2. CALL

Interpretation vs Translation

Certain languages such as Python, Scheme, and PHP are interpreted on a *runtime* (usually written in C) and thus are running on other code. These are usually easier to debug and faster to code.

Translated code takes the source from a higher level language and creates an equivalent in a another language, usually a lower one. *Compilation* is an example of *translation*. This is run directly on the hardware. This is faster to execute, but takes up more space, loses portability, and is harder to debug.

Typical C Translation Steps

Compiler: Translates *source language* (e.g. C) into *assembly* (e.g. MIPS MAL) [.c → .s]
Assembler: Converts *assembly* into *machine code*; creates header sections, fills in addresses [.s → .o]
Linker: Fills in *symbol table*; converts *relative* → *absolute* address; loads libs [.o → a.out or prog.exe]
Loader: Inserts program into memory; takes care of dynamic libraries [runs program]

Dynamically vs Statically Linking Libraries

Dynamic linking is the process of not bundling libraries into the executable. Instead, the symbol table is filled out upon execution. Every call to a relative address involves a slight delay as the program calculates the actual address at runtime based on the library address.

The process of *dynamically linking libraries* saves disk space (sometimes tremendously). It also allows for more efficient memory usage if many programs are using the same library since only one copy then needs to be loaded. However, libraries can change their *APIs* with new versions, so there can be software incompatibility or DLL hell if the required dynamic library is not found.

A compromise is reached in a technique called *prelinking*, which checks the current version of libraries for their size and inserts the actual addresses into the symbol table assuming the library will be loaded. This allows dynamic loading to be space-efficient and quick at the same time. However, updates of libraries will require a new "prelinking" step for all programs that depend on that library.

3. Synchronous Digital Systems

Goal: *ISA* is a contract that specifies a set of software instructions that the processor must be able to execute. The software agrees to only call the instructions agreed upon.

Synchronous: Circuit elements bound by a clock, which is a measure of how often registers sample for new data.

Digital: Discrete 1/0 storage elements as well as waveforms with only two possible states (1/0).

System: Abstractions of details in discrete components and then dealing with interactions between these blocks.

4. State Machines

There are two main types of circuits: *combinational logic* and *state*

CL circuits perform operations on inputs in (almost) continuous time. The output is constantly updated (after a delay) whenever one of the inputs change. CL circuits have no memory.

State stores information using *flip flops*. The output is always the current contents of the *flip flop*. Input is *sampled* on the rising edge of the *clock* signal. A finite state machine is a limited set of rules that takes a given input and state and outputs an output and a new state. Thus, the # of states is a finite number. The state stores where the circuit currently is. Transitions are usually shown on a state machine transition diagram.

5. Solved Problems

Background: Java is a special case in programming languages; it differentiates between the **source** code and the **bytecode**. Both the bytecode and the source code are portable among many different computers, even those with different ISAs. The same source code produces the same bytecode regardless of the computer.

1. Describe the following relationships:
Java source → Java bytecode is an example of the process of (interpretation/**translation**/none)
Java bytecode is (**interpreted**/translated/none) on the (hardware/**runtime**/none)

2. Are Java bytecode instructions part of a "Java" ISA? **Yes. The JVM is an example of a software implementation of a processor with a distinct ISA.**

3. Now we tell you that Java performs something special called dynamic compilation, in which certain parts of the bytecode are translated into native machine code for faster performance, taking advantage of special instructions on the CPU when available. Is this a violation of the hardware's ISA contract?
No. Optimizations are done only if the instruction is present and detected.

4. Consider a parking meter using a state machine. The meter must be able to keep track of the total value of the coins that have been inserted, up to a dollar, at which point the machine resets to 0 and outputs a ticket. It should be able to accept 5c, 10c, 25c, and 50c coins. Any coin insertion that takes the machine over a dollar should be rejected and the state should be unchanged.

How many different states are there?
0, 5, 10, 15, 20, 25, 30, ... 95 = 20 states

How many total transitions are there?
4 possible coins can be inserted at each state, 4 x 20 = 80 transitions

How many transitions result in a new state?
Up to 50, all coins result in a new state. Between 55 and 75, 3 possible transitions. Between 80 and 90, two possible transitions. Between 90 and 95, 1 possible transition. Total = 4 x 11 + 3 x 5 + 2 x 3 + 1 x 2 = 44 + 15 + 6 + 2 = 67 transitions