Three-Address Instructions

Assignments:
- \( x = y \oplus z \) (binary operator)
- \( x = \circledast y \) (unary)
- \( x = y \) (copy)

Memory Operations:
- \( x = \gamma y \); \( x = \gamma y \) and \( x = y \); for assignments via pointer variables.

Simulator c3esim

- Memory Model:
  - Addressable as a Unit not Bytes
  - Unlimited Symbolic Variables
  - Simple Instructions with Basic Elements
- Base Types:
  - Character, Integer, Double
- Three-Address Instructions:
  - Operands are Either Symbols (Variables) or (Typed) Constants
  - Control Flow Instructions
    - Unconditional/Conditional Jump or Goto
    - Call/Return using Address and Argument Stacks
  - Arithmetic and Logic Instructions
    - Addition, Subtraction, Multiplication, Division
    - Relational Operators: results integer: Zero is False, non-zero is True

Structure of an Assembly Program

Data section header!

Comments!

Text (Code) section header!
Structure of an Assembly Program

Data Declaration and Allocation Section
- variable name
- basic type (word, byte, skip)
- initial value
- skip directive reserves and initialization space

Text (Code) Section
- entry labels for procedures
- code starts executing at “main”

Conditional Control-Flow

- Predicate Evaluation
  - Constants or Variables Operands
  - No Side Effects
  - True ≠ non-zero
  - False ≠ zero

Example:
- “if y >= 0 goto L4”
- How does this code behave?

Example:
- “if y >= 0 goto L4”
- How does this code behave?

Call Sequence Example

Operands & Arguments Stack

- Stack frame reserves and initializes space
- Text directive reserves and initialization space

Conditional Control-Flow

- Predicate Evaluation
  - Constants or Variables Operands
  - No Side Effects
  - True ≠ non-zero
  - False ≠ zero

Example:
- “if y >= 0 goto L4”
- How does this code behave?

Call Sequence Example

Operands & Arguments Stack

- Stack frame reserves and initializes space
- Text directive reserves and initialization space
Call Sequence Example

```
main:
    a = 3

add:
    t = x + y
    return t
```

Predefined I/O Functions

- Six Basic I/O Functions
  - fread: character read
  - fwrite: character write
  - iread: integer read
  - iwrite: integer write
  - fread: float/double read
  - iwrite: float/double write

- Strings:
  - Mapped to a sequence of chars with null terminated character.
  - No predefined I/O functions
Handling Arrays & Structures/Records

- No Support for Arrays or Structures
  - Use Memory Operation Instructions
  - Compute & Manipulate Addresses
  - Use Load/Store for Read/Write

Strings:
- Mapped to a sequence of chars
  - No predefined I/O functions

Predefined I/O Functions

- Six Basic I/O Functions
  - read: character read
  - write: character write
  - in: integer read
  - out: integer write
  - float: float/double read
  - float: float/double write

Structures/Records

- No Support for Arrays or Structures
  - Use Memory Operation Instructions
  - Compute & Manipulate Addresses
  - Use Load/Store for Read/Write

Compiler Design

Memory

• Use Load/Store for Read/Write
• Use Memory Operation Instructions
• Compute & Manipulate Addresses
• No predefined I/O functions

PC
Handling Arrays & Structures/Records

- No Support for Arrays or Structures
  - Use Memory Operation Instructions
  - Compute & Manipulate Addresses
  - Use Load/Store for Read/Write

UNIX Simulator Implementation

- Uses do and par UNIX tools
  - Limited error checking (work in progress)
  - A lot of run-time checks - it is slow... (who cares?)