Review Notes on Programming Languages

1 Abstract Machine

1.1 Definition

Consider a programming language L. An abstract machine, denoted by M_L, is a set of data structures and algorithms that can perform storage and execution of programs in L. A generic abstract machine is made of a *store* (memory for data/programs) and an *interpreter* (executes the instructions in programs). A program written in L, P_L can be considered a partial function that maps input data D_input to output data D_output:

$$P_L : D \longrightarrow D$$

1.2 Implementing a Language: Interpreted vs Compiled

Implementation of an *abstract machine* can be done (in increasing order of abstraction) in direct hardware (very fast execution but difficult to modify), firmware emulation (use low level microcode in a special separate read only memory, and this results in 2 level abstract machine: the assembly machine on top of the micro-programmed machine, and the assembly language interpreter on top of the micro-instructions), and finally software simulation.

Let's assume the abstract machine exists on a host machine M_H (we don't care how it's implemented) with it's machine language LM_H. Implementing language L on the host machine can be done in one of 2 ways:

1. Interpreted implementation: implicit translation from L to LM_H via simulation of M_L's constructs by LM_H. This is called a decoding rather than translation because code is executed, NOT output. An interpreter is a partial function that maps data to data.
2. Compiled implementation: explicit translation from L to LM_H

1.3 Interpreter

Definition: Assume P_L is the program written in the language we want L, D is the input data, L_h is the language of the host abstract machine M_H. An interpreter I^L_{Lh} implements the partial function:

$$Program + y^n = z^n$$

Google a diagram of a generic interpreter. The execution cycle starts with fetching the next instruction to be performed, decoding it, fetching the operands, executing a primitive instruction,

and finally either HALTING, or storing the result and returning to start to process the next instruction. The 4 functions of an interpreter are:

1) Operations that *process primitive data types* (ints, floats etc... though these are not always considered primitive depending on the language)

2) Operations and data structures that control the *sequence of execution*

3) Operations and data structures that control *data transfer*

4) Operations and data structures for *memory management*

1.4 Hardware Instantiation of an Interpreter

How are the 4 functions of a generic interpreter implemented by a physical computer?

1. Operations on primitive data types such as arithmetic and logical ops are performed by the ALU (arithmetic, boolean ops, shifts, tests).
2. Instruction sequence control is done by the program counter (PC).
3. Data transfer is done by CPU registers that inerface with main memory, such as the memory address register (MAR) and memory data register (MDR)
4. Memory processing in modern machines is done using multiple levels of memory (register-cache-main memory-secondary storage) along with algorithms designed for multi-programmed CPU's

1.5 Machine Language

Given abstract machine M_L, the language L understood by the the abstract machine's interpreter is called the *machine language* of M_L

- A physical computer, is a hardware instantiation of an abstract machine since it has memory and is made of logical circuits and electronic components.
- The machine's language has relatively simple instructions. A common instruction with 2 operands has the format *Opcode Opoerand1 Operand2*