This handout summarizes a simple TAC intermediate language. There are many choices as to the exact instructions to include in such a language, and you will probably want to modify and extend this variant when we translate IC programs into TAC.

### Instruction Forms

- **Arithmetic and Logic Instructions.**

  The basic instruction forms are:

  \[
  a = b \text{ OP } c \\
  a = \text{ OP } b
  \]

  where \( \text{OP} \) can be

  - an arithmetic operator: \( \text{ADD}, \text{SUB}, \text{DIV}, \text{MUL} \)
  - a logic operator: \( \text{AND}, \text{OR}, \text{XOR} \)
  - a comparison operator: \( \text{EQ}, \text{NEQ}, \text{LE}, \text{LEQ}, \text{GE}, \text{GEQ} \)
  - a unary operator: \( \text{MINUS}, \text{NEG} \)

- **Data Movement Instructions.**

  Copy: \( a = b \)

  Load/store: \( a = \ast b \) \( \ast a = b \)

  Array load/store: \( a = b[i] \) \( a[i] = b \)

  Field load/store: \( a = b.f \) \( a.f = b \)

- **Branch Instructions.**

  Label: \( \text{label } L \)

  Unconditional jump: \( \text{jump } L \)

  Conditional jump: \( \text{cjump } a \ L \) (jump to \( L \) if \( a \) is true)

- **Function Call Instructions.**

  Call with no result: \( \text{call } f(a_1, \ldots, a_n) \)

  Call with result: \( a = \text{call } f(a_1, \ldots, a_n) \)

  (Note: there is no explicit TAC representation for parameter passing, stack frame setup, etc.)