

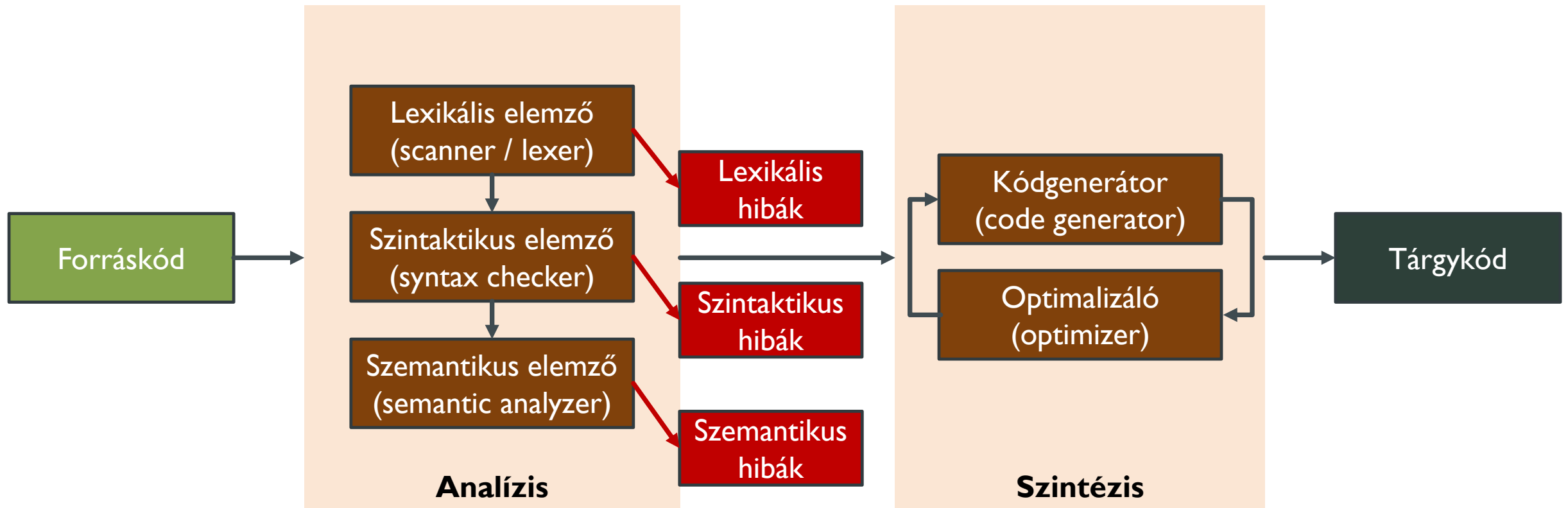


# KÓDGENERÁLÁS

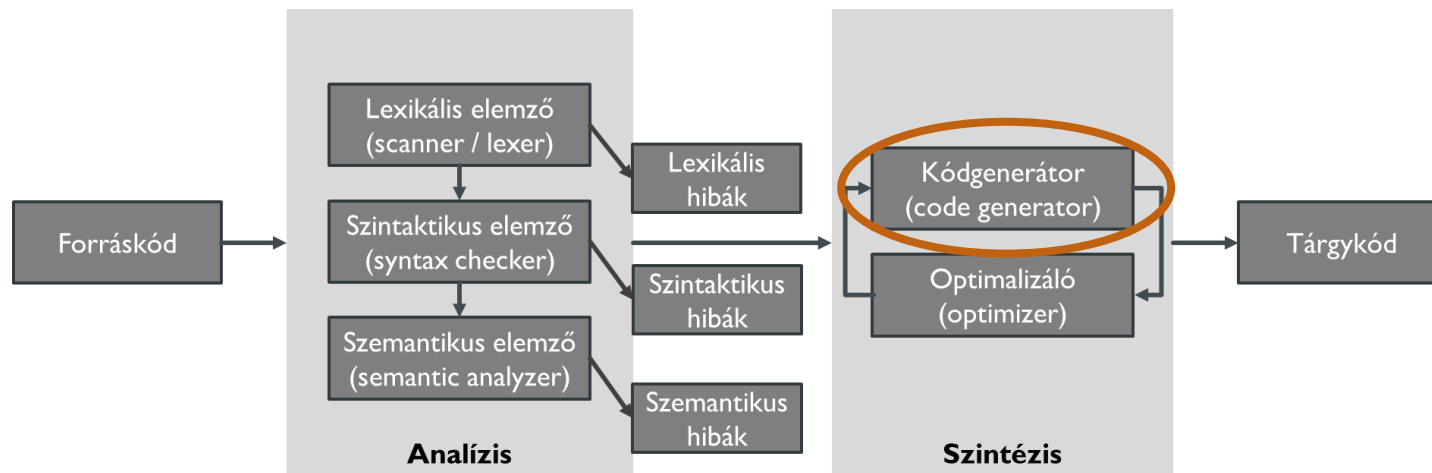
FORMÁLIS NYELVEK ÉS FORDÍTÓPROGRAMOK ALAPJAI

Dévai Gergely  
ELTE

# A FORDÍTÓPROGRAMOK LOGIKAI FELÉPÍTÉSE



# KÓDGENERÁTOR



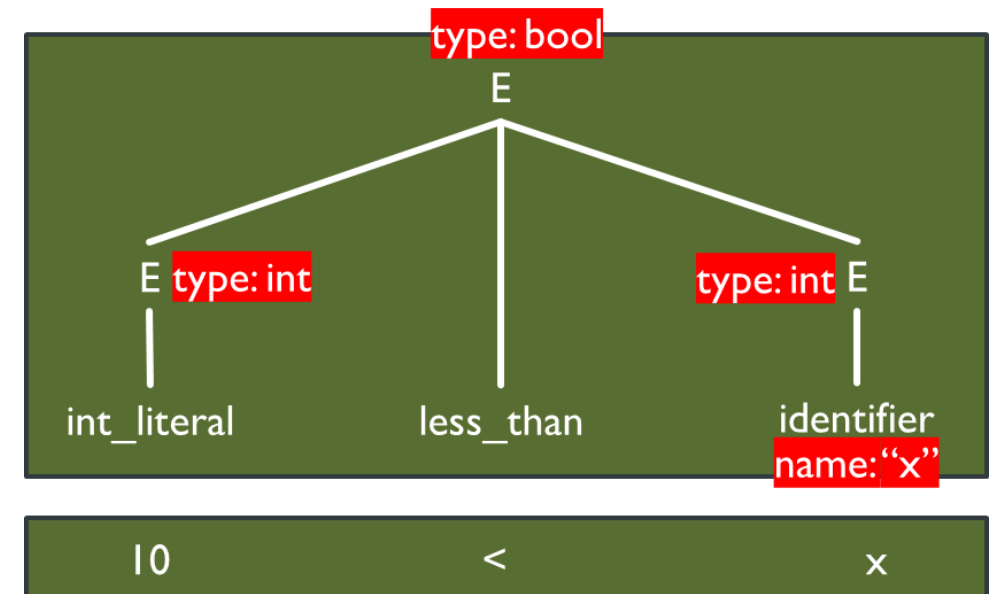
- *Feladat:*  
Alacsonyabb szintű belső reprezentációkra, végül tárgykóddá alakítja a programot
- *Bemenet:*  
Szintaxisfa attribútumokkal, szimbólumtábla
- *Kimenet (az utolsó menetben):*  
Tárgykód
- *Eszközök:*  
Kódgenerálási sémák

# KÓDGENERÁTOR KIMENETE

- Közvetlenül gépi kódot csak nagyon indokolt esetben érdemes generálni
- Helyette assembly kód (pl. valamely platform assembly nyelve vagy LLVM) generálható, amit assemblerekkel fordítunk tovább
- Transzláció: magas szintű nyelvek közötti fordítás
  - Ez lehet végcél, pl. projektek portolása egyik nyelvről a másikra
  - Elterjedt nyelvekre fordítás esetén használhatjuk azok fordítóit a gépi kód / bájtkód előállításához
- **Ebben az előadásban x86-os architektúrára, 32 bites assembly kódot generálunk NASM szintaxissal.**
  - Lásd az előző előadás anyagát az assembly programozásról...

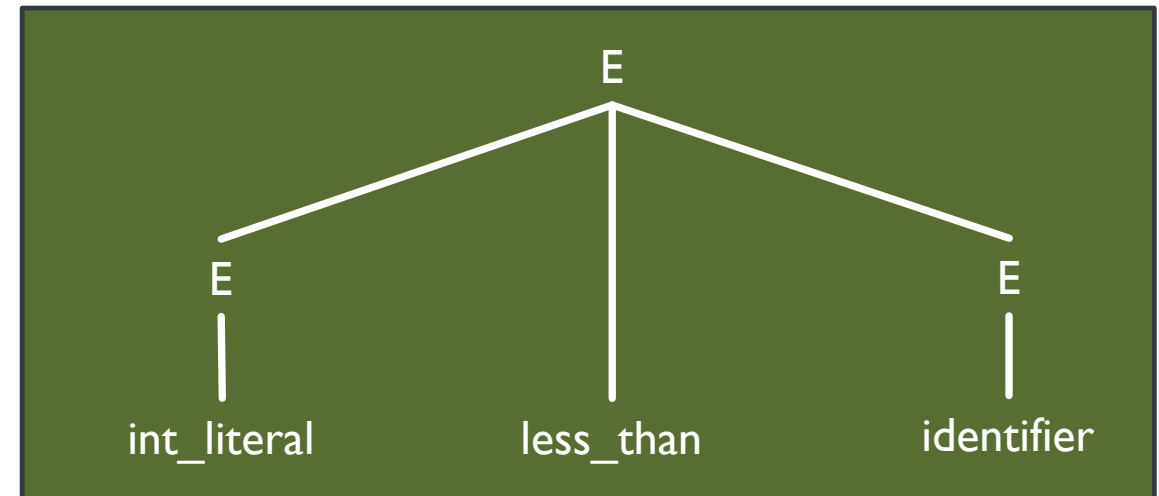
# KÓDGENERÁLÁS ATTRIBÚTUMNYELVTANNAL

- Emlékeztető az attribútumnyelvtanokról:
  - A szintaxist leíró nyelvten szimbólumaihoz *attribútumokat* rendelünk
    - A szemantikus elemzés vagy a kódgenerálás, kódoptimalizálás számára fontos, kiegészítő információk
  - A szabályokhoz *akciókat* (programkód részleteket) rendelünk
    - Meglévő attribútumértékekből újabb attribútumok értékeit számolják ki
    - Ellenőrzéseket végeznek, szemantikus hibákat jeleznek
- Eddig a típusellenőrzéshez használtunk attribútumnyelvtanokat, most a generált kódot is az attribútumokba tesszük.



# PÉLDA

```
E → int_literal {  
  E.type = int;  
}  
  
E → identifier {  
  if identifier.name not in symbol_table then  
    error(...);  
  else  
    E.type = symbol_table.get_type(identifier.name);  
}  
  
E1 → E2 less_than E3 {  
  if E2.type != int || E3.type != int then  
    error(...);  
  else  
    E1.type = bool;  
}
```



10

<

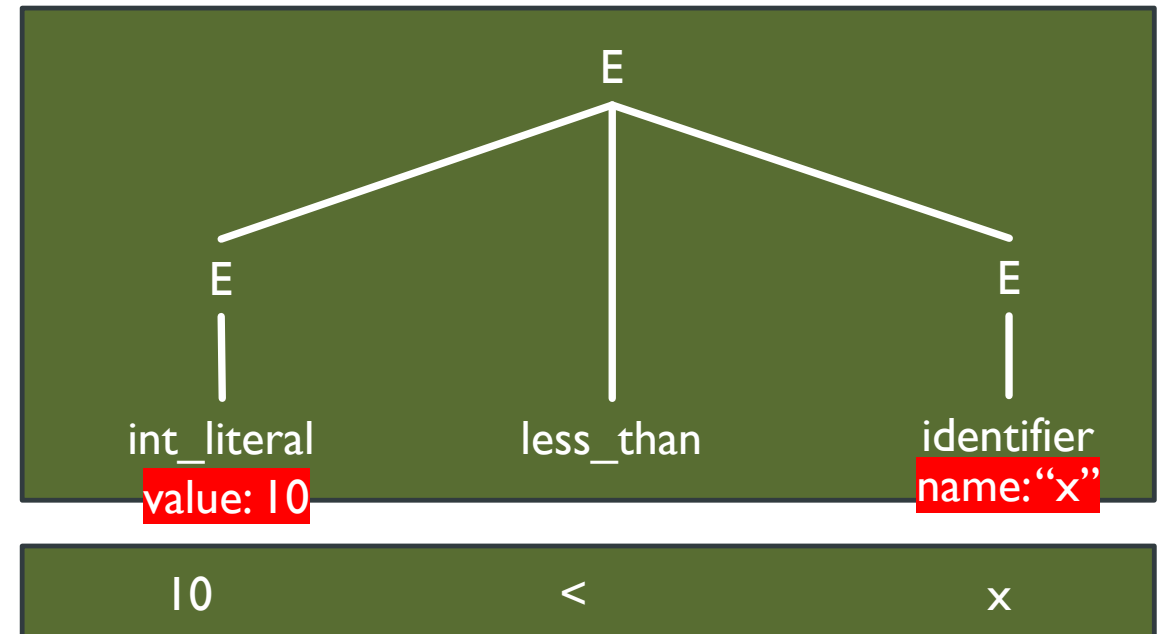
x

## Attribútumok:

- *E.type*: A kifejezés típusa (pl. 'int', 'bool')
- *E.code*: A kifejezés kódja (szöveg)
- *identifier.name*: Az azonosító neve (szöveg)
- *int\_literal.value*: A literál értéke (szám vagy szöveg)

# PÉLDA

```
E → int_literal {  
  E.type = int;  
}  
  
E → identifier {  
  if identifier.name not in symbol_table then  
    error(...);  
  else  
    E.type = symbol_table.get_type(identifier.name);  
}  
  
E1 → E2 less_than E3 {  
  if E2.type != int || E3.type != int then  
    error(...);  
  else  
    E1.type = bool;  
}
```

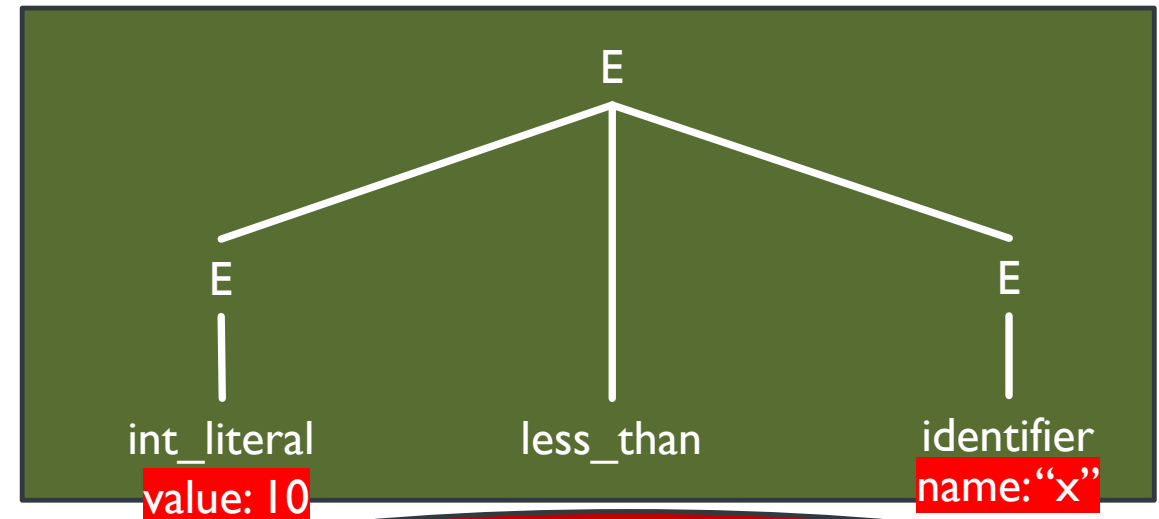


## Attribútumok:

- *E.type*: A kifejezés típusa (pl. 'int', 'bool')
- *E.code*: A kifejezés kódja (szöveg)
- *identifier.name*: Az azonosító neve (szöveg)
- *int\_literal.value*: A literál értéke (szám vagy szöveg)

# PÉLDA

```
E → int_literal {  
  E.type = int;  
}  
  
E → identifier {  
  if identifier.name not in symbol_table then  
    error(...);  
  else  
    E.type = symbol_table.get_type(identifier.name);  
}  
  
E1 → E2 less_than E3 {  
  if E2.type != int || E3.type != int then  
    error(...);  
  else  
    E1.type = bool;  
}
```



Ezeket a lexikális elemző kitölti.  
(„kitüntetett szintetizált attribútumok”)

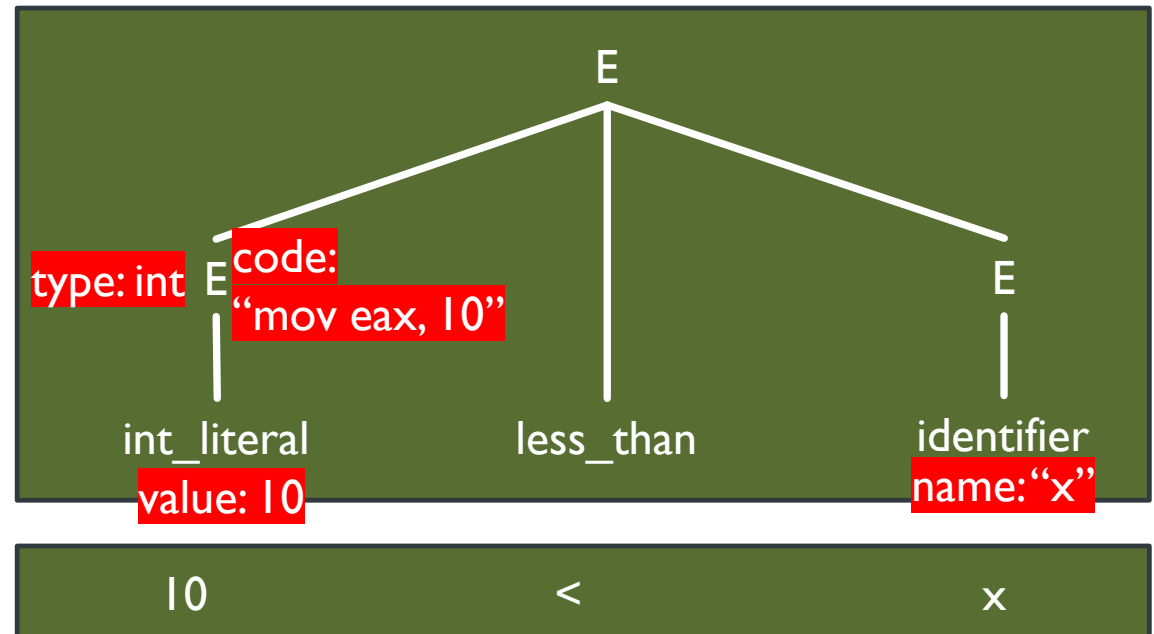
## Attribútumok:

- *E.type*: A kifejezés típusa (pl. ‘int’, ‘bool’)
- *E.code*: A kifejezés kódja (szöveg)
- *identifier.name*: Az azonosító neve (szöveg)
- *int\_literal.value*: A literál értéke (szám vagy szöveg)



# PÉLDA

```
E → int_literal {  
  E.type = int;  
  E.code = "mov eax," + int_literal.value + "\n";  
}  
E → identifier {  
  if identifier.name not in symbol_table then  
    error(...);  
  else  
    E.type = symbol_table.get_type(identifier.name);  
}  
E1 → E2 less_than E3 {  
  if E2.type != int || E3.type != int then  
    error(...);  
  else  
    E1.type = bool;  
}
```

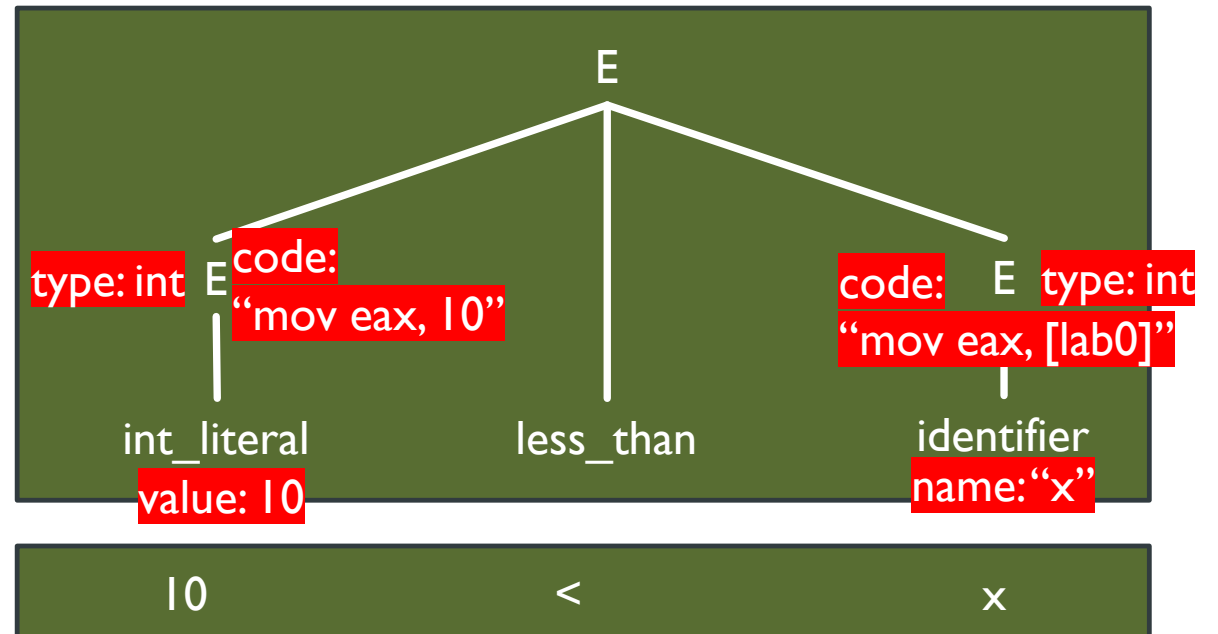


## Attribútumok:

- *E.type*: A kifejezés típusa (pl. 'int', 'bool')
- *E.code*: A kifejezés kódja (szöveg)
- *identifier.name*: Az azonosító neve (szöveg)
- *int\_literal.value*: A literál értéke (szám vagy szöveg)

# PÉLDA

```
E → int_literal {  
  E.type = int;  
  E.code = "mov eax, " + int_literal.value + "\n";  
}  
E → identifier {  
  if identifier.name not in symbol_table then  
    error(...);  
  else  
    E.type = symbol_table.get_type(identifier.name);  
    if E.type == int then  
      E.code = "mov eax, [" + symbol_table.get_label(identifier.name) + "]\n";  
    else if ...  
  }  
E1 → E2 less_than E3 {  
  if E2.type != int || E3.type != int then  
    error(...);  
  else  
    E1.type = bool;  
}  
}
```



## Attribútumok:

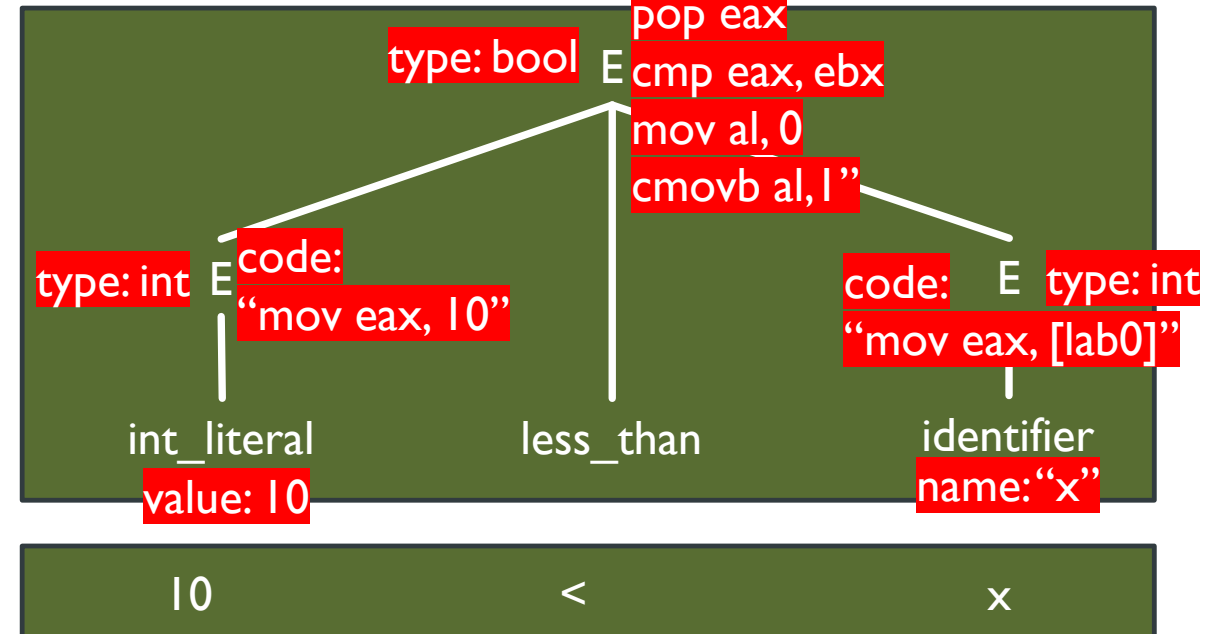
- *E.type*: A kifejezés típusa (pl. 'int', 'bool')
- *E.code*: A kifejezés kódja (szöveg)
- *identifier.name*: Az azonosító neve (szöveg)
- *int\_literal.value*: A literál értéke (szám vagy szöveg)

# PÉLDA

```

E → int_literal {
  E.type = int;
  E.code = "mov eax, " + int_literal.value + "\n";
}
E → identifier {
  if identifier.name not in symbol_table then
    error(...);
  else
    E.type = symbol_table.get_type(identifier.name);
    if E.type == int then
      E.code = "mov eax, [" + symbol_table.get_label(identifier.name) + "]\n";
    else if ...
  }
E1 → E2 less_than E3 {
  if E2.type != int || E3.type != int then
    error(...);
  else
    E1.type = bool;
    E1.code = E2.code + "push eax\n" + E3.code + "mov ebx, eax\n" +
      "pop eax\n" + "cmp eax, ebx\n" + "mov al, 0\n" + "cmovb al, 1\n";
  }

```



## Attribútumok:

- *E.type*: A kifejezés típusa (pl. 'int', 'bool')
- *E.code*: A kifejezés kódja (szöveg)
- *identifier.name*: Az azonosító neve (szöveg)
- *int\_literal.value*: A literál értéke (szám vagy szöveg)

# KÓDGENERÁLÁS ATTRIBÚTUMNYELVTANOKKAL

- Emlékeztető:
  - „Az akciókban csak az adott szabályban szereplő szimbólumok attribútumai olvashatók és írhatók.”
- A nyelvtani szabályokhoz kódgenerálási szabályokat (kódgenerálási sémákat) fogunk rendelni.
  - Ezek feltételezik, hogy a jobb oldal kódjai már készen állnak.
  - Ezekből és plusz utasításokból állítjuk össze a bal oldal kódját.

# SZÁMLITERÁL KIFEJEZÉS KÓDGENERÁLÁSI SÉMÁJA

- A számokat 32 bites, előjeles egész számként fogjuk tárolni.
- Minden szám típusú kifejezés eredményét az **eax** regiszterbe fogjuk kiszámolni.

$E \rightarrow \text{int\_literal}$

**mov eax, *literál értéke***

# LOGIKAI LITERÁL KIFEJEZÉS KÓDGENERÁLÁSI SÉMÁJA

- A logikai értékeket 1 bájton fogjuk tárolni. Adatreprezentáció:
  - false: 0
  - true: 1
- Minden logikai típusú kifejezés eredményét az **al** regiszterbe fogjuk kiszámolni.

$E \rightarrow \text{false}$

**mov al, 0**

$E \rightarrow \text{true}$

**mov al, 1**

# VÁLTOZÓ MINT KIFEJEZÉS KÓDGENERÁLÁSI SÉMÁJA

- A változók adatait a szimbólumtáblában tároljuk.
- Mindegyikhez egy egyedi címkét generálunk, amikor betesszük őket a táblába.
- A változó használatakor kiolvassuk a címkét, és beépítjük a generált kódba.

Név	Fajta	Típus	Címke
“v”	változó	int	label0

Egész szám:

$E \rightarrow \text{identifier}$

**mov eax, [címké]**

Logikai:

$E \rightarrow \text{identifier}$

**mov al, [címké]**

# BEÉPÍTETT OPERÁTOROK KÓDGENERÁLÁSI SÉMÁJA

Példa: összeadás

$E_1 \rightarrow E_2 \text{ plus\_op } E_3$

$E_2$  kódja  
push eax  
 $E_3$  kódja  
mov ecx, eax  
pop eax  
add eax, ecx

Példa: szorzás

$E_1 \rightarrow E_2 \text{ mul\_op } E_3$

$E_2$  kódja  
push eax  
 $E_3$  kódja  
mov ecx, eax  
pop eax  
mul ecx

Példa: kisebb operátor

$E_1 \rightarrow E_2 \text{ less\_op } E_3$

$E_2$  kódja  
push eax  
 $E_3$  kódja  
mov ecx, eax  
pop eax  
cmp eax, ecx  
mov al, 0  
cmovb al, 1

Példa: logikai „és”

$E_1 \rightarrow E_2 \text{ and\_op } E_3$

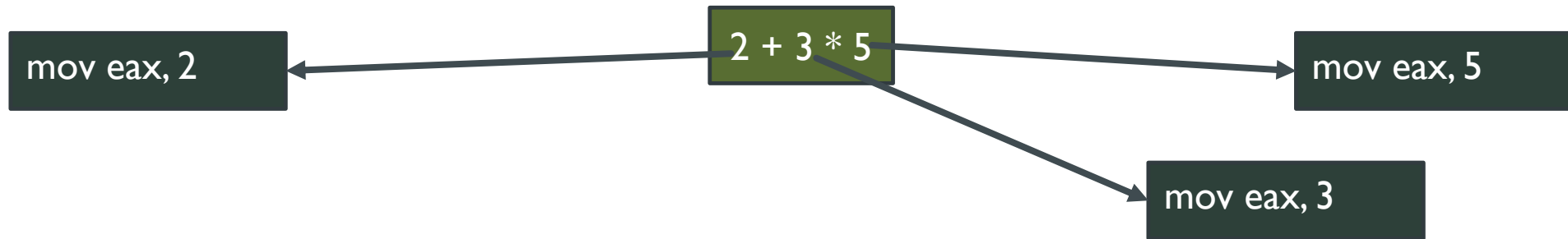
$E_2$  kódja  
push ax  
 $E_3$  kódja  
mov cx, ax  
pop ax  
and al, cl



PÉLDA

$$2 + 3 * 5$$

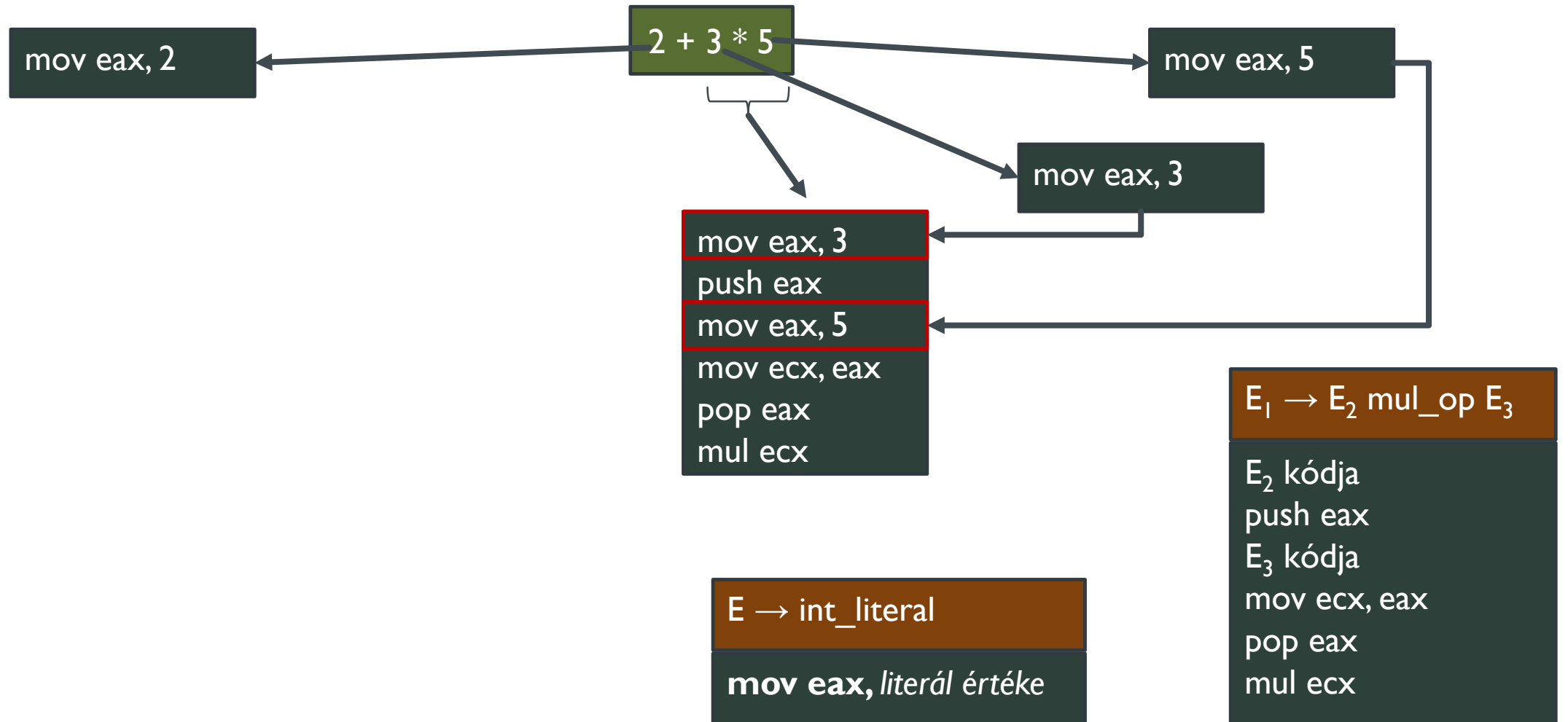
# PÉLDA



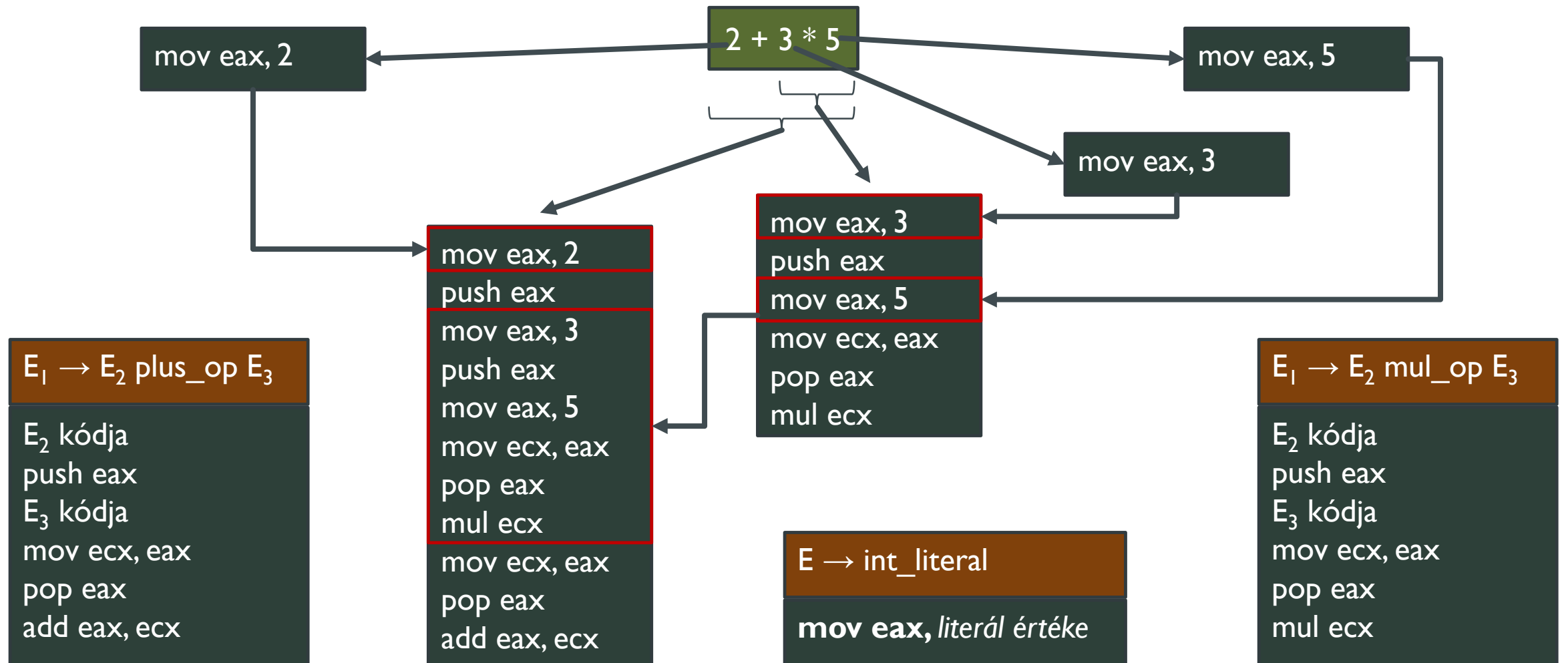
$E \rightarrow \text{int\_literal}$

`mov eax, literál értéke`

# PÉLDA



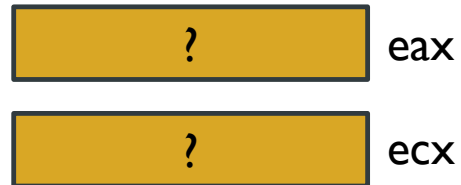
# PÉLDA



# PÉLDA

$2 + 3 * 5$

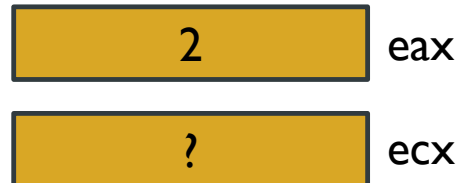
→  
mov eax, 2  
push eax  
mov eax, 3  
push eax  
mov eax, 5  
mov ecx, eax  
pop eax  
mul ecx  
mov ecx, eax  
pop eax  
add eax, ecx



# PÉLDA

$2 + 3 * 5$

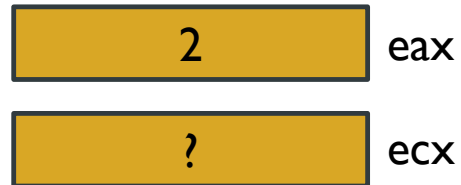
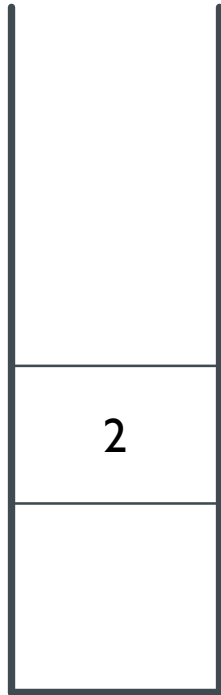
→  
mov eax, 2  
push eax  
mov eax, 3  
push eax  
mov eax, 5  
mov ecx, eax  
pop eax  
mul ecx  
mov ecx, eax  
pop eax  
add eax, ecx



# PÉLDA

$2 + 3 * 5$

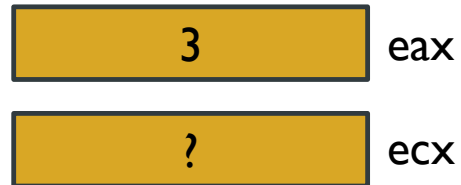
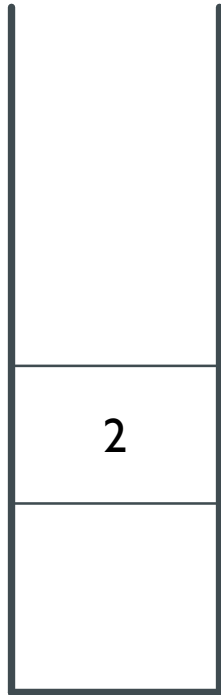
→  
mov eax, 2  
push eax  
mov eax, 3  
push eax  
mov eax, 5  
mov ecx, eax  
pop eax  
mul ecx  
mov ecx, eax  
pop eax  
add eax, ecx



# PÉLDA

$2 + 3 * 5$

→  
mov eax, 2  
push eax  
mov eax, 3  
push eax  
mov eax, 5  
mov ecx, eax  
pop eax  
mul ecx  
mov ecx, eax  
pop eax  
add eax, ecx

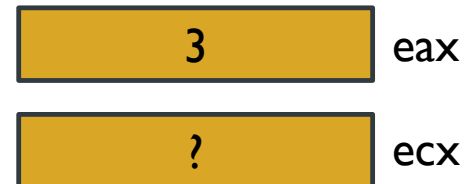
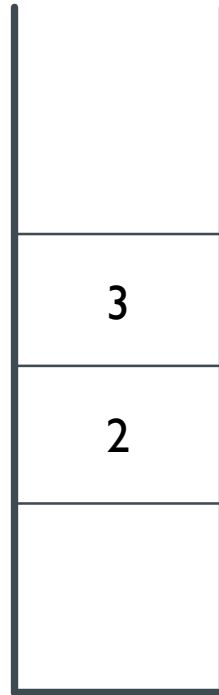




# PÉLDA

$2 + 3 * 5$

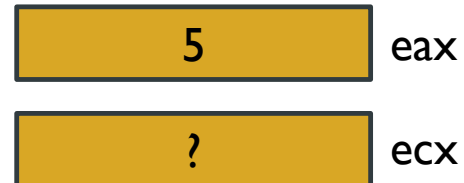
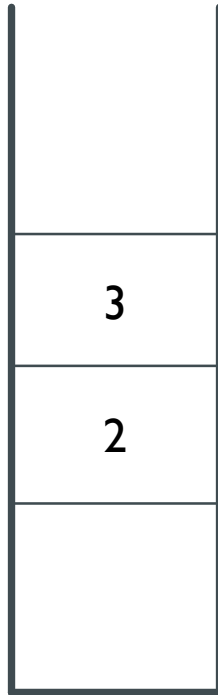
→  
mov eax, 2  
push eax  
mov eax, 3  
push eax  
mov eax, 5  
mov ecx, eax  
pop eax  
mul ecx  
mov ecx, eax  
pop eax  
add eax, ecx



# PÉLDA

$2 + 3 * 5$

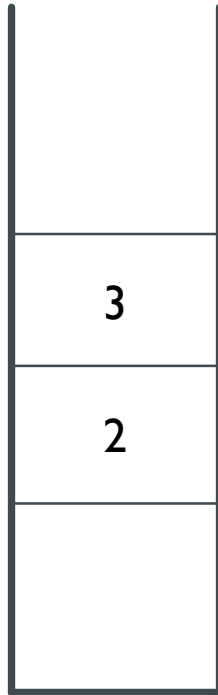
→  
mov eax, 2  
push eax  
mov eax, 3  
push eax  
mov eax, 5  
mov ecx, eax  
pop eax  
mul ecx  
mov ecx, eax  
pop eax  
add eax, ecx



# PÉLDA


$2 + 3 * 5$

→  
mov eax, 2  
push eax  
mov eax, 3  
push eax  
mov eax, 5  
mov ecx, eax  
pop eax  
mul ecx  
mov ecx, eax  
pop eax  
add eax, ecx

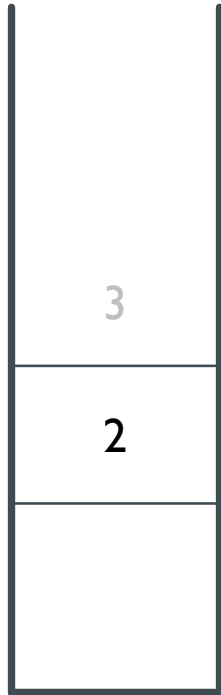


# PÉLDA

$2 + 3 * 5$



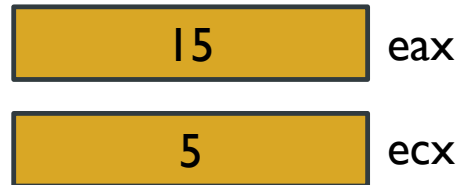
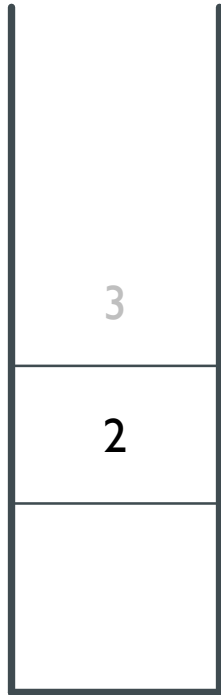
```
mov eax, 2  
push eax  
mov eax, 3  
push eax  
mov eax, 5  
mov ecx, eax  
pop eax  
mul ecx  
mov ecx, eax  
pop eax  
add eax, ecx
```



# PÉLDA

$2 + 3 * 5$

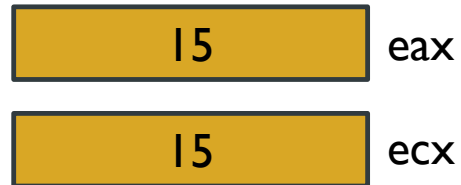
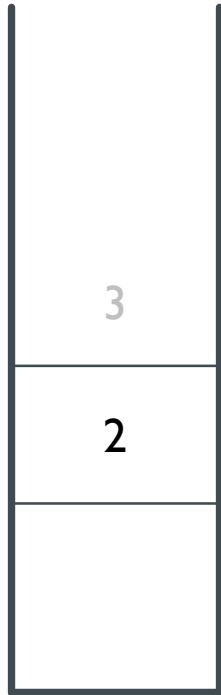
```
mov eax, 2  
push eax  
mov eax, 3  
push eax  
mov eax, 5  
mov ecx, eax  
pop eax  
mul ecx  
mov ecx, eax  
pop eax  
add eax, ecx
```



# PÉLDA

$2 + 3 * 5$

```
mov eax, 2  
push eax  
mov eax, 3  
push eax  
mov eax, 5  
mov ecx, eax  
pop eax  
mul ecx  
mov ecx, eax  
pop eax  
add eax, ecx
```



# PÉLDA

$2 + 3 * 5$

```
mov eax, 2  
push eax  
mov eax, 3  
push eax  
mov eax, 5  
mov ecx, eax  
pop eax  
mul ecx  
mov ecx, eax  
pop eax  
add eax, ecx
```

3

2


2

eax

15

ecx

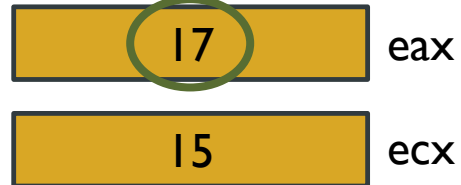
# PÉLDA



```
mov eax, 2  
push eax  
mov eax, 3  
push eax  
mov eax, 5  
mov ecx, eax  
pop eax  
mul ecx  
mov ecx, eax  
pop eax  
add eax, ecx
```



$2 + 3 * 5$





# ÉRTÉKADÁS KÓDGENERÁLÁSI SÉMÁJA

Egész szám:

$U \rightarrow \text{identifier assign\_op } E$

*E kódja*

**mov [címké], eax**

Logikai:

$U \rightarrow \text{identifier assign\_op } E$

*E kódja*

**mov [címké], al**

- A változó címkéjét itt is a szimbólumtáblából vesszük.

# ELÁGAZÁS KÓDGENERÁLÁSI SÉMÁJA

Egy ágú:

$U \rightarrow \text{if } E \text{ then } P \text{ end}$

*E kódja*

**cmp al, l**

**jne near vége**

*P kódja*

**vége:**

Két ágú:

$U \rightarrow \text{if } E \text{ then } P_1 \text{ else } P_2 \text{ end}$

*E kódja*

**cmp al, l**

**jne near hamis**

*P<sub>1</sub> kódja*

**jmp vége**

*hamis:*

*P<sub>2</sub> kódja*

**vége:**

- A kódgenerálási sémában szereplő címkék (pl. *vége*, *hamis*) helyett a séma minden felhasználásakor egyedi címkéket kell generálni (pl. **label0**, **label1**, **label2**, ...)

# CIKLUS KÓDGENERÁLÁSI SÉMÁJA

Elöltesztelő:

$U \rightarrow \text{while } E \text{ do } P \text{ done}$

*eleje:*  
*E kódja*  
**cmp al, l**  
**jne near vége**  
*P kódja*  
**jmp eleje**  
*vége:*

Hátultesztelő:

$U \rightarrow \text{do } P \text{ while } E$

*eleje:*  
*P kódja*  
*E kódja*  
**cmp al, l**  
**je near eleje**

# PÉLDA

Név	Fajta	Típus	Címke
“b”	változó	bool	lab0
“c”	változó	bool	lab1

```
while b do  
  if c then  
    b := false  
  endif  
done
```

# PÉLDA

Név	Fajta	Típus	Címke
"b"	változó	bool	lab0
"c"	változó	bool	lab1

```
while b do  
  if c then  
    b := false  
  endif  
done
```

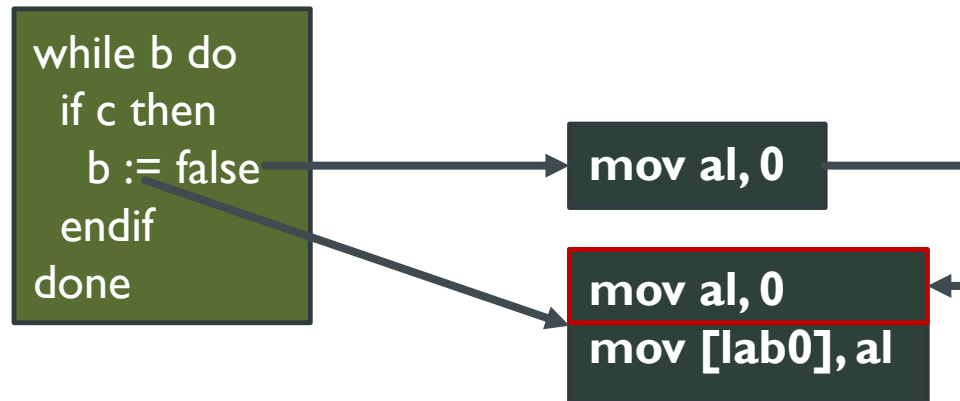
→ **mov al, 0**

**E → false**

**mov al, 0**

# PÉLDA

Név	Fajta	Típus	Címke
"b"	változó	bool	lab0
"c"	változó	bool	lab1

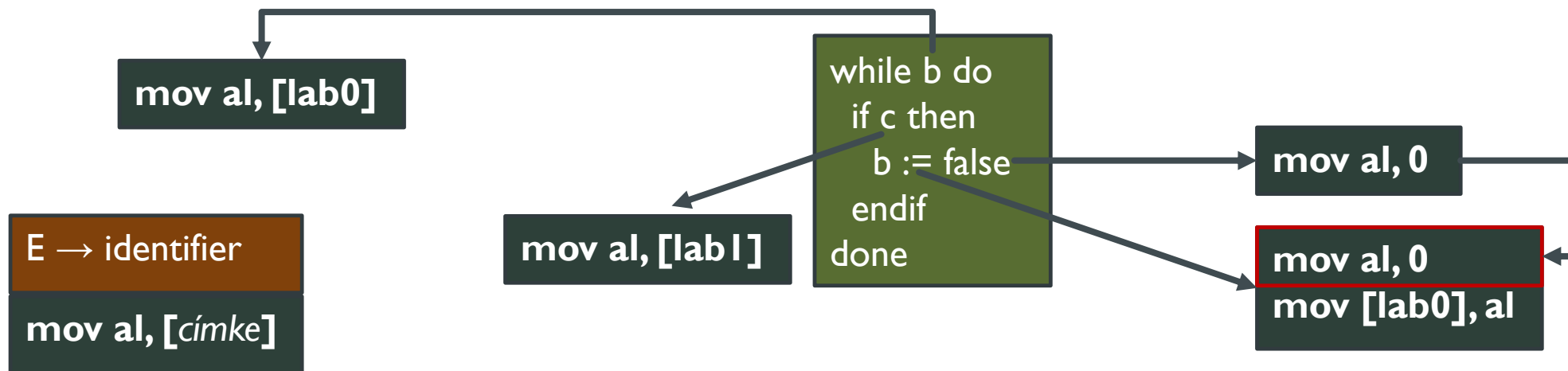


$U \rightarrow \text{identifier assign\_op } E$

*E kódja*  
`mov [címke], al`

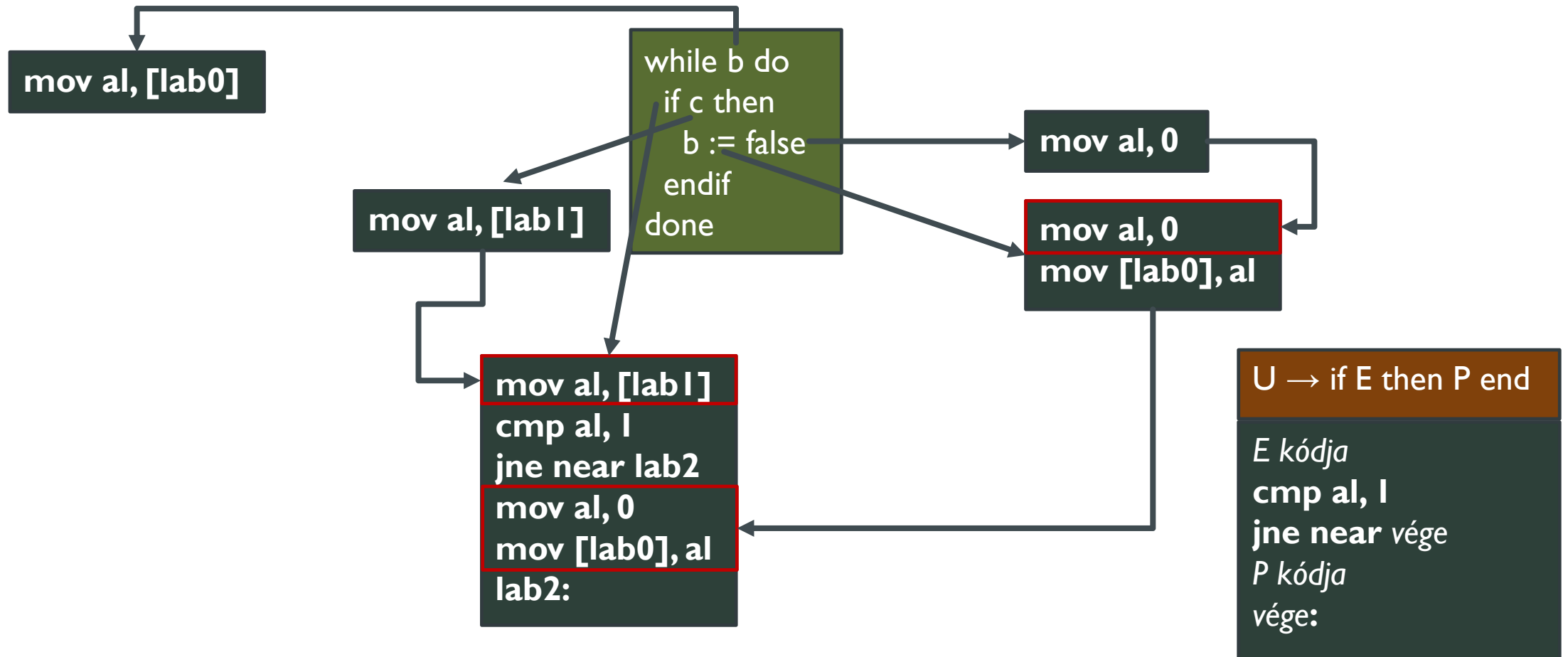
# PÉLDA

Név	Fajta	Típus	Címke
"b"	változó	bool	lab0
"c"	változó	bool	lab1



# PÉLDA

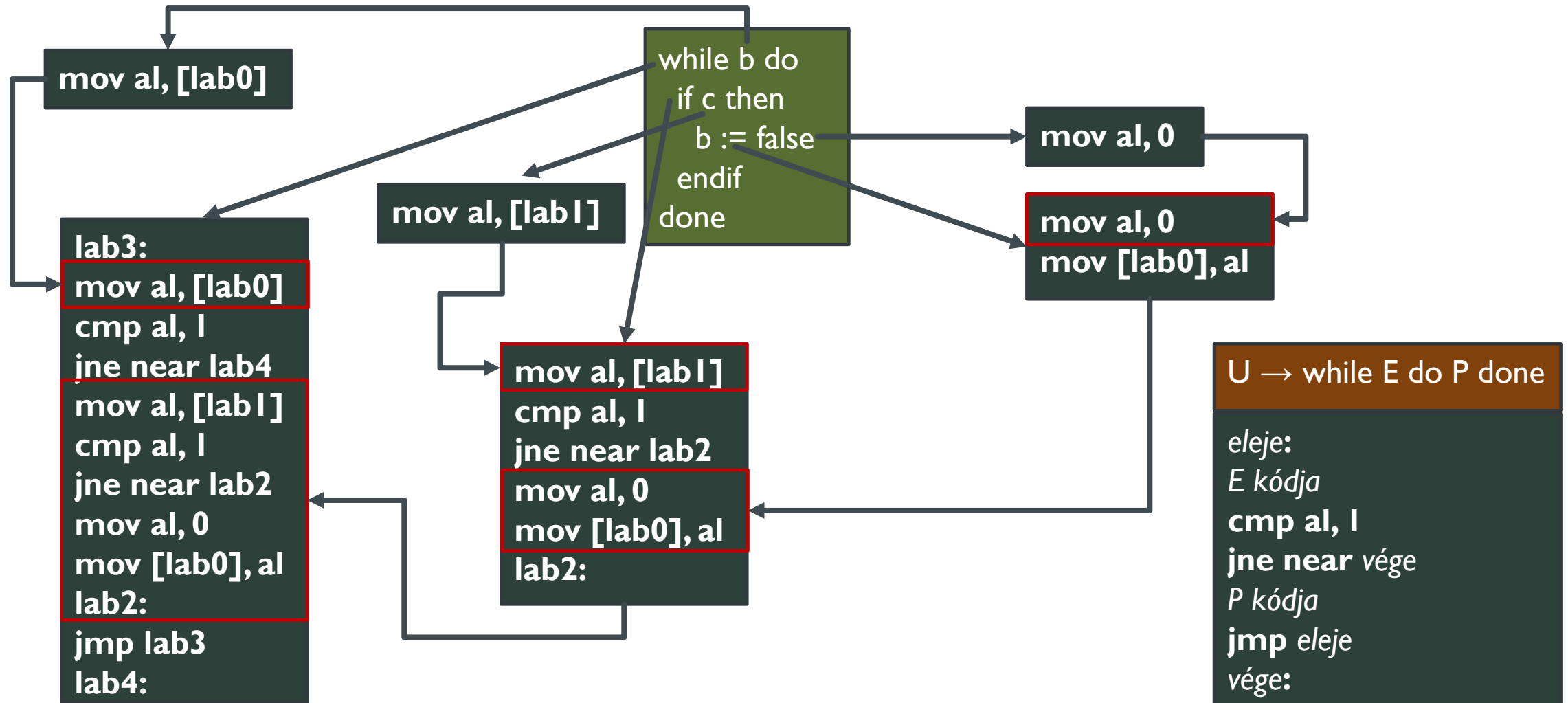
Név	Fajta	Típus	Címke
"b"	változó	bool	lab0
"c"	változó	bool	lab1





# PÉLDA

Név	Fajta	Típus	Címke
"b"	változó	bool	lab0
"c"	változó	bool	lab1



# PÉLDA

```
while b do  
  if c then  
    b := false  
  endif  
done
```



```
lab3:  
mov al, [lab0]  
cmp al, 1  
jne near lab4  
mov al, [lab1]  
cmp al, 1  
jne near lab2  
mov al, 0  
mov [lab0], al  
lab2:  
jmp lab3  
lab4:
```

? al

? zero flag

lab0	lab1
1	1

# PÉLDA

```
while b do  
  if c then  
    b := false  
  endif  
done
```



```
lab3:  
mov al, [lab0]  
cmp al, 1  
jne near lab4  
mov al, [lab1]  
cmp al, 1  
jne near lab2  
mov al, 0  
mov [lab0], al  
lab2:  
jmp lab3  
lab4:
```

1 al

lab0 lab1  
1 1

? zero flag

# PÉLDA

```
while b do  
  if c then  
    b := false  
  endif  
done
```



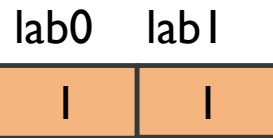
```
lab3:  
  mov al, [lab0]  
  cmp al, 1  
  jne near lab4  
  mov al, [lab1]  
  cmp al, 1  
  jne near lab2  
  mov al, 0  
  mov [lab0], al  
lab2:  
  jmp lab3  
lab4:
```



al



zero flag



lab0

lab1

# PÉLDA

```
while b do
  if c then
    b := false
  endif
done
```



```
lab3:
mov al, [lab0]
cmp al, 1
jne near lab4
mov al, [lab1]
cmp al, 1
jne near lab2
mov al, 0
mov [lab0], al
lab2:
jmp lab3
lab4:
```

| al

| zero flag

lab0	lab1

# PÉLDA

```
while b do  
  if c then  
    b := false  
  endif  
done
```



```
lab3:  
  mov al, [lab0]  
  cmp al, 1  
  jne near lab4  
  mov al, [lab1]  
  cmp al, 1  
  jne near lab2  
  mov al, 0  
  mov [lab0], al  
lab2:  
  jmp lab3  
lab4:
```

| al

| zero flag

lab0	lab1

# PÉLDA

```
while b do  
  if c then  
    b := false  
  endif  
done
```



```
lab3:  
  mov al, [lab0]  
  cmp al, 1  
  jne near lab4  
  mov al, [lab1]  
  cmp al, 1  
  jne near lab2  
  mov al, 0  
  mov [lab0], al  
lab2:  
  jmp lab3  
lab4:
```

| al

| zero flag

lab0	lab1

# PÉLDA

```
while b do  
  if c then  
    b := false  
  endif  
done
```



```
lab3:  
mov al, [lab0]  
cmp al, 1  
jne near lab4  
mov al, [lab1]  
cmp al, 1  
jne near lab2  
mov al, 0  
mov [lab0], al  
lab2:  
jmp lab3  
lab4:
```

| al

| zero flag

lab0	lab1



# PÉLDA

```
while b do  
  if c then  
    b := false  
  endif  
done
```



```
lab3:  
mov al, [lab0]  
cmp al, 1  
jne near lab4  
mov al, [lab1]  
cmp al, 1  
jne near lab2  
mov al, 0  
mov [lab0], al  
lab2:  
jmp lab3  
lab4:
```

0 al

1 zero flag

lab0	lab1
1	1

# PÉLDA

```
while b do  
  if c then  
    b := false  
  endif  
done
```

```
lab3:  
  mov al, [lab0]  
  cmp al, 1  
  jne near lab4  
  mov al, [lab1]  
  cmp al, 1  
  jne near lab2  
  mov al, 0  
  mov [lab0], al  
lab2:  
  jmp lab3  
lab4:
```



0 al

1 zero flag

lab0	lab1
0	1

# PÉLDA

```
while b do  
  if c then  
    b := false  
  endif  
done
```



```
lab3:  
mov al, [lab0]  
cmp al, 1  
jne near lab4  
mov al, [lab1]  
cmp al, 1  
jne near lab2  
mov al, 0  
mov [lab0], al  
lab2:  
jmp lab3  
lab4:
```

0 al

1 zero flag

lab0	lab1
0	1

# PÉLDA

```
while b do  
  if c then  
    b := false  
  endif  
done
```



```
lab3:  
mov al, [lab0]  
cmp al, 1  
jne near lab4  
mov al, [lab1]  
cmp al, 1  
jne near lab2  
mov al, 0  
mov [lab0], al  
lab2:  
jmp lab3  
lab4:
```

0 al

1 zero flag

lab0	lab1
0	1

# PÉLDA

```
while b do  
  if c then  
    b := false  
  endif  
done
```



```
lab3:  
mov al, [lab0]  
cmp al, 1  
jne near lab4  
mov al, [lab1]  
cmp al, 1  
jne near lab2  
mov al, 0  
mov [lab0], al  
lab2:  
jmp lab3  
lab4:
```

0 al

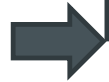
lab0	lab1
0	1

0 zero flag

# PÉLDA

```
while b do  
  if c then  
    b := false  
  endif  
done
```

```
lab3:  
  mov al, [lab0]  
  cmp al, 1  
  jne near lab4  
  mov al, [lab1]  
  cmp al, 1  
  jne near lab2  
  mov al, 0  
  mov [lab0], al  
lab2:  
  jmp lab3  
lab4:
```



0 al

0 zero flag

lab0	lab1
0	1

# SZIMBÓLUMTÁBLA LEKÉPEZÉSE ASSEMBLY KÓDDÁ

- Végigiterálunk a szimbólumtábla bejegyzésein
- Minden bejegyzésből egy-egy címkét és hozzá tartozó memórafoglalást generálunk
- A típus határozza meg a foglalt memóraf méretét
- Ezek a **.bss** szakaszba kerülnek (kezdőérték nélküli változók)

Név	Fajta	Típus	Címke
"b"	változó	bool	lab0
"i"	változó	int	lab1



```
section .bss  
lab0: resb 1  
lab1: resd 1
```

# A TELJES PROGRAM SÉMÁJA

```
global main  
extern külső címkék
```

```
section .bss  
változók
```

```
section .text  
main:  
programutasítások  
ret
```