CS61C — Precheck: CALL, Boolean Algebra

Spring 2025 — Discussion 7

1 Discussion Pre-Check

1.1 The compiler may output pseudoinstructions.

True. It is the job of the assembler to replace these pseudoinstructions.

1.2 The main job of the assembler is to perform optimizations on the assembly code.

False. That's the job of the compiler. The assembler is primarily responsible for replacing pseudoinstructions and resolving offsets.

1.3 The object files produced by the assembler are only moved, not edited, by the linker.

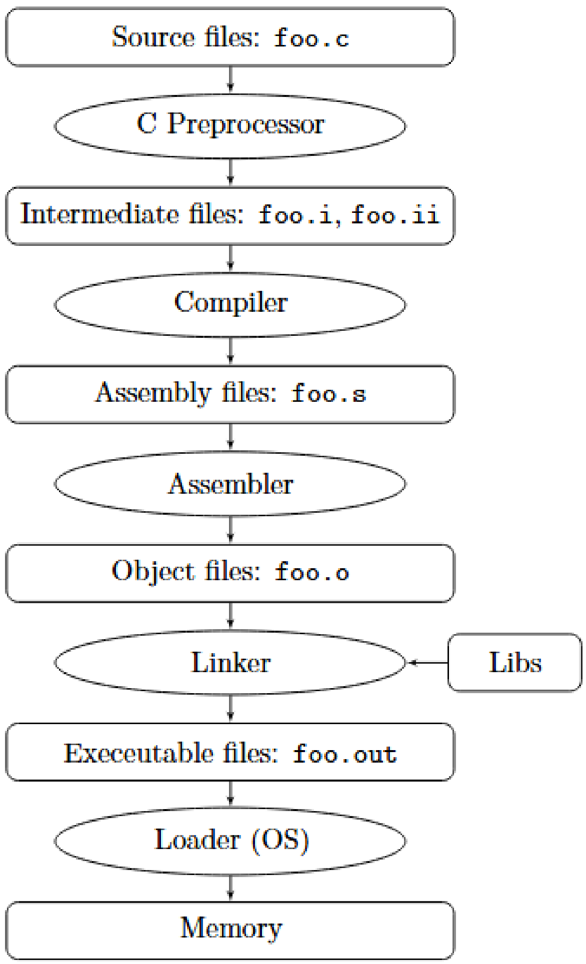
False. The linker needs to relocate all absolute address references.

1.4 The destination of all jump instructions is completely determined after linking.

False. Jumps relative to registers (i.e. from jalr instructions) are only known at run-time. Otherwise, you would not be able to call a function from different call sites.

2 CALL

The following is a diagram of the CALL stack detailing how C programs are built and executed by machines:



To translate a C program to an executable:

1. (C Preprocessor): translates all defined macros before passing to the compiler.
2. **Compiler**: Translates high-level language code (e.g. `foo.c`) to assembly
   - **Output**: Assembly language code (RISC-V) which may contain pseudoinstructions!
3. **Assembler**: Replaces pseudoinstructions and creates an *object file* with machine language, symbol table, relocation table, and debugging information.
   - **Output**: Object file (`foo.o`)
4. **Linker**: Combines multiple object files / libraries to create an executable.
   - **Output**: Executable machine code (e.g. `foo.out`).
5. **Loader**: Creates the environment to run machine code and begins execution.

2.1 How many passes through the code does the Assembler have to make? Why?

Two: The first finds all the label addresses, and the second resolves forward references while using these label addresses.

2.2 Which step in CALL resolves relative addressing? Absolute addressing?

The assembler usually handles relative addressing. The linker handles absolute addressing, resolving the references to memory locations outside.

2.3 Describe the six main parts of the object files outputted by the Assembler (Header, Text, Data, Relocation Table, Symbol Table, Debugging Information).

- Header: Sizes and positions of the other parts
- Text: The machine code
- Data: Binary representation of any data in the source file
- Relocation Table: Identifies lines of code that need to be "handled" by the Linker (jumps to external labels (e.g. lib files), references to static data)
- Symbol Table: List of file labels and data that can be referenced across files
- Debugging Information: Additional information for debuggers

3 Boolean Logic

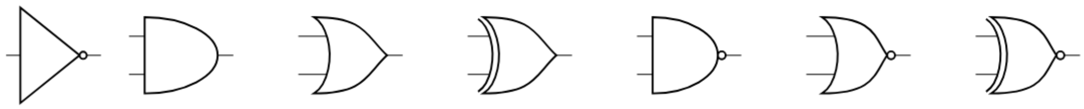
In digital electronics, it is often important to get certain outputs based on your inputs, as laid out by a truth table. Truth tables map directly to Boolean expressions, and Boolean expressions map directly to logic gates. However, in order to minimize the number of logic gates needed to implement a circuit, it is often useful to simplify long Boolean expressions.

We can simplify expressions using the nine key laws of Boolean algebra:

Name	AND Form	OR form
Commutative	$x \cdot y = y \cdot x$	$x + y = y + x$
Associative	$(xy)z = x(yz)$	$(x + y) + z = x + (y + z)$
Identity	$x \cdot 1 = x$	$x + 0 = x$
Null	$x \cdot 0 = 0$	$x + 1 = 1$
Absorption	$x \cdot (x + y) = x$	$x + x \cdot y = x$
Distributive	$(x + y) \cdot (x + z) = x + yz$	$x \cdot (y + z) = xy + xz$
Idempotent	$x \cdot x = x$	$x + x = x$
Inverse	$x \cdot \overline{x} = 0$	$x + \overline{x} = 1$
De Morgan's	$\overline{x \cdot y} = \overline{x} + \overline{y}$	$\overline{x + y} = \overline{x} \cdot \overline{y}$

Additionally, we have many boolean functions which take boolean signals (0 or 1) as input and output a boolean result (0 or 1). When designing digital systems, boolean functions are represented as **logic gates**.

3.1 Label each of the following logic gates with their respective boolean function, and draw a truth table representing their outputs:



NOT, AND, OR, XOR, NAND, NOR, XNOR

Here are the outputs for each boolean function combined into a single truth table. All possible combinations of the inputs x and y are shown the left, and the output of the the boolean function based on the current inputs is shown on the right.

Input(s)		**NOT**	**AND**	**OR**	**XOR**	**NAND**	**NOR**	**XNOR**
x	y	$\overline{x}$	$x \cdot y$	$x + y$	$x \oplus y$	$\overline{x \cdot y}$	$\overline{x + y}$	$\overline{x \oplus y}$
0	0	1	0	0	0	1	1	1
0	1	1	0	1	1	1	0	0
1	0	0	0	1	1	1	0	0
1	1	0	1	1	0	0	0	1