Avery Karlin

Fall 2015

Contents

Primary Textbook:
Teacher:

1 Chapter 1 - Introduction

1. Computer systems are viewed from a bottom up approach of how transistors run basic programs and a top down approach of how complicated programs are converted into simple logic commands
   (a) This is used to understand commands a computer does well and does badly to program more efficiently, and to understand how changes in technology change the speed of computing
   (b) C is used due to being high level enough to write large programs, but low level enough to allow direct modification of bits
2. The concept of abstraction is central to computer science, focusing on a higher level, rather than the component ideas to save time and mental effort, assuming the details work
   (a) On the other hand, it must go in turn with deconstruction, breaking down an abstract idea into more concert sub-ideas, the opposite of abstraction, in case there is a problem
3. The concept of viewing hardware and software as joint components of a single system, which must both be taken into account to design either, is another central idea, to design the most effective components
4. Processors/CPUs are the primary unit of a computer, used to direct the processing of information and perform the calculations required to process it, though there are other components to make use easier
   (a) Originally, they were made of large boards covered in integrated circuit packages, but now are just a single silicon microprocessor chip with millions of transistors
   (b) Memory is another major component, made up of a series of slots, each with an address and able to hold a single value, connected to CPU
   (c) Programs are sets of instructions executed one at a time, stored in memory, while the program counter contains the current/next instruction in the program, often just incrementing after each
5. The only limitations of computers are time and amount of memory, but otherwise all computers can do the same tasks, though not at the same pace
   (a) This is due to computers being universal computing devices, as digital machines which could be increased in precision unlike analog/physical machines, but which were not made for individual tasks
   (b) Turing in 1937 proposed the Turing machine, which would be able to carry out all computations of some type, and later began to define what computation is, abstracting tasks by a black box model, showing the task, input, and output, with no specification about how it is performed
      i. Turing's thesis states that a Turing machine can do all computations, such that improvements to it do not change the amount of computations, making a universal Turing machine able to simulate all different Turing machines
   (c) Computers/universal Turing machines are able to do any computations, due to being programmable
6. Computer problems must be converted into voltages to influence the flow of electrons which the computer is made of, made of a series of methods to allow carrying out of complex tasks
   (a) The levels of transformation are the levels of choice to convert the problem into an electron flow for the computer, starting with the statement in a natural/human language, which has too much ambiguity to give directly to the computer

(b) The first transformation is to an algorithm, or a finite step by step procedure, with definiteness, or precisely stated, and effective compatibility, or able to be carried out by a computer,
   i. There are many possible algorithms, depending on the number of steps allowed concurrently by the computer, with many different speeds and lengths

(c) After, it is converted to a mechanical/programming language, specifically created to avoid ambiguity, often designed for specific purposes
   i. High level languages are those far from the computer itself, often machine independent, such that they don't rely on the computer specifics, such as C
   ii. Low level languages are specific to the computer, such that there is one for each computer generally, called assembly

(d) The third level is conversion into the computers instruction set architecture (ISA), or the specification of interface between programs and the hardware, generally x86 on Intel processors
   i. The ISA specifies the instructions and operations the computer can do, what inputs (operands) it requires, and the operand formats (data types) it is able to accept
   ii. It also contains mechanisms for the computer to find operands, called addressing modes, the number of unique memory locations, and the number of bits in each location
   iii. High level languages are converted to ISA format by a compiler, while low level languages are converted by an assembler

(e) After, the ISA is transformed into an implementation, based on the microprocessor-specific microarchitecture
   i. Unlike the ISA, which specifies what the computer can do, the microarchitecture specifies how the computer actually performs those tasks

(f) The microarchitecture is made out of logic circuits, determining the trade-off of cost and efficiency during manufacturing, each of which is further made out of a specific type of circuit materials, with its own specifications

2 Chapter 2 - Bits, Data Types, and Operations

1. The movement of electrons is controlled by devices reacting to the presence of voltage, rather than the amount for simplicity
   (a) These voltages are signified by binary digits/bits, 1 for voltage near the maximum, and 0 for voltage near zero (rounding due to device variation)
   (b) Bits are then combined to get a large number of distinct values
2. Data types are representations of values in which operations are encoded within the data types, mainly using 2's complement integers, ASCII codes, and floating points
   (a) Unsigned integers are used to identify locations and counts, represented by positional binary representation, ranging from 0 to $2^k - 1$ for k binary digits
   (b) Signed integers are used for arithmetic, with a leading 0 to signify positive, from 0 to $2^{k-1} - 1$ for k binary digits, using different systems for negative
      i. The signed magnitude system uses a leading 1 for a negative value, with a leading 1 on 0 signifying -0
      ii. The 1's complement system uses a leading 1 for negative values, signifying it is

equivalent to switching every other binary digit in the value (complement) to get the magnitude

iii. The 2's complement system is the standard data type, equivalent to the 1's complement - 1 for each negative value, found to be easiest to design hardware to do arithmetic on

3. Computers generally use an arithmetic and logic unit (ALU) for addition, converting two inputs to one output of the sum, performing it in the same column method with carrying as standard addition, without actually determining the values, leading to the 2's complement system
   (a) Thus, it is necessary for the sum of a number and its inverse to equal 0, ignoring all extra digits, such that only the correct number of digits is kept, and it is necessary for the sequence to be equal to one added, such that it works for results in the range, providing the advantage of the complements
   (b) The advantage of 2's complement over 1's complement is making full use of the maximum amount of digits, rather than having two versions of 0
   (c) The precision of the value is the number of numerical bits, while the range is the maximum magnitude of the bit sequence
4. Since addition is the main arithmetic bit operation, subtraction of bits is simply done by adding the additive inverse of the second value
   (a) Left shifts are equivalent to shifting all values to the left, adding a zero as the rightmost, dropping the overflow, equal to multiplying by 2 assuming no overflow
   (b) Arithmetic right shifts are shifting to right, adding the sign bit as the leftmost, while logical right shifts are shifting to the right, adding a 0 as the leftmost, equal to dividing by 2 for arithmetic shifts, and dividing unsigned by 2 for logical
      i. For odd values shifted right, the result is rounded based on the value in the 2nd rightmost spot
   (c) Sign extension/SEXT is used to shrink the amounts of bits used for a smaller number, removing all but one leading zero for positive, all but one leading one for negative
      i. Vice versa, the leading digits can be added, due to requiring the same number of bits used for adding or subtracting
   (d) Overflow is noted by the loss of the leading digit, such that the sum of two positive is negative, or vice versa, such it can be dealt with
      i. For unsigned addition, an outgoing carry denotes an overflow to be dealt with
5. Logical operations are based on viewing binary as true/1 and false/0, and are binary functions, requiring two logical inputs/bits
   (a) They are also able to be used on sequences of bits of the same length, testing corresponding bits from each
      i. Bit masks are sequences which are used to separate/modify specific bits from a sequence, using logical operations
   (b) Bit-wise AND returns 1 iff both inputs are 1, bit-wise OR (inclusive OR) returns 0 iff both inputs are 0, and bit-wise XOR (exclusive OR) returns 1 iff only one of the inputs is 1
   (c) The NOT/complement function takes a single input (unary function), inverting/switching each of the bits in the sequence
6. Bit vectors are binary sequences used to track if units are available/1 or busy/0, numbering units from right to left, starting with 0

7. Floating points are used to describe numbers of a high range, but low precision, in terms of scientific notation, such that for a 32 bit float, 1 signifies the sign, then 8 for the range/exponent, then 23 for the precision/fraction
   (a) Thus, the number is equal to (1 + fraction) * $2^{exponent-2^7+1}$, where the exponent is not allowed to be 0 or the maximum exponent
      i. If the exponent is 0, it is modified to 1 under the IEEE Standard for Floating Point Arithmetic
   (b) The fraction is found by converting the numerator and whole number to binary, then shifting to find the real exponent value, such that the fraction is the sequence except the leading 1
      i. This is done similarly to how it would be found for scientific notation, except that both parts are converted to binary before
8. ASCII (American Standard Code for Information Interchange) converts character codes from input and output into unique 8 bit codes universally
   (a) As a result, keys generally have multiple associated codes, due to different characters able to be produced by each
9. Hexadecimal notation easily is produced from binary, taking 4 bit blocks into binary digits for human ease

3 Chapter 3 - Digital Logic Structures

1. Microprocessors are generally made of metal-oxide semiconductor (MOS) transistors, divided into p and n types, each with three terminals, the gate, source, and drain
   (a) For some amount of voltage, generally 2.9 V, supplied to the gate of an n-type, the switch turns on, such that electricity can move from the source to the drain, forming a closed circuit
      i. This is generally only denoted by shorthand in drawings by the gate, which is written out, while the other terminals are just drawn as wire, denoted overall by a parallel line next to the wire with the gate coming from it
      ii. p-type transistors have a small circle between the gate and the line leading to the gate
   (b) p-type transistors work the opposite manner, acting as a wire only when there is virtually no voltage, while acting as an open circuit if 2.9 V is placed in the gate
   (c) As a result, circuits with both types are called complementary metal-oxide semiconductor (CMOS) circuits
2. Logic gate structures are transistor circuits for the purposes of logical functions, AND, OR, and NOT/Inverter gates as the fundamental gates
   (a) NOT gates are made up of two MOS transistors of opposite types, current flowing into the gates, the input leading to the gates of both, with a voltage going into the source of the p-type
      i. The drain of the p-type is connected to the output and the source of the n-type, with the drain of the n-type grounded
      ii. Thus, if there is voltage to the gates, voltage flows from the source to the output, while otherwise, voltage flows from the inputs to the ground, with none to the output, the latter necessary to create a grounded circuit if broken

iii. Inverters are drawn as a triangle with a small circle before the output wire, or can be drawn with just the small bubble/circle to show inversion as a shorthand

(b) The NOR (Not OR) gate has two p-type in series, each with a separate input, and with each input connected to one of two parallel n-type resistors, which are connected in series to the p-types, with the output between
   i. Each n-types are grounded, with voltage automatically flowing to the first p-type, such that it acts similarly to two NOT gates combined, such that the gate is grounded unless there is an output as well
   ii. OR gates are then created by adding an inverter to the output of the NOR gate
   iii. NOR gates are drawn as a crest-like shape, with an input to each horn, the output from the bottom point, with a small circle before the output wire, OR gates without the circle

(c) NAND (Not AND) gates have two parallel p-type transistors, each connected to an input and a voltage source, joining to the output, each connected to one of two in series n-types connected to the ground
   i. As a result, the circuit is grounded only if the output is zero, such that there is current through both inputs, while otherwise current is through the output
   ii. AND gates as a result are just NAND gates with an added inverter after
   iii. NAND gates are drawn as a closed semi-circle with a small circle before the output line, with two input lines, while AND gates are drawn without the circle

(d) DeMorgan's Law states that A OR B = NOT((NOT A) AND (NOT B)), showing the relationship between the NOT and OR gates

(e) The binary gates can be extended as a result to a larger number of inputs by extending the number of transistors that make up the gate

(f) Full adders take three one bit inputs, a_i, b_i, and the $carry_i$, such that the two results are s_i and $carry_{i+1}$, used to sum two bits in a column, adding the three inputs into 8 series of AND gates, each with a different combination of inverters before
   i. The results of the OR gates are then fed into two OR gates, depending on how they combine for the truth table, to produce the sum of the column
   ii. Thus, for a 2s complement or unsigned numbers, since it can be added in columns just as a decimal number, each full adder can be used for a different column, taking the carry of the previous, to produce the sum of binary digits

(g) The Programmable Logic Array (PLA) has an array of AND gates with a combination of inputs, such that for n inputs, there must be 2^n AND gates, each with varying negations
   i. These are then connected in some formation to an array of OR gates for each output, based on if the inputs produce each output
   ii. This as a result is able to be used to form any truth table in circuit form, consisting of only AND, OR, and NOT gates, such that those sets of gates are called logically complete