Branches and Binary Operators

BOA: Branches and Binary Operators

Next, lets add

- Branches (`if` -expressions)
- Binary Operators (`+` , `-` , etc.)

In the process of doing so, we will learn about

- **Intermediate Forms**
- **Normalization**

Branches

TRUE = NON-ZERO
FALSE = 0

Lets start first with branches (conditionals).

We will stick to our recipe of:

1. Build intuition with **examples**,
2. Model problem with **types**,
3. Implement with **type-transforming-functions**,
4. Validate with **tests**.

Examples

First, lets look at some examples of what we mean by branches.

- For now, lets treat `0` as "false" and non-zero as "true"

Example: `If1`

```
if 10:
  22
else:
  sub1(0)
```

→ 22

- Since `10` is *not* `0` we evaluate the "then" case to get `22`

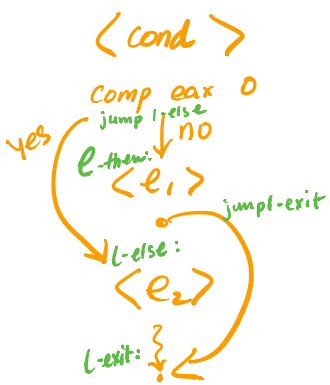
Example: `If2`

```
if sub(1):
  22
else:
  sub1(0)
```

=0

→ -1

if cond:
  e_1
else:
  e_2



- Since `sub(1)` *is* `0` we evaluate the "else" case to get `-1`

QUIZ: `If3`

`if-else` is also an *expression* so we can nest them:

What should the following evaluate to?

```
let x = if sub(1):
          22
        else:
          sub1(0)
in
  if x:
    add1(x)
  else:
    999
```

- **A.** 999
- **B.** 0
- **C.** 1
- **D.** 1000
- **E.** -1

Control Flow in Assembly

To compile branches, we will use **labels**, **comparisons** and **jumps**

Labels

```
our_code_label:
  ...
```

Labels are *"landmarks"* - from which execution (control-flow) can be *started*, or - to which it can be *diverted*

Comparisons

`cmp a1, a2`

- Perform a (numeric) **comparison** between the values `a1` and `a2`, and
- Store the result in a special **processor flag**

Jumps

```
jmp LABEL     # jump unconditionally (i.e. always)
je  LABEL     # jump if previous comparison result was EQUAL
jne LABEL     # jump if previous comparison result was NOT-EQUAL
```

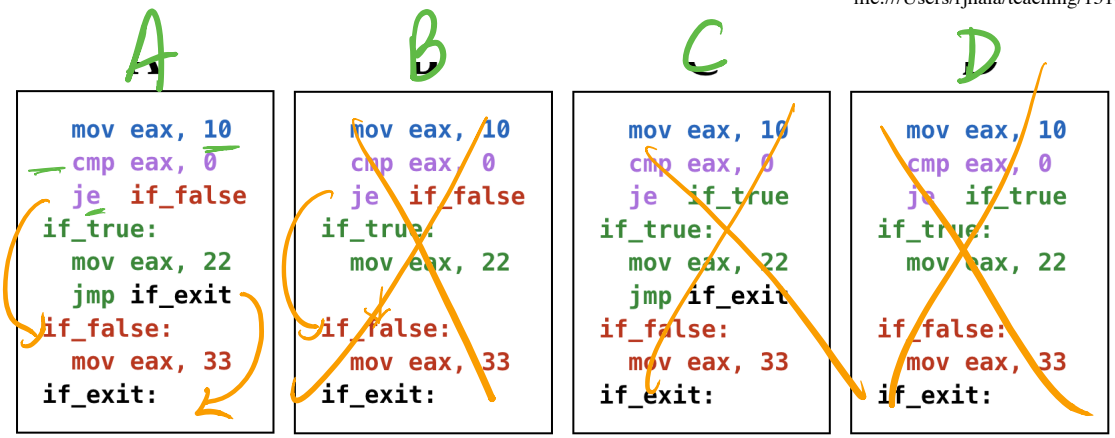
Use the result of the **flag** set by the most recent `cmp *` To *continue execution* from the given `LABEL`

QUIZ

Which of the following is a valid x86 encoding of

```
if 10:
  22
else
  33
```

A B C D

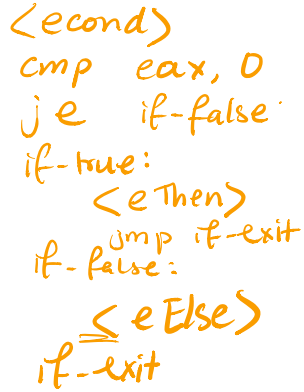


QUIZ: Compiling if-else

Strategy

To compile an expression of the form

```
if eCond:
  eThen
else:
  eElse
```



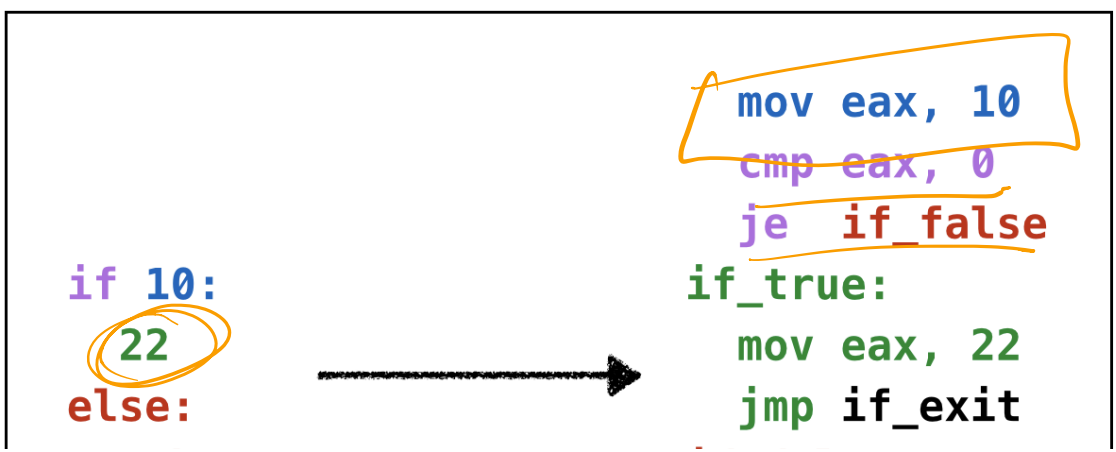
We will:

1. Compile `eCond`
2. Compare the result (in `eax`) against `0`
3. Jump if the result *is zero* to a **special** `"IfFalse"` label
   - At which we will evaluate `eElse`,
   - Ending with a special `"IfExit"` label.
4. (Otherwise) continue to evaluate `eTrue`
   - And then jump (unconditionally) to the `"IfExit"` label.

Example: If-Expressions to Asm

Lets see how our strategy works by example:

Example: if1



```
sub1(0)
```

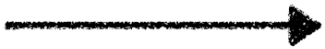
```
if_false:
  mov eax, 0
  sub eax, 1
if_exit:
```

Example: if1

Example: if2

```
mov eax, 1
sub eax, 1
cmp eax, 0
je  if_false
```

```
if sub(1):
  22
else:
  sub1(0)
```



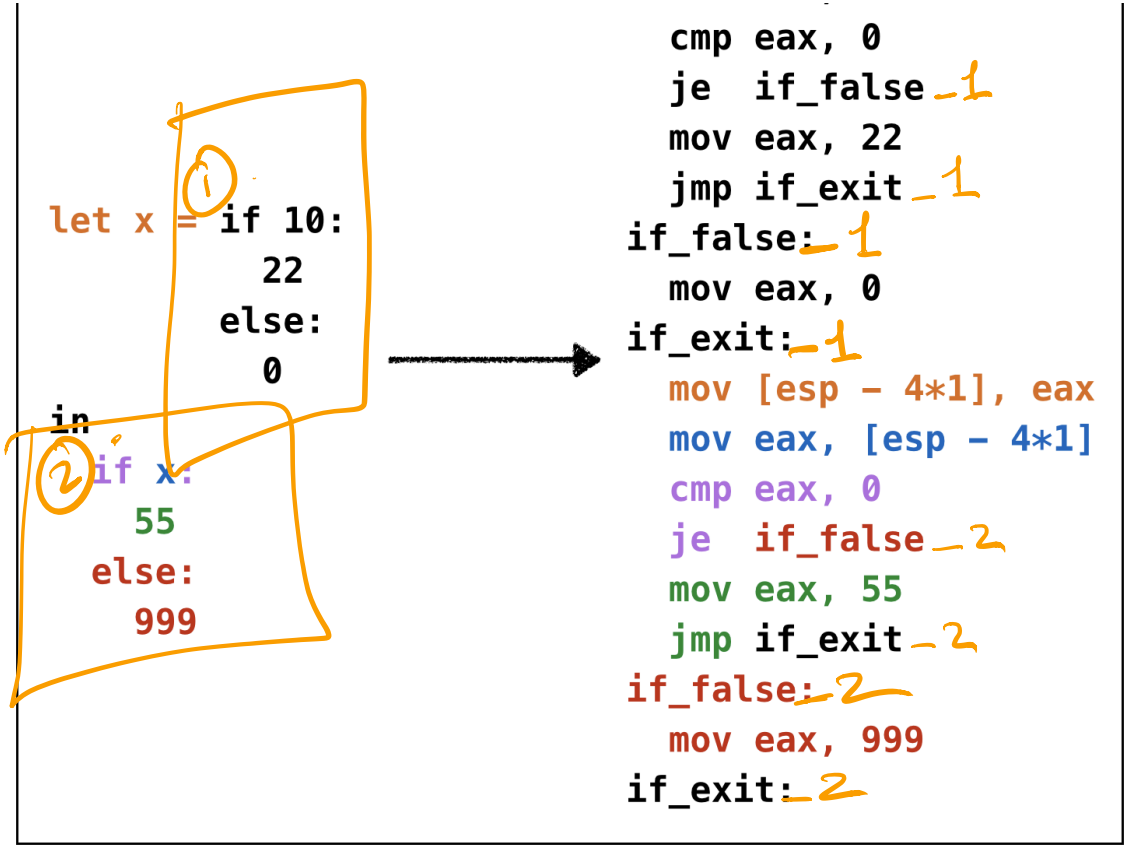
```
if_true:
  mov eax, 22
  jmp if_exit
if_false:
  mov eax, 0
  sub eax, 1
if_exit:
```

Example: if2

Example: if3

ASM

```
mov eax, 10
```

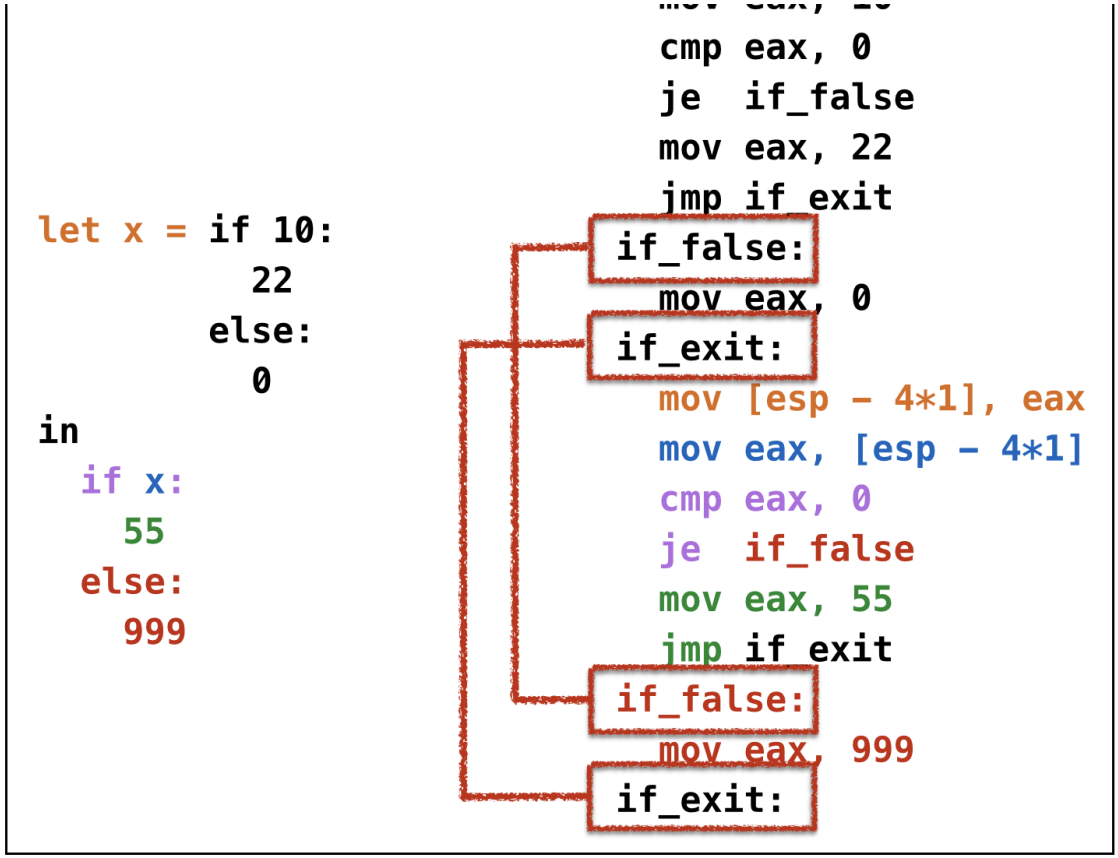


Example: if3

Oops, **cannot reuse labels** across if-expressions!

- Can't use same label in two places (invalid assembly)

```
                                    mov eax, 10
```



Example: if3 wrong

Oops, need **distinct labels** for each branch!

- Require **distinct tags** for each `if-else` expression

```
let x = if 10:
        1 22
        else:
          0
in
  if x:
  2 55
  else:
    999
```

```
  mov eax, 10
  cmp eax, 0
  je  if_1_false
  mov eax, 22
  jmp if_1_exit
if_1_false:
  mov eax, 0
if_1_exit:
  mov [esp - 4*1], eax
  mov eax, [esp - 4*1]
  cmp eax, 0
  je  if_2_false
  mov eax, 55
  jmp if_2_exit
if_2_false:
  mov eax, 999
if_2_exit:
```

Example: if3 tagged