Rudimentary Compilation (Objectives)

- Given a simple language of expressions, the student will be able to write a simple compiler for those expressions in order to give semantics to the language.

1

Compilers and Syntax

- Compilers give meaning to syntax.
  - What does the following syntax mean?

```
x = 1 + 2 *3;
```

- To start, we will write a compiler for a simple calculator language.

$$
\begin{array}{rcl}
Calc & \rightarrow & Calc;E \\
 & | & E \\
E & \rightarrow & E+T \\
 & | & E-T \\
 & | & T
\end{array}
\qquad
\begin{array}{rcl}
T & \rightarrow & T*F \\
 & | & T/F \\
 & | & F \\
F & \rightarrow & \mathbf{num} \\
 & | & (E) \\
 & | & -F
\end{array}
$$

2

Lemon Specification of Calc

calc ::= calc SEMICOLON expr.
calc ::= expr.

expr ::= expr PLUS term.
expr ::= expr MINUS term.
expr ::= term.

term ::= term TIME factor.
term ::= term DIVIDE factor.
term ::= factor.

factor ::= NUM.
factor ::= LPAREN expr RPAREN.
factor ::= MINUS factor.

3

What is the meaning of a Calc Program?

- Does the Calc program below have meaning?

```
1+2 ; -7
```

  - If so, what is it?
  - If not, why and how can we give it meaning?

4

Meaning of **num**

- Let's consider of a subset of the Calc language

$$\begin{array}{rcl} Calc & \rightarrow & Calc; E \\ & | & E \\ E & \rightarrow & T \\ T & \rightarrow & F \\ F & \rightarrow & \textbf{num} \end{array}$$

How do we give it meaning in our compiler?

5

Regression – Review of MIPS Assembler

- We will express the meaning of a high-level language with MIPS assembler
- MIPS Architecture
  - 32-bit instructions
  - 32-bit word size
  - 32 general purpose registers
    - Reserved: $zero, $at, $k0, $k1, $gp, $fp, $sp, $ra
    - Preserved: $s0 - $s7
    - Not preserved: $v0, $v1, $t0 - $t9
    - Arguments: $a0 - $a3

MIPS Assembler Continued

- Load/store instructions
  - Load immediate
    li $v0, 5
  - Load word from memory (register indirect)
    lw $v0, 0($t1)
  - Store word to memory (register indirect)
    sw $v0, 0($t1)
  - Move a register
    move $t0, $t1
- Arithmetic instructions
  - Addition, subtraction, multiplication, division
    add $t0, $t1, $t2
    sub $t0, $t1, $t2
    mult $t0, $t1, $t2
    div $t0, $t1, $t2

MIPS Assembler Continued

- Logical instructions
  - Bit-wise and, or, xor, not
    - and $t0, $t1, $t2
    - not $t0, $t1
- Comparison
  - slt, sle, sgt, sne, seq
    - sne $t0, $t1, $t2

I/O: read integer

```
li $v0, 5
syscall
```

The read value is in $v0 after the call

9

I/O: write integer

- Print the integer 5

```
li $a0, 5
li $v0, 1
syscall
```

10

I/O: print string

- Print “Hello”
  - Define the string in the data section
    .string0: .asciiz “Hello”

```
la $a0, .string0
li $v0, 4
syscall
```

11

Meaning of addition and subtraction

- How do we determine the meaning of the following syntax?

$$E \rightarrow E + T \mid E - T$$

12

Practice Problem

- Finish the compiler for *T* and *F*.

13

What is missing?

- Even in simple calculations, we can encounter repeated calculations
  - Solution → variables
- Syntax for variable declarations

$$
\begin{array}{lcl}
P & \rightarrow & Vars\ Calc \\
Vars & \rightarrow & \textbf{var}\ IdList; \\
IdList & \rightarrow & IdList, Id \\
 & | & Id
\end{array}
$$

- What is the meaning of a variable declaration?

14

Binding Time

- Binding – an association between a name and what it names (e.g, variable and memory location).
- Times when bindings occur
  - Language design time
    - types, keywords, etc.
  - Language implementation time
    - Left to the implementation
    - Precision of operations, evaluation order of parameters, etc.
  - Program writing time
    - Variable names, etc.
  - Compile time
    - Memory layout
  - Load time
    - Machine addresses
  - Run time
    - Values to variables, method invocation

15

Binding

- Static binding
  - Refers to binding that occurs before run time
    - Statically scoped variables
    - Functions, some methods
- Dynamic binding
  - Refers to bind that occurs at run time
    - Dynamically scoped variables
    - Some virtual methods
      - Smalltalk

16

Object Lifetime

- Where should a variable be stored?
  - It depends on its lifetime
- Object lifetime – the period of time between the creation and destruction of an object (an object is a piece of data)
- Storage locations
  - Static data area – objects whose lifetime is the entire execution of a program
    - Global variables
    - String constants
  - Stack – objects whose lifetime consist of a procedure call
    - Allocated on entry to and deallocated on exit from a procedure
    - E.g., local variables
  - Heap – objects whose lifetime vary depending on execution
    - Pointers
    - Requires garbage collection

17

Allocating Space

- Where should variables in the calc language be allocated?
- Once we allocate a variable to a memory location, we need to retain that information for variable references.
  - Where should that information be stored?
    - Symbol table (hash table is one organization)

18

Symbol Tables

- What items should be entered in a symbol table?
  - variable names
  - literal constants and strings
  - source text labels

19

Information in a Symbol Table

- character string for each name
- data type
- storage class (base address, static data area, heap, stack)
- offset in storage area

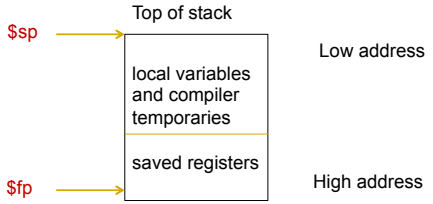
20

Symbol Table Organization

- How should the table be organized?
- Linear list
  - O(n) probes per lookup
  - easy to expand, no fixed size
  - one allocation per insertion
- Binary tree
  - O(log n) probes per lookup (balanced tree)
  - easy to expand, no fixed size
  - one allocation per item
- Hash table
  - O(1) probes per lookup (expected)
  - expansion costs vary with collision resolution scheme

21

MIPS Stack Frame



- $fp points to the base of stack
  - $fp must be set in assembler explicitly
- $sp points to location at the top of the stack

22

Adding Variable References

- Add the following syntax to a Calc program

$$Factor \rightarrow Id$$

- Update the compiler to handle variable declarations and references

23