Code generation and control flow

Essentials for recursively code generation in Cool

- conditional code generation
- match expression including `null` arm
- type checking

Conditional Expression

```
  beqz $a0, false   # if false jump to false label
                    # otherwise continue
true:
  # then arm of conditional

  b done            # branch unconditionally to done
false:
  # else arm of conditional

done:
  # end of conditional
```

Trick question

`beqz` is a macro MIPS instruction. What is the actual MIPS instruction when macro gets expanded?

Answer

`beq $a0, $zero, false` register `$zero` which is an alias for `$0` holds the value of zero and it cannot change

There is an another way to write this conditional

> Hint: use `bne` instruction

Match Expression

Remember pattern mach expression can have null arm but it is optional.

- If `null` arm is missing, it will get added by the compiler. You may be wondering why null arm is needed?
  - The reasons is all other arms expect value to not be null.
  - In the cool-manual it specifically says that if value is null and no null branch is present then we should pass the control to runtime exception.

Match Expression (Cont.)

We already did a semantic check which means type of each arm of the branch is less than previous (or $\mathrm{T1} < \mathrm{T2}$)

```
e match {
    case t1: T1 => { }
    case t2: T2 => { }
    case null =>   { }
}
```

For code generation we assume branch arms do not overlap so we can write the code for arm sequentially. However, we need to write the code for `null` arm first.

Basic flow

- if `null` arm exist then check if expr is `null` and branch to there
  - if it does not exist then create `null` branch that always jumps to runtime error
- for each branch arm compare static type of the branch and type of expression object and see if object "fits" otherwise move on to the next arm.
- when each branch arm is finished branch unconditionally to the end
- remember to write the code for branch expression just visit `b.get_expr().accept(this)`

Type testing

Remember each object at offset 0 (or `i_TAGOFFSET`) has the class tag number. That is useful for type checking for `Ctypecase`