

COMP 442/6421 Compiler Design

Instructor: Dr. Joey Paquet paquet@cse.concordia.ca
TA: Zachary Lapointe zachary.lapointe@mail.Concordia.ca

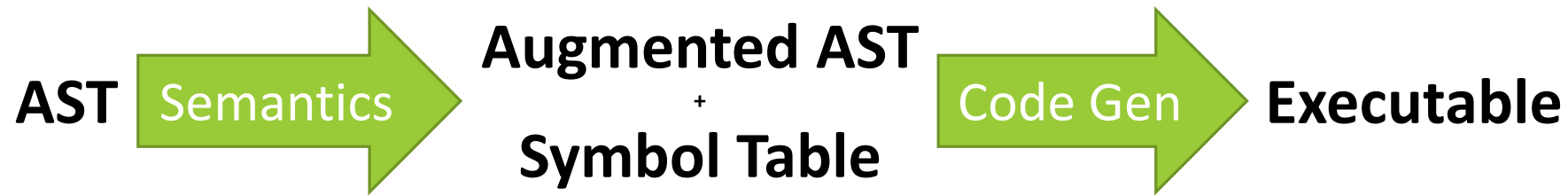
LAB 8 – CODE GENERATION II:
ADVANCED FEATURES AND MEMORY MANAGEMENT

Lab 8 Outline

- Bit operations
- Advanced code generation issues
 - Tags
 - Floating point numbers
- Final Thoughts

Augmented AST and Symbol Table

- AST++
 - A rich structure which represents the meaning of the program
 - It is the primary artifact used for code generation (A4)
- Symbol Table
 - Contains everything the programmer named
 - We will be adding new memory information to it for code generation
 - Temporary variables
 - Tags and/or stack memory offsets
 - Secondary artifact used for code generation (A4)



Bit-logic – Two's Complement

- The moon processor interprets integers using the *two's complement* format.
 - Most significant bit is sign bit
- Generally, you don't need to worry about this
 - Addition, subtraction, multiplication, divisions and comparison operations all work as you'd expect.
 - Just keep in mind if you end up looking at specific bits
- Two's complement conversion to decimal, for n bits, numbered 0 to n-1
 - $-b_{n-1}2^{n-1} + \sum_{i=1}^{n-2} b_i 2^i$
 - Sums of powers of 2, with the largest being negative

Binary	Decimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

Bit-logic – Bitwise Operations

- Many Moon processor logic operations work *bitwise*
 - They operate on pairs of bits
- This works as you'd expect for certain operations
 - Arithmetic: *add, sub, mul, div, mod*
 - Comparisons: *ceq, neq, clt, cle, cgt, cge*
- Logical operations **won't** work as you expect
 - *and, or, not*
 - They operate bit by bit
 - You'll need to find a different way to implement the logical operators

$$\begin{array}{r} (4)_{10} \\ \text{AND } (3)_{10} \\ \hline (0)_{10} \end{array}$$

$$\begin{array}{r} (0100)_2 \\ \text{AND } (0011)_2 \\ \hline (0000)_2 \end{array}$$

Bit-logic – Bit Masks

- A useful tool for helping with this are *bit masks*
 - *What is it?* An integer, which is picked specifically for its bit pattern
 - *How does it work?* By doing a bitwise AND between an integer and a *mask*
 - *What does it do?* It hides, or *masks*, certain bits in a word, keeping only the ones you're interested in.
 - *Why?*
 - Particularly useful for implementing floating point operations and interpretations
 - Separating the sign, exponent and significand
 - Remember the moon processor only uses immediate decimal integers, not binary integers
 - As well, remember immediates are limited to 2 bytes
- Java allows writing integer literals using binary notation
 - Any integer literal, preceded with **0b**, will be interpreted as binary
 - `int number = 0b0110; //stores 6 in number`
 - Unrelated: you can put underscores withing literals to improve readability
 - Example
 - [Reference](#)

Tags for control flow

- Tags in moon code are necessary for jumping between functions and conditional structures
 - They are straightforward to use, but make sure generated tags are **always unique**
 - Prefixes can help with this
 - Be careful of tricky edge cases:
 - Function overloading
 - Function overriding and inheritance
 - Similar free functions and member functions
 - If using tag for memory, uniqueness is much harder
 - Example
 - `class_function_functionName_param1Type_param2Type`
 - `if_22, then_22, else_22`

Floating Point Numbers

- Floating point numbers are quite difficult to implement
 - The moon processor is only defined for integers, floats have to be *implemented*
 - Floating point operations are much more complicated than integer operations
- The IEEE-754 specification on floats can help
- The basics
 - 4 bytes
 - **Sign**: most significant bit
 - **Exponent**: 8 next bits
 - **Significand**: remaining 23 bits
 - Bit masks and bit shifting can be used to isolate and manipulate these

Final Thoughts

What Comes Next? – Compiler Design

- Further learning in Compiler Design
 - Optimization
 - Advanced error handling
 - Different and novel language features
 - Runtimes and Interpreted languages
- Compilers in the wild
 - Many languages have *free* or open source compilers
 - [GCC](#) (C, C++, Fortan, Go, ...)
 - [Java](#)
 - [Clojure](#)
 - *and many more . . .*