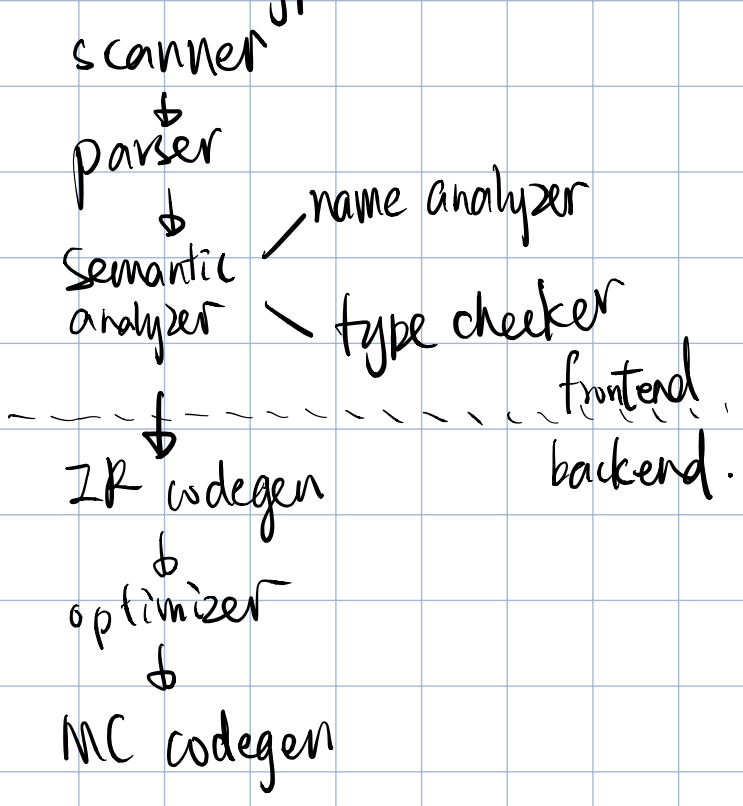
Lec15 Types



- type coercion: implicit cast from one data type to another
- Static Typing: compile-time type check
  dynamic typing: run-time type check
  Combination of two: Java
- Strong vs. Weak typing
- Type error reporting: Avoid cascading errors.
  When type incompatibility discovered, report error, pass it up the tree;
  When get an error as operand: Don't (re) report, pass up the tree.

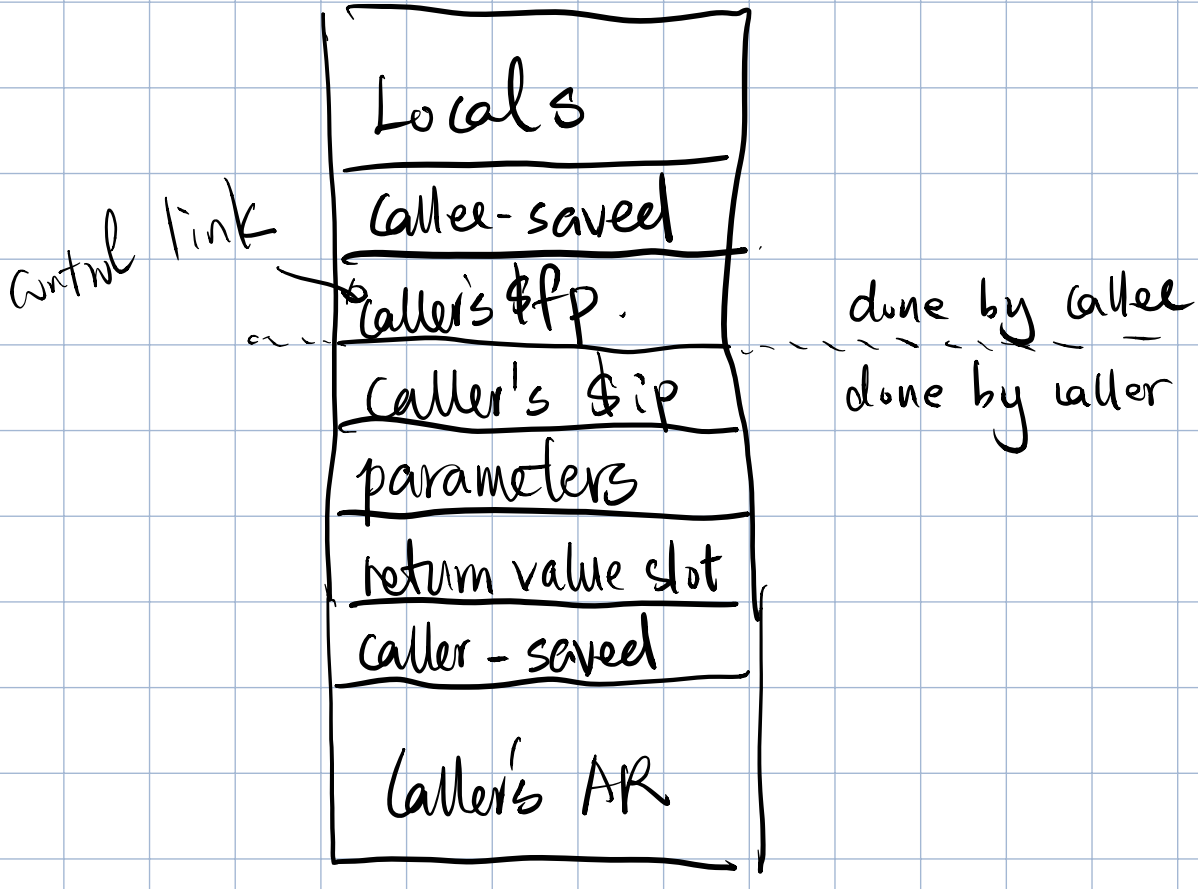
Lec16 Runtime Environment

- Static allocation of memory doesn't allow recursion.
- Stack + Heap.

  Static data, especially strings, are put in the global area to keep the stack small.

  Stack grows/shrinks at runtime and can handle variable whose size is unknown at compile-time. So stack size is unknown at compile-time.

  $sp: next empty slot on top of stack; $fp: base of the frame.



- Function Code Generation
  - Function Entry: Caller responsibility
    Store caller-saved registers;
    Push parameters: ① set slot for return value; ② push args
    Push return address;
    Jump to callee's first instruction.
  - Function Entry: Callee responsibility
    Save caller's $fp, aka. "control link".
    Set new $fp.
    Save callee-saved registers.
    Make space for locals
  - Function exit: Callee responsibility
    Set return value
    Restore callee-saved registers
    Grab stored $ip
    restore $fp
    restore $sp
    Jump back to caller
  - Function Exit: caller responsibility
    Grab return value
    Restore caller-saved registers.

Lec17 Parameter Passing

- Four ways
  - Pass by value
  - Pass by reference
  - Pass by value-result
  - Pass by name

For reference and value-result, the argument must be l-value.

- Strategy: Use a stack to simulate the process.

Lec 18 Runtime Access to Variables.

- MIPS

  opcode Operand1 Operand2.

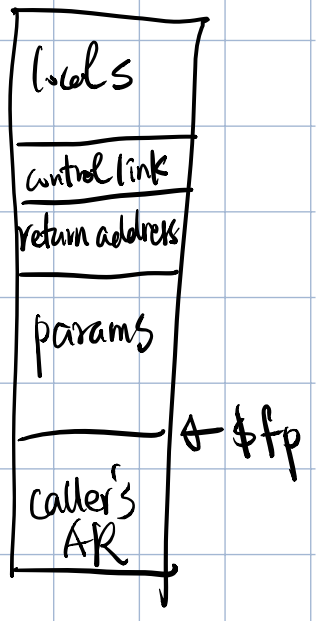
  lw register memAddr
  sw register memAddr.

- Variable types: Local, Non-local, global
- Relative access for locals: offset from $fp.

For each function

Assuming $fp points to the 1st parameter pushed by the caller. This's is a bit different from the lecture note, but consistent with the online note.

```
Set offset = 0
for each parameter
    add name to symbol table
    offset -= size of parameter
offset -= size of return address
offset -= size of control link
offset -= size of callee saved registers
for each local
   add name to symbol table
   offset -= size of variable
```



- Global variables

  Space allocated at compile time, never deallocated.
  Global variables are referred to by name, not by address.

- Non-local variables
  Inner function access variables declared in the outer function.
  Basically, cross-AR variable access.

  - Static non-local variable access.
    Add an "Access Link" to each AR, pointing to the local area of the outer function.
    Outer function isn't necessarily the caller!

  We can reduce # of access links to jump into "displays."
  A side table stores "variable-address" information.
  Faster but takes more space.

- Dynamic non-local variable access
  We don't know which variable we're referring to at compile-time. Two ways: deep access, shallow access.
  - Deep access
    If a variable isn't local, follow control link to the caller and check if it defines the variable. If not, follow the control link again.
    Note that we need to store a "name-address" mapping.
  - Shallow Access
    Keep a table with one entry for each variable declaration.
    At a function call, save caller's locals address into table;
    When the callee finishes, restore caller's locals.

Lec 19