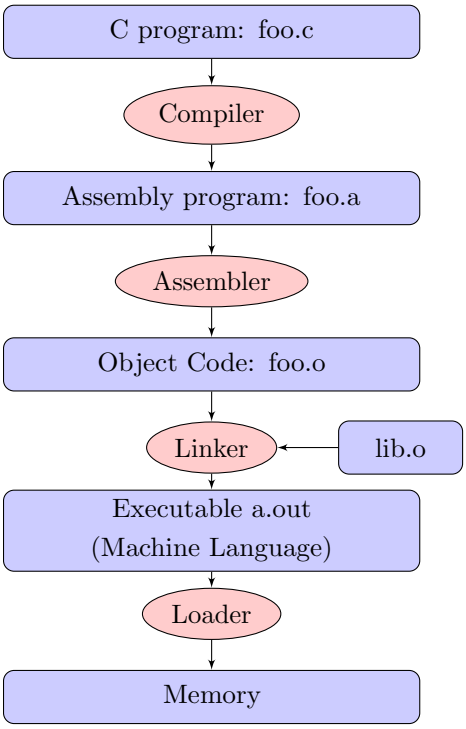
CS 61C Spring 2019

CALL, Logic

Discussion 9: February 20, 2019

1 CALL

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



1.1 What is the Stored Program concept and what does it enable us to do?

It is the idea that instructions are just the same as data, and we can treat them as such. This enables us to write programs that can manipulate other programs!

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

Two, one to find all the label addresses and another to convert all instructions while resolving any forward references using the collected label addresses.

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

- Header: Size and position of 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 Linker (jumps to external labels (e.g. lib files), reference to static data)
- Symbol Table: List of file labels and data that can be referenced across files
- Debugging Information: Additional information for debuggers

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

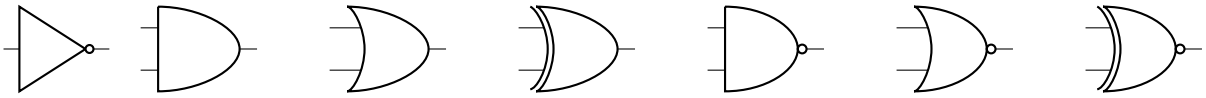
Assembler, linker

1.5 What does RISC stand for? How is this related to pseudoinstructions?

Reduced Instruction Set Computing. Minimal set of instructions leads to many lines of code. Pseudoinstructions are more complex instructions intended to make assembly programming easier for the coder. These are converted to basic instructions by the assembler.

2 Logic Gates

2.1 Label the following logic gates:



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

2.2 Convert the following to boolean expressions on input signals A and B:

(a) NAND

$\bar{A}\bar{B} + \bar{A}B + A\bar{B}$

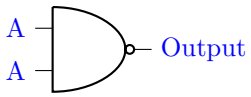
(b) XOR

$\bar{A}B + A\bar{B}$

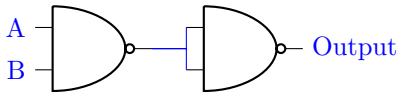
(c) XNOR

$\bar{A}\bar{B} + AB$

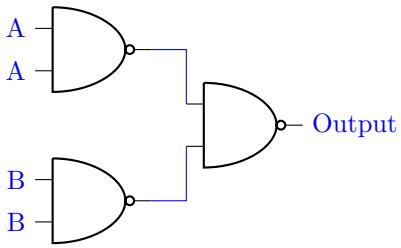
2.3 Create a NOT gate using only NAND gates.



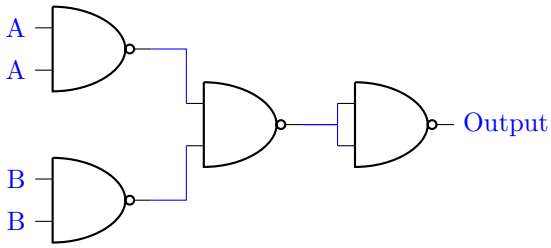
2.4 Create an AND gate using only NAND gates. (Hint: use 2.3!)



2.5 Create an OR gate using only NAND gates.



2.6 Create a NOR gate using only NAND gates. (Hint: use 2.3 and 2.5!)



2.7 How many different two-input logic gates can there be? How many n-input gates?

A truth table with n inputs has 2^n rows. Each logic gate has a 0 or a 1 at each of these rows. Imagining a function as a 2^n-bit number, we count 2^{2^n} total functions, or 16 in the case of $n = 2$.

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	$\mathrm{AB} = \mathrm{BA}$	$\mathrm{A} + \mathrm{B} = \mathrm{B} + \mathrm{A}$
Associative	$\mathrm{AB(C)} = \mathrm{A(BC)}$	$\mathrm{A} + (\mathrm{B} + \mathrm{C}) = (\mathrm{A} + \mathrm{B}) + \mathrm{C}$
Identity	$1\mathrm{A} = \mathrm{A}$	$0 + \mathrm{A} = \mathrm{A}$
Null	$0\mathrm{A} = 0$	$1 + \mathrm{A} = 1$
Absorption	$\mathrm{A}(\mathrm{A} + \mathrm{B}) = \mathrm{A}$	$\mathrm{A} + \mathrm{AB} = \mathrm{A}$
Distributive	$(\mathrm{A} + \mathrm{B})(\mathrm{A} + \mathrm{C}) = \mathrm{A} + \mathrm{BC}$	$\mathrm{A(B} + \mathrm{C}) = \mathrm{AB} + \mathrm{AC}$
Idempotent	$\mathrm{A(A)} = \mathrm{A}$	$\mathrm{A} + \mathrm{A} = \mathrm{A}$
Inverse	$\mathrm{A}(\overline{\mathrm{A}}) = 0$	$\mathrm{A} + \overline{\mathrm{A}} = 1$
Demorgan's	$\overline{\mathrm{AB}} = \overline{\mathrm{A}} + \overline{\mathrm{B}}$	$\overline{\mathrm{A} + \mathrm{B}} = \overline{\mathrm{A}}(\overline{\mathrm{B}})$

3.1 Simplify the following Boolean expressions:

(a) $(A + B)(A + \overline{B})C$

$$\begin{aligned}(A + B)(A + \bar{B})C &= (A + B\bar{B})C \\ &= AC\end{aligned}$$

(b) $\bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + AB\bar{C} + A\bar{B}\bar{C} + ABC + A\bar{B}C$

$$\begin{aligned}\bar{A}\bar{C}(\bar{B} + B) + A\bar{C}(B + \bar{B}) + AC(B + \bar{B}) &= \bar{A}\bar{C} + A\bar{C} + AC \\ &= \bar{A}\bar{C} + A\bar{C} + A\bar{C} + AC \\ &= (\bar{A} + A)\bar{C} + A(\bar{C} + C) \\ &= A + \bar{C}\end{aligned}$$

(c) $\overline{A(\bar{B}\bar{C} + BC)}$

$$\begin{aligned}\overline{A(\bar{B}\bar{C} + BC)} &= \bar{A} + \overline{\bar{B}\bar{C} + BC} \\ &= \bar{A} + \overline{\bar{B}\bar{C}}\,\overline{BC} \\ &= \bar{A} + (B + C)(\bar{B} + \bar{C}) \\ &= \bar{A} + B\bar{C} + \bar{B}C\end{aligned}$$

(d) $\overline{A}(A + B) + (B + AA)(A + \overline{B})$

$$
\begin{aligned}
\overline{A}(A+B)+(B+AA)(A+\overline{B} &= (\overline{A}A+\overline{A}B)+(B+AA)(A+\overline{B}) \\
&= \overline{A}B+(B+AA)(A+\overline{B}) \\
&= \overline{A}B+(B+A)(A+\overline{B}) \\
&= \overline{A}B+(BA+AA+B\overline{B}+A\overline{B}) \\
&= \overline{A}B+(BA+A+A\overline{B}) \\
&= \overline{A}B+A \\
&= A+B
\end{aligned}
$$