Intermediate Code Generation

Lecture 8

Exercise

Question?

S → repeat S until E end

- The above grammar defines repeat-until loops, where the loop body is executed at least once; we exit loop when its condition is true.
- Add the required action symbols and write the required semantic routines for such loops. Generate three address codes of the given example.

Input Example:

```
repeat
        a := a-1
        b := b+1
until (a-b)    end
```

Control statements (repeat-until)

S → repeat S until E end

Input Example:

```
repeat
        a := a-1
        b := b+1
until (a-b)    end
```

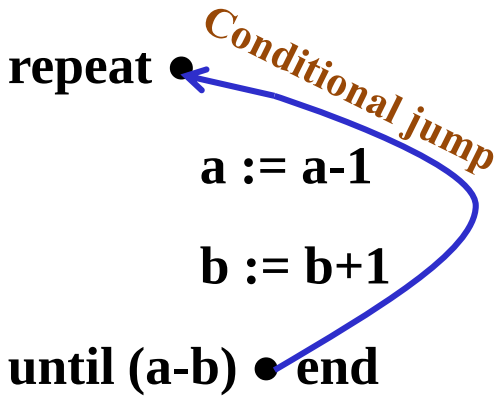
Control statements (repeat-until)

S → repeat #label S until E end

Input Example:

conditional jump: Destination of jump should be saved in **SS** by #label.

(to be used when compiler reaches to the end of loop)



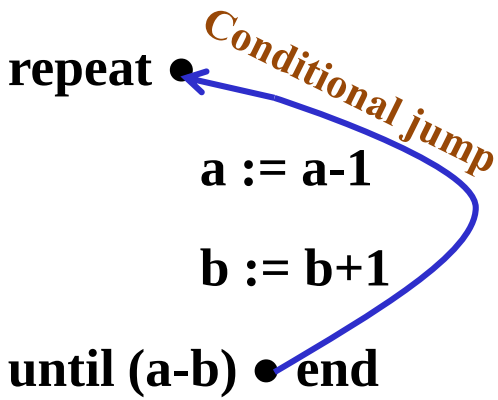

```
#label: begin
                push(i)
        end
```

Control statements (repeat-until)

S → repeat #label S until E #until end

At the end of **repeat-until**, a conditional jump to the start of loop's body (saved by #label) is generated by #until. (No need to *Back Patching*

Input Example:

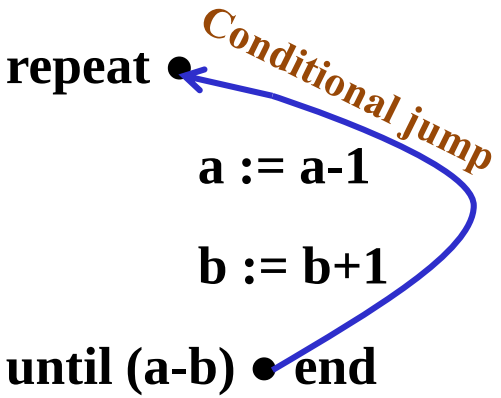


#until: begin
PB[i] ← (jpf, ss(top), ss(top-1),);
i ← i + 1;
pop(2)
end

Control statements (repeat-until)

S → repeat #label S until E #until end

Input Example:



Program Block:

i	PB[i]	Semantic Actions
0	(-, a, #1, t1)	#sub
1	(:=, t1, a,)	#assign
2	(+, b, #1, t2)	#add
3	(:=, t2, b,)	#assign
4	(-, a, b, t3)	#sub
5	(jpf, t3, 0,)	#until
6		

Question?

The following grammar defines syntax of for loops. Add the required action symbols and write the required semantic routines for such loops. Generate three address codes of the given example.

$S \rightarrow$ **for id :=** E_1 **to** E_2 A **do** S **end**
$A \rightarrow$ **by** E_3
$A \rightarrow \varepsilon$

b+c : loop variable (j) initial value
a*b : loop variable (j) limit (constant)
c*d : loop variable (j) step (constant)

Input Example:

```
for j := b+c to a*b by c*a do
        d := d+j
end
```

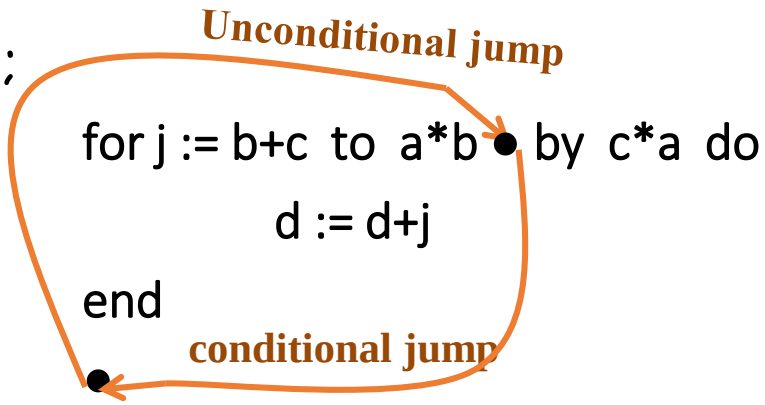
Control Statements (for loop)

$S \rightarrow$ for #pid #pid id := E_1 #assign to E_2 A do S end
$A \rightarrow$ by E_3
$A \rightarrow \varepsilon$

Input Example:

2 #pid put 2 copies of id's address in SS; one copy is used and popped by #assign.

The second copy is later (after seeing E_2) used for comparison with limit of loop's variable.




```
#pid : begin
       p ← findaddr(input);
       push(p)
       end
```

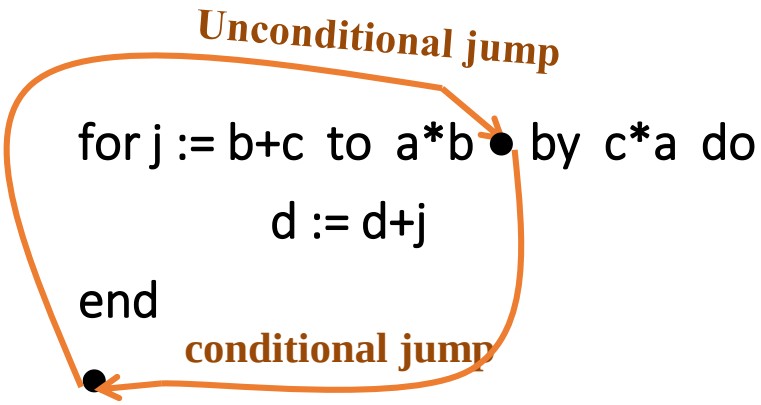
```
#assign : begin
          PB[i] ← (:=, ss(top), ss(top-1),);
          i ← i + 1; pop(2)
          end
```

Control Statements (for loop)

$S \rightarrow$ for #pid #pid id := E_1 #assign to E_2 #save A do S end
$A \rightarrow$ by E_3
$A \rightarrow \varepsilon$

A place for conditional jump is saved to be later used (by *back patching*).

Input Example:

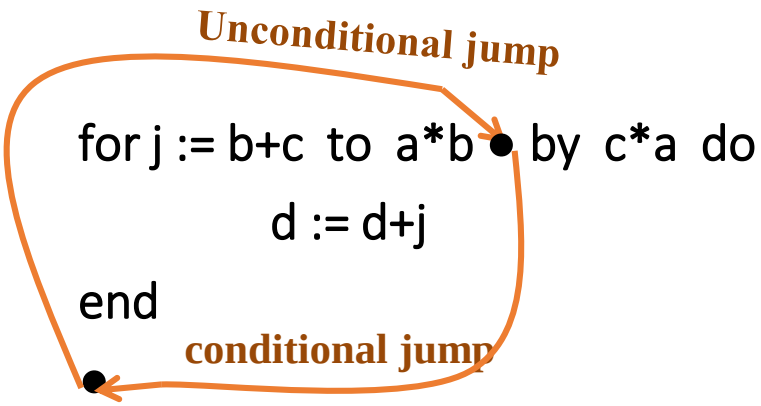


#save: begin
push(i), i←i+1
end

Control Statements (for loop)

S → for #pid #pid id := E_1 #assign to E_2 #save A do S #for end
A → by E_3
A → ε

Input Example:



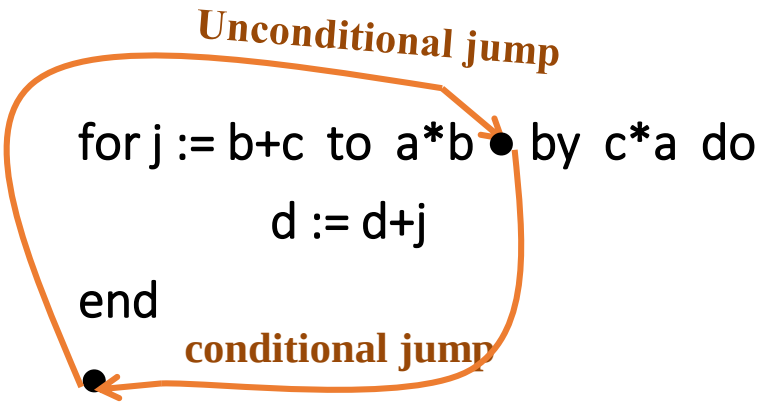
In the end of loop and by semantic routine #for:

- Loop's variable should be increased by step,
- An unconditional jump to the start loop is generated, and
- The place saved by #save should be filled by a conditional jump

Control Statements (for loop)

S → for #pid #pid id := E_1 #assign to E_2 #save A do S #for end
A → by E_3
A → ε

Input Example:

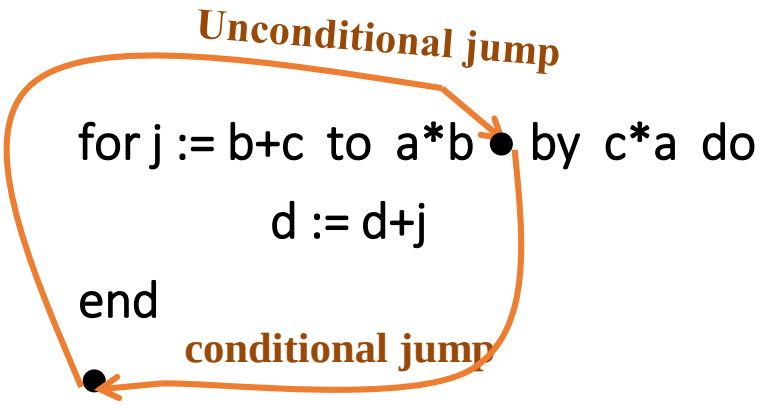


#for: begin
PB[i] ← (+, ss(top), ss(top-3), ss(top-3)); i ← i+1;
PB[i] ← (jp, ss(top-1), ,); i←i+1;
PB[ss(top-1)] ← (>, ss(top-3), ss(top-2), i,);
Pop(4)
end

Control Statements (for loop)

S → for #pid #pid id := E_1 #assign to E_2 #save A do S #for end
A → by E_3
A → #step1

Input Example:



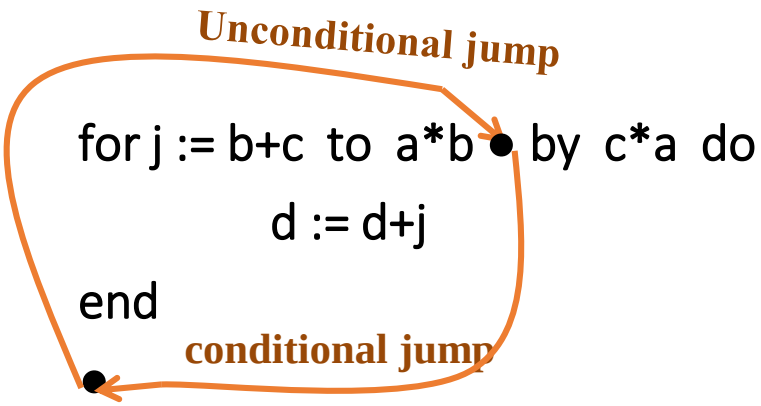
If there is not an explicit step, (A → ε is used) , the step should be set to the default value of 1 (by #step1).

#step1: begin
t← gettemp
PB[i] ← (:=, #1, t,)
i← i+1, push(t)
end

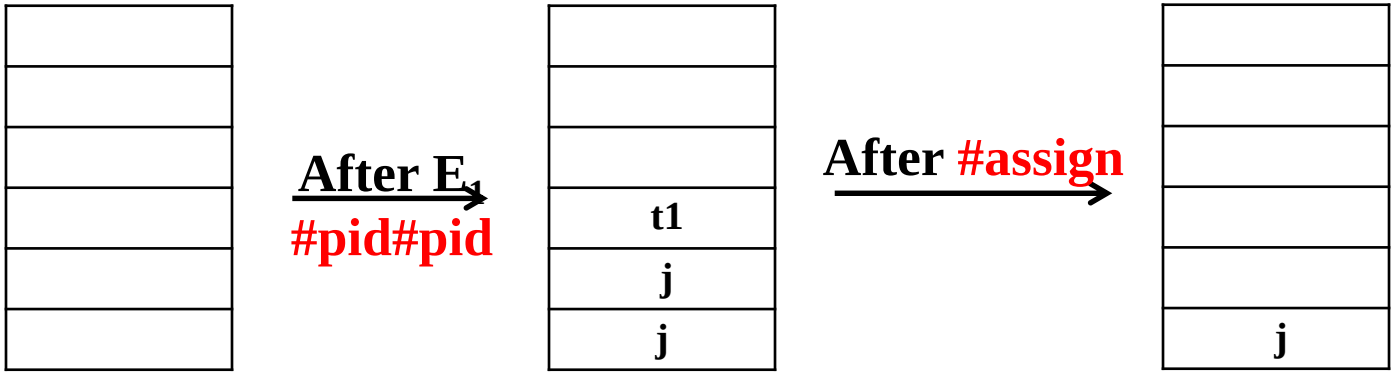
Control Statements (for loop)

S → for #pid #pid id := E_1 #assign to E_2 #save A do S #for end
A → by E_3
A → #step1

Input Example:



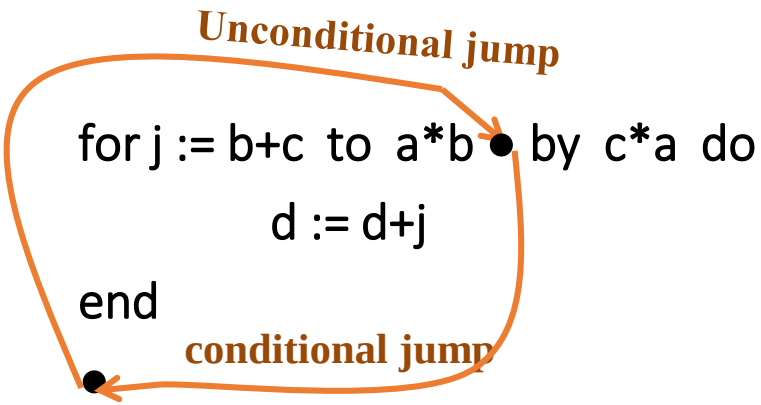
Semantic Stack:



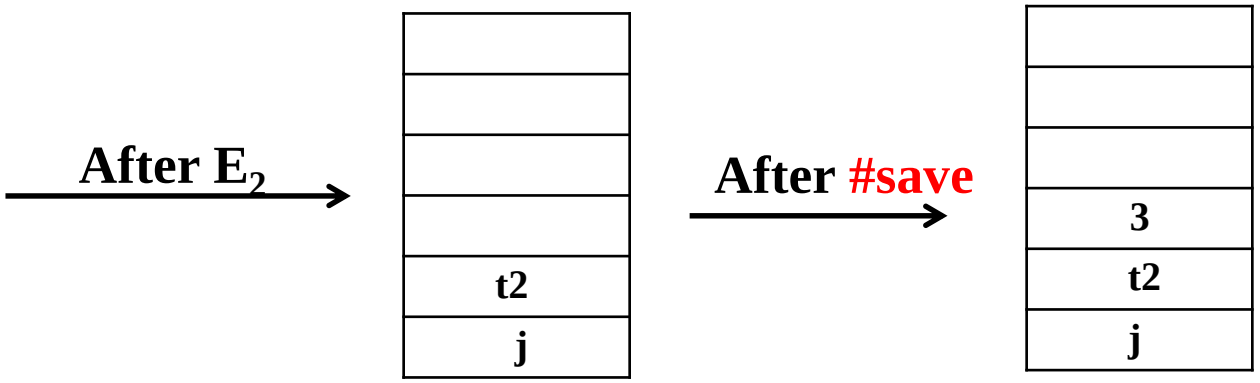
Control Statements (for loop)

S → for #pid #pid id := E_1 #assign to E_2 #save A do S #for end
A → by E_3
A → #step1

Input Example:



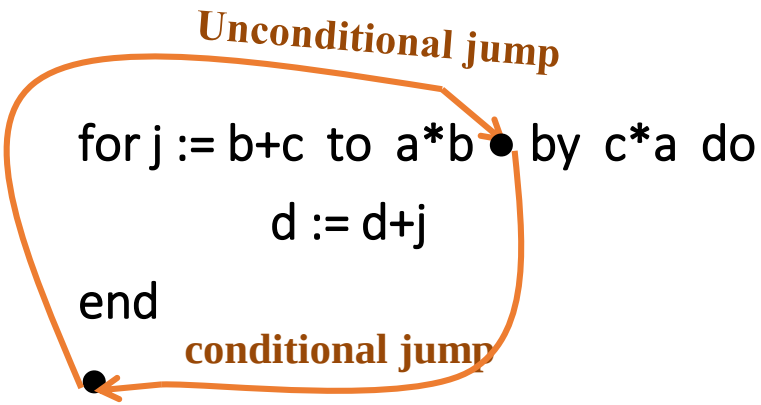
Semantic Stack:



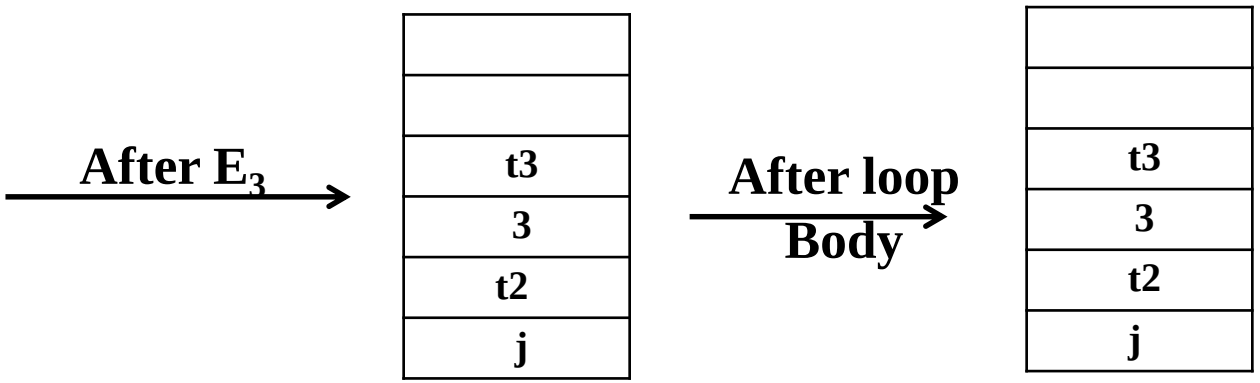
Control Statements (for loop)

S → for #pid #pid id := E_1 #assign to E_2 #save A do S #for end
A → by E_3
A → #step1

Input Example:



Semantic Stack:

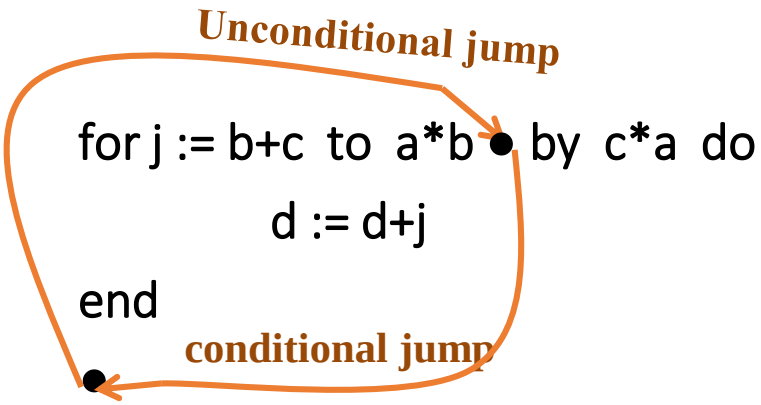


Control Statements (for loop)

S → for #pid #pid id := E_1 #assign to E_2 #save A do S #for end
A → by E_3
A → #step1

i	PB[i]	Semantic Actions
0	(+, b, c, t1)	#add (by E_1)
1	(:=, t1, j,)	#assign
2	(*, a, b, t2)	#mult (by E_2)
3	(>, j, t2, ?=9)	#for
4	(*, c, a, t3)	#mult (by E_3)
5	(+, d, j, t4)	#add (by S)
6	(:=, t4, d,)	#assign (By S)
7	(+ , t3, j, j)	#for
8	(jp, 3, ,)	#for

Input Example:



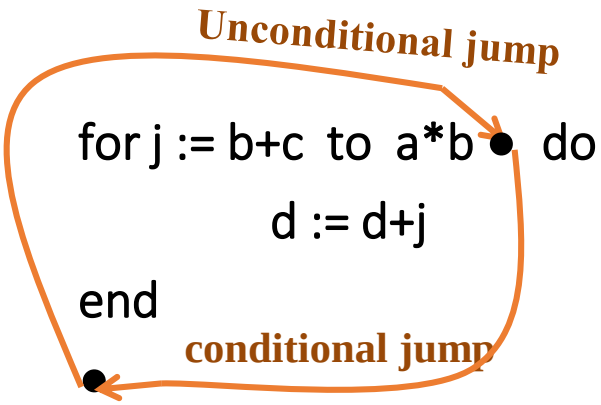
Program Block

Control Statements (for loop without step)

S → for #pid #pid id := E_1 #assign to E_2 #save A do S #for end
A → by E_3
A → #step1

i	PB[i]	Semantic Actions
0	(+, b, c, t1)	#add (by E_1)
1	(:=, t1, j,)	#assign
2	(*, a, b, t2)	#mult (by E_2)
3	(>, j, t2, ?=9)	#for
4	(:=, #1, t3,)	#step1
5	(+, d, j, t4)	#add (by S)
6	(:=, t4, d,)	#assign (By S)
7	(+ , t3, j, j)	#for
8	(jp, 3, ,)	#for

Input Example:



Program Block

Question?

The following grammar defines syntax of case statements, where at most one of case statements is to be executed.

Can we generate intermediate code for these statements by just using a sematic stack to store the addresses that are required for back-patching?

Example:

```
case (c * d) of
        a: a := a + 1;
        b: b := b + 2;
        c: c := c + 3;
        otherwise: e := c*d
end
```

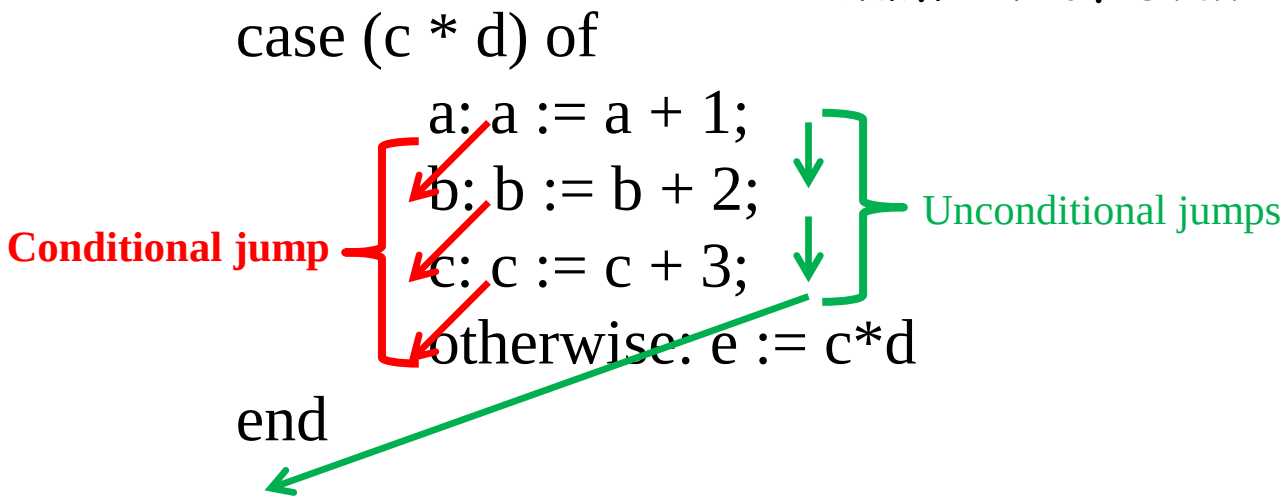
S → case E of L end
L → id: S B
B → ε | otherwise S | ; is : S B

Control Statements (Switch)

S → case E of L end
L → id: S B
B → ε | otherwise S | ; is : S B

Note: At most one statement is executed.

Difficulty in generating unconditional jumps: Variable number of statements



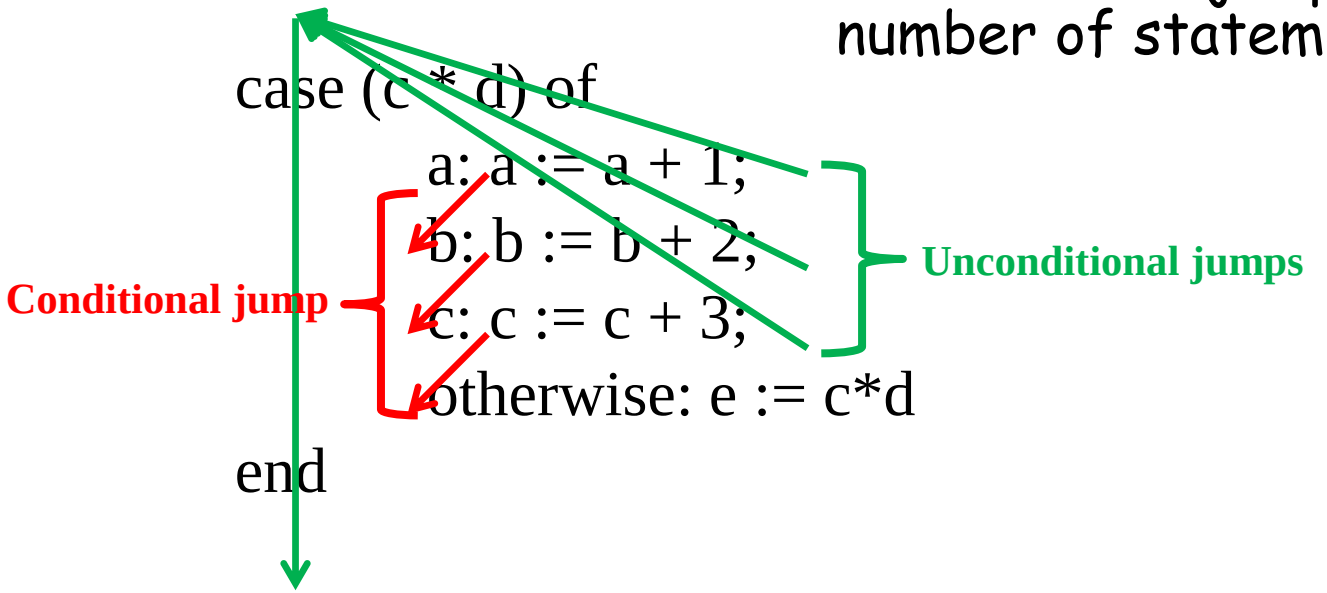
Solution 1: Link unconditional jumps (Inefficient!)

Control Statements (Switch)

S → case E of L end
L → id: S B
B → ε | otherwise S | ; is : S B

Note: At most one statement is executed.

Difficulty in generating unconditional jumps: Variable number of statements



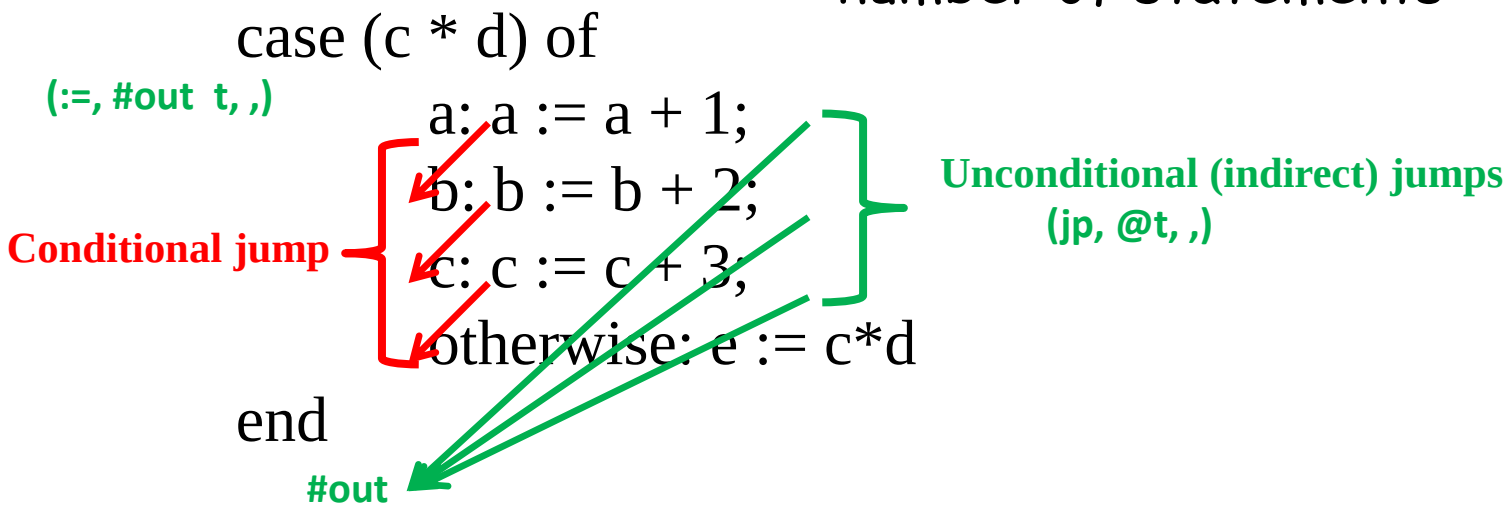
Solution 2: Jump backward! (Only two jumps are needed for exit)

Control Statements (Switch)

S → case E of L end
L → id: S B
B → ε | otherwise S | ; is : S B

Note: At most one statement is executed.

Difficulty in generating unconditional jumps: Variable number of statements



Solution 3: Jump indirectly! (Only one jump is needed for exit)