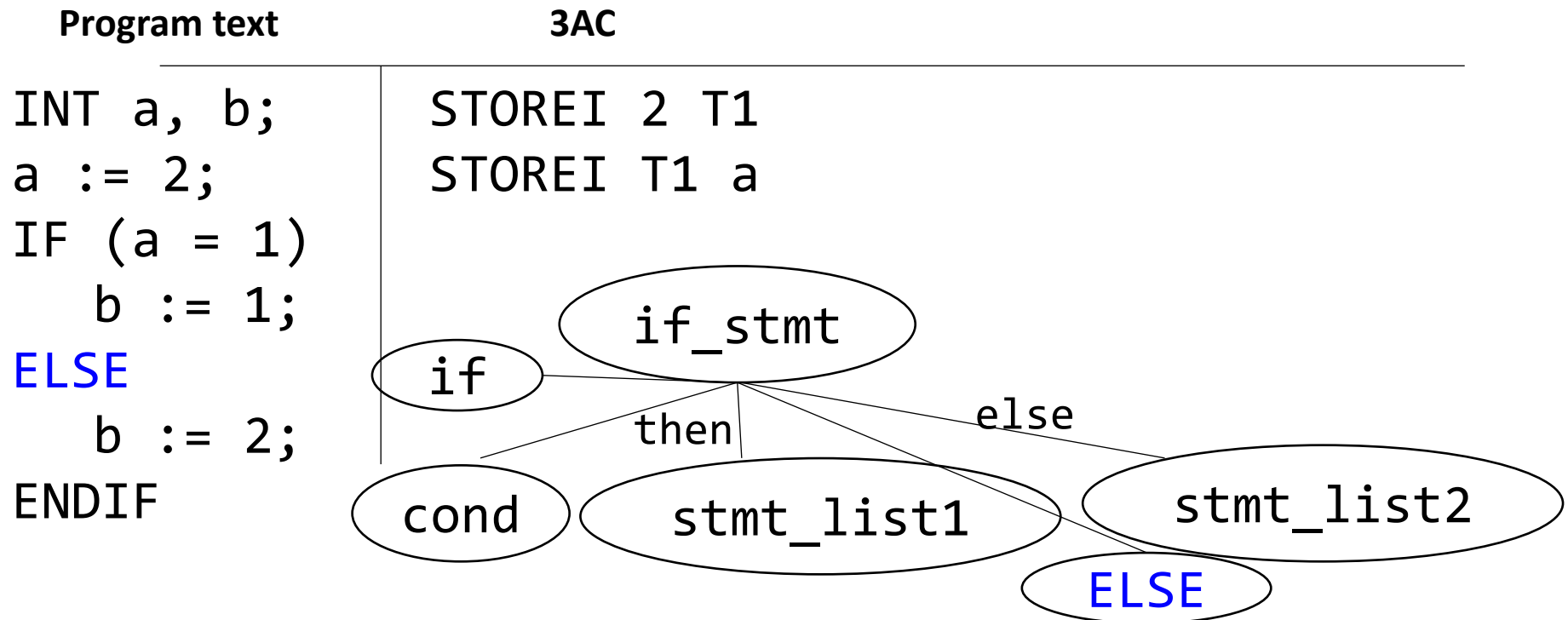
Code-generation – if-statement



4. Generate unconditional jump to out label (label2). Label2 can be obtained from the semantic record of if (slide 26)

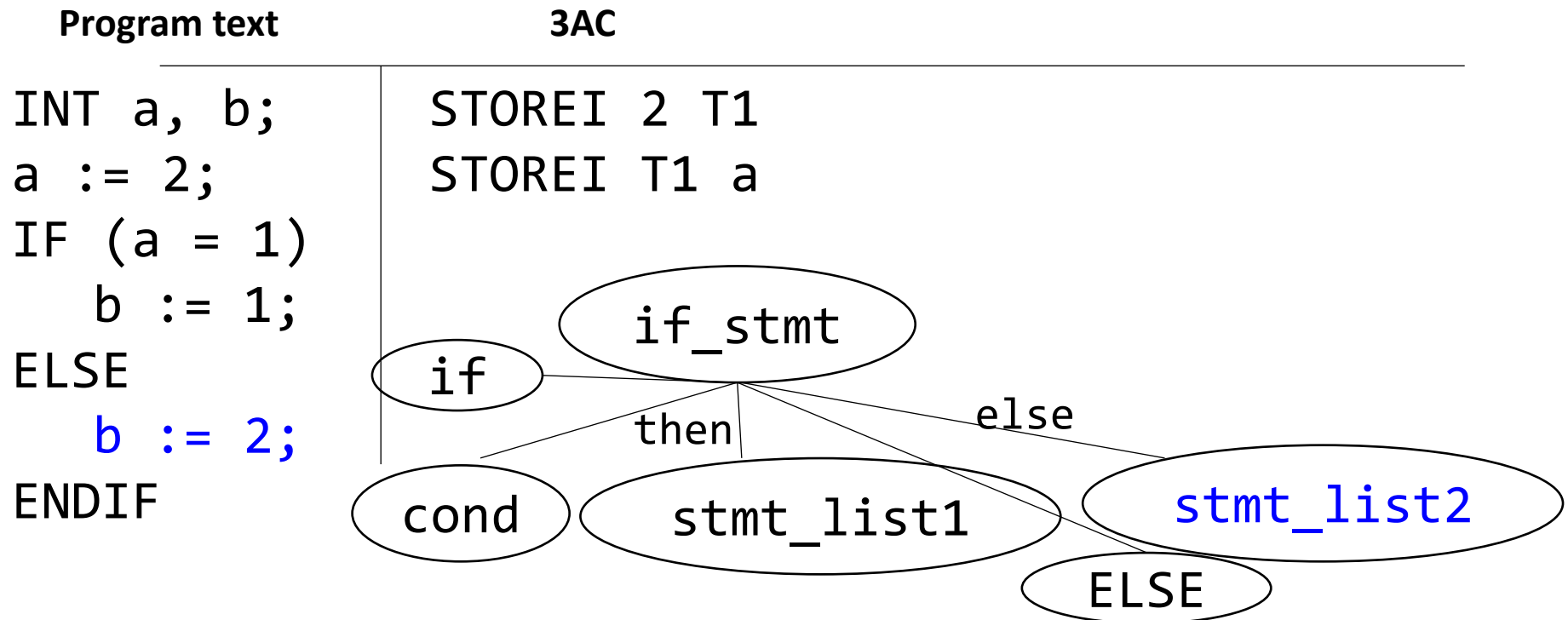
```
JUMP label2
```

Code-generation – if-statement



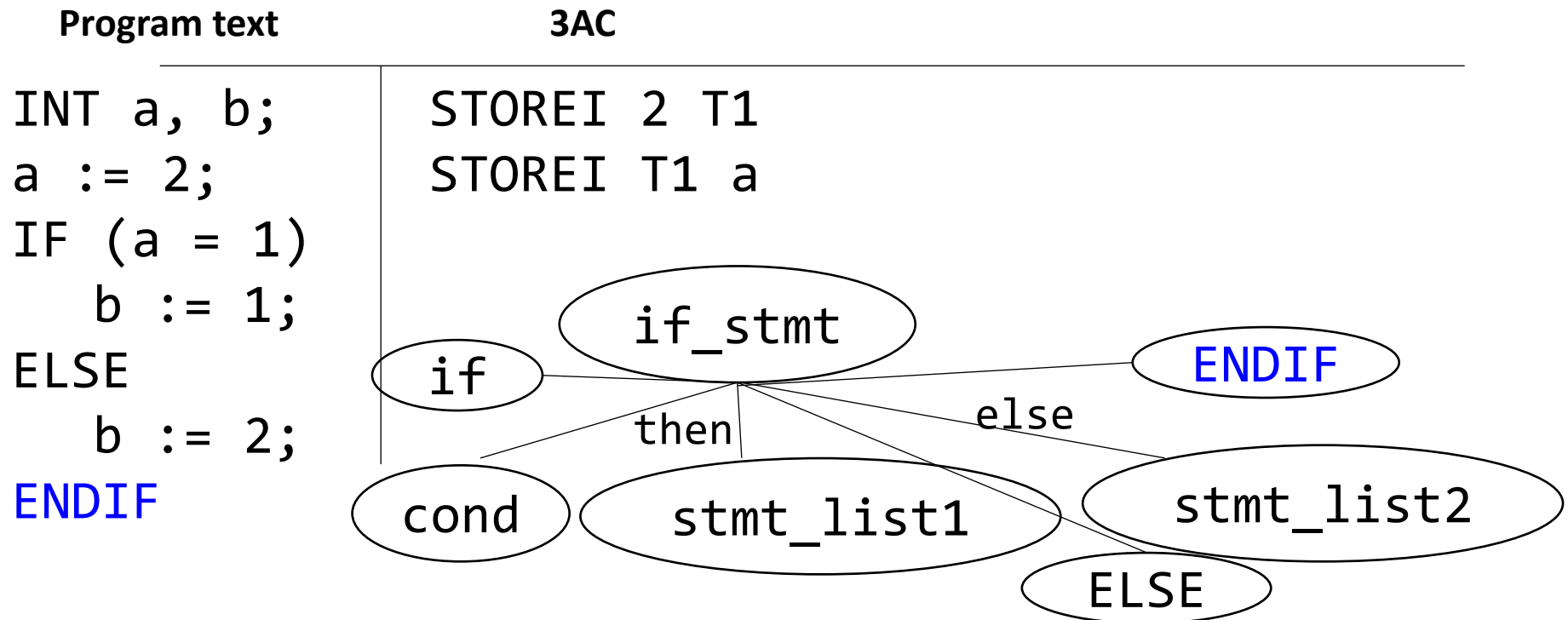
4. Associate else part label (**label11**) with address of next instruction i.e. generate a statement: **LABEL label11**
Label11 can be obtained from semantic record of **if** updated by **cond** (slide 29)

Code-generation – if-statement



5. Generate code for stmt_list2
(STOREI 2 T4
STOREI T4 b)

Code-generation – if-statement



5. Associate out label (label12) with address of next instruction
i.e. generate a statement: LABEL label12

Observations

- We added tokens IF, ELSE, ENDIF to AST
- Generated code is equivalent but not exact
 - e.g. “NE a T2 label1” is replaced with an equivalent “JUMPO bool_expr label1”
- Done in one pass

Will this approach work when generating machine code directly?

Generating code for ifs

```
if <bool_expr_1>  
    <stmt_list_1>  
else  
    <stmt_list_2>  
endif
```

```
<code for bool_expr_1>  
j<!op> ELSE_1  
<code for stmt_list_1>  
jmp OUT_1  
ELSE_1:  
    <code for stmt_list_2>  
OUT_1:
```

Notes on code generation

- The `<op>` in `j<!op>` is dependent on the type of comparison you are doing in `<bool_expr>`
- When you generate JUMP instructions, you should also generate the appropriate LABELs
- Remember: labels have to be unique!

do-while

- `do{S}while(B);` //S is executed at least once and again and again and again... while B remains true

do-while

- `do{S}while(B);` //S is executed at least once and again and again and again... while B remains true

LOOP:

`<stmt_list>`

`<bool_expr>`

`j<!op> OUT`

`jmp LOOP`

OUT:

repeat-until

- `repeat{S}until(B);` //S is executed at least once and again and again and again... while B remains false

repeat-until

- `repeat{S}until(B);` //S is executed at least once and again and again and again... while B remains false

LOOP:

 <stmt_list>

 <bool_expr>

 j<!op> LOOP

OUT: