Computer Systems

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, added normally
   (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 (1 negative, 0 positive), then 8 for the range/exponent, then 23 for the precision/fraction
   (a) This is also called single, while doubles are similarly denoted, though the exponent is 11 bits, while the fraction is 52 bits
      i. Thus, doubles have an exponent from 0 to 2047 instead of 0 to 255, with a bias of 1023
   (b) 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 valued as if it is 1 under the IEEE 754 Standard for Floating Point Arithmetic
      ii. As a result, the 127 added to the exponent is called the bias, allowing for negative exponents as well
   (c) 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
      ii. The fraction can be thought to be continuing on the binary exponents, continuing as 2^{-1}, 2^{-2} and so forth
      iii. Due to converting non-binary denominator fractions into floating points, it can lead to various rounding errors, such that arithmetic with it would be close, but not exactly correct
         A. Thus, bounds of error are used to check the result, rather than equality functions
   (d) The sign bit is unrelated to the two's complement format
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 (nibbles) into binary digits for human ease, used identically to binary of that form, able to be added similarly
   (a) An additional bit at the end is removed similarly to twos complement, and overflow is noted and dealt with by the same means
10. Hexadecimal notation is noted by an x before the number, while decimal notation is denoted by # before the number, generally in assembly languages

3 Chapter 3 - Digital Logic Structures

3.1 Building Blocks

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, depending only on the current input
  (a) These can be drawn either in pull-up network (PUN), drawing from p-type transistors connected to the original source, switching from in-series to parallel or vice versa when at the n-type
    i. In this case, AND is parallel, OR is in series, automatically adding a NOT without an inverter
    ii. It can also be down pull-down network (PDN), drawn from the grounding to the n-type, such that AND is in series, OR is parallel, automatically adding a NOT without an inverter, considered easier
  (b) 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 power/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
  (c) 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, XOR without a circle, but with an additional crest line mirroring the main one on the bottom
  (d) 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

(e) DeMorgan's Law states that NOT(A OR B) = (NOT A) AND (NOT B), showing the relationship between the NOT and OR gates
   i. It follows as a result that NOT(A AND B) = (NOT A) OR (NOT B) is also a valid form

(f) Boolean algebra is also often written in the form of $\cdot$ as AND, $+$ as OR, and $^-$ as NOT, with the order of operations as NOT, AND, then OR
   i. As a result, $X \cdot 0 = 0, X + 1 = 1, X \cdot 1 = X$, and $X + 0 = X$ as the main identities, with both associative properties of AND and OR, and distributive property of AND over OR and of OR over AND

(g) Karnaugh Maps convert a logic array to a statement, making one side of the square the possible combinations of half the variables, the other side the other half
   i. It is drawn such that any adjacent cells differ by one variable, including across borders, though not diagonally, such that the order is 00-01-11-10, instead of 00-01-10-11 which is sequential binary
   ii. Adjacent entries of some power of 2 as a result can be designated as an AND, signifying the change from each, adding each separate group together with an OR, even if containing values held by other groups

(h) 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, thought it is noted that more levels creates a delay of that magnitude

3.2 Combinational Circuits

1. Combinational logic circuits, or decision elements, are structures of logic gates, depending only on immediate input data provided, rather than any stored data, contrasting data storage structures

   (a) Decoders are a circuit which takes two inputs, and returns four bits based on which combination of the inputs is 1, called the opcode, used often for turning on a specific hardware unit
      i. It thus decodes two ordered inputs, returning 1000 if 00 is the input, 0100 if 01, 0010 if 10, and 0001 if 11
      ii. This is done by placing four AND gates in parallel, each with a varying number and sequence of inverters directly before them

   (b) Mux, or multiplexers, select one of two inputs to use as output based on the select signal
      i. This is done by feeding the select signal and its inversion into two AND gates, each with one input going to it, combining the results in an OR gate to return the chosen value
      ii. This is extended to more inputs by increasing the number of possible select signals to the AND gate
      iii. This is drawn by a trapezoid with the select signal going to a leg, the inputs to the longer side, the output from the shorter, or as a rectangle
         A. Multiple select signals can be denoted by a slash on the wire with the number of signals written above the slash

   (c) 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
iii. These are drawn as either a square with an indication mark in the middle for a single unit with the carry going in from the right, out from the left, inputs from the top, output from the bottom
    A. For a complete full adder, it is drawn as a trapezoid with the inputs to the larger end, the inward carry to the top leg, and the output from the small end
iv. Subtraction is thus done by a full added by negating one of the input bits, then adding a carry of one to the sum based on the rules of 2's complement
v. Half adders are defined as an adder without an initial carry input, though still with a carry output

(d) 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
iii. While PLAs are able to implement any circuit, they are often not the most efficient method of creating the circuit
iv. Multiple bit inputs and outputs are denoted by a slash with the number of bits written next to the slash

(e) Arithmetic Logic Unit are thus made by a full adder, often combined with a decoder to separate between signals for AND, OR, ADD, or SUB, with a Mux to use the same adder for ADD and SUB

3.3 Memory

1. Basic storage elements are logical structures capable of storing some amount of data, contrasting combinational logic circuits

(a) R-S latches can store a single bit, made of two NAND gates, the output of each connected to the input of the other, with two external inputs (S and R)

i. In the quiescent state, S and R both begin as 1, such that whichever default input reaches first makes that output 1, at which point the other NAND gate returns 0, preserving the outputs
ii. If the S gate returns 1, then the overall data value is 1, otherwise 0, such that S or R can be set to 0 quickly to make that gate return 1, setting the variable as one of the values
iii. The term clearing the variable is used to denote setting it back to 0, such that clearing S or R makes that gate have a 1 output
iv. Both inputs are not able to be set to 0, in which case the result will depend on the

electrical properties of the transistors, rather than logic

v. As a result, the S gate signal is taken separately to get the overall output of the circuit

vi. Thus, it can either be drawn with both NAND gates with crossing outputs in the same direction, or with a cycle of NAND gates, taking the output of the S gate either way

(b) Gated D latches allow it to be controlled when a latch is set or cleared, such that it is two NAND gates, each gaining input from D, the gate in series with the R gate inverted, such that it determines which is set

i. Each of the D gates also have a write-enable signal, which determines if the byte is set in the first place, such that if it is activated briefly, one of the D gates stops signaling, setting the R-S latch based on D

(c) Registers store a series of bits as a unit, made up of a series of gated D latches, designated as an array of bits, starting from [0] as the rightmost

i. The same write-enable signal is given to each of the D latches, with a different D value given to each based on the value being stored

ii. Subunits of the series of bits are designated $[l : r]$ where l is the value of the leftmost, r is the value of the rightmost, called a field of the register

2. Memory is made up of a large number of locations, each with a unique identifier, or address, each storing some amount of data, the amount of bytes/bits (1 byte = 8 bits), called the addressability of the memory

(a) The total number of addresses is called the memory's address space, as some power of two based on the numbers of bits encoded as each memory address, such that the range is $[0, 2^n - 1]$

(b) Most memory systems are byte addressable, such that they store a single byte per memory address, corresponding to one ASCII character, but computers are often 64-bit addressable to allow for double floating point numbers to be stored, used in large calculations

3. Memory is denoted as size m by n bits, where m is the address space, n is the addressability, made up of a decoder to determine which memory address row, called the word line, with the size of the addressability

(a) The decoder sends the word line value into the mux, to determine which D latch provides the value through the mux, sending it to each column of D-latch/mux combinations to give one bit of the word

(b) The write enabled is connected to an AND gate with the decoder, such that the decoder signal must be set to the correct word line, along with the write enabled to set the latches

(c) Each column of mux/D-latches is also connected to a D input value to set the byte if the write enabled is activated

3.4 Sequential Logic Circuit

1. Combinational logic circuits are those that have no ability to store data from the past, while storage circuits store data from the past

(a) On the other hand, sequential logic circuits do both, such that they base the decisions on both the previous stored output and current input data, used for finite state machines

(b) As a result, this is capable of monitoring sequences, such as a sequential rotation lock

2. Thus, sequential circuits rely on the state of the circuit, defined as the external data of the input and current setting fed in, combined with all prior operations in the sequence and the settings deemed relevant
   (a) Each step within the sequence is thus considered a state of the system, as a snapshot of the current characteristics of the system
   (b) Finite state machines are thus defined as a system with a finite number of states, external inputs, external outputs, as well as explicit specification of state transitions and what determines an external output value
      i. These are represented by a state diagram, drawing a circle for each state, with connections/arcs from the current state to each other subsequent/next state, each with the external input to provide that transition written at the base
      ii. The internal output values of the system are written near the state, designating the code provided for the state, with the external output written inside the state bubble
3. State transitions are often defined by a clock circuit, with a signal alternating between 0 and 1 cyclically in some amount of time, triggering a transition when on
   (a) Clocks are denoted as an input by a small triangle pointing into a circuit at the entry point, drawn with one edge as an edge of the circuit
4. Thus, the combinational logic circuit aspect of the circuit is created based on the combination necessary, the storage element acting as a master-slave flip-flop instead of a gated D latch, which would modify the value stored immediately, rather than waiting until the next clock cycle
   (a) Thus, the master-slave flip-flop is made up of two D-latches for each byte, the output of one as the input of the other, the clock connected as the write-enabled, but negated for one, such that it is only able to change the value of one at a time
   (b) As a result, it writes the second when the clock is activated writing the value from the first, such that it is inputted into the circuit, and when the clock is deactivated, writes the first with the new value of the circuit
      i. The process of changing the feed-out value when the clock activates is called positive/rising edge triggered, while the process of changing the stored when it deactivates is called negative/falling edge triggered
      ii. The term clocked is used to refer to rising edge triggered generally
   (c) Registers are generally used to refer to a set of flip-flops, rather than a set of latches, though may be used for either

4 Chapter 4 - Von Neumann Model

4.1 Hardware

1. Computers are made up of a program, or a set of instructions, each a well-defined work command of the smallest possible division, and a hardware device, such that a von Neumann model is simplest fundamental model for processing programs
   (a) It is made up of a memory, processing unit, input, output, and control unit (controlling the order of instructions processed), storing the program in the memory
   (b) In von Neumann/computer model diagrams, filled arrows are used to signify command/control signals, while unfilled arrows are used to signify data

2. The memory for a standard computer in modern day is 2^{28} by 8 bits, while the LC-3 has 2^{16} by 16 bits
   (a) The contents from memory are read by placing the location in the memory's address register (MAR), which then has the data stored there placed in the memory's data register (MDR), written by activating Write Enable signal, then writing the address and data value in each respectively
   (b) As a result, the MAR for an LC-3 has 16 bits, due to the address requiring that many to specify, while the MDR has 16 bits for one data piece as well
3. The processing unit is made up of at least one ALU, used for simple arithmetic and logic functions, with the length of the quantity evaluated called the word length, each value a word
   (a) In the LC-3, the word length is 16 bits, though most computers in modern day have either 32 bits or 64 bits (such as the SPARC-V9)
      i. Since the addressibility of the LC-3 is equal to a word, it can be called word-addressable
   (b) The ALU for an LC-3 is able to perform addition, bitwise NOT, and bitwise AND
   (c) Most processors also have a small amount of memory, called the TEMP, next to the ALU, to store data for calculations with multiple parts needed in the near future
      i. This is due to taking a long amount of time to access the main computer memory, relative to the time needed to do the calculation, generally with the TEMP having a certain amount of registers, each for one word
      ii. The TEMP is also called the register file, composed in the LC-4 of three MUX, a write enabled, an input, and creating two 16 bit outputs
      iii. In the LC-3, it has 8 registers (R0, R1, ..., R7), while the SPARC-V9 has 32 registers
4. Input and output are made up of peripherals able to sense and display information, generally at the simplest the monitor and keyboard for an LC-3
   (a) A simple keyboard for an LC-3 requires two registers, a KBDR for the ASCII code of the last key hit and KBSR for maintaining the status of keys hit, while a simple monitor requires two registers, DDR for ASCII code being displayed and DSR for status information
5. The control unit keeps track of the order of instructions in the program, using the information register (IR)/program memory to contain the current instruction being executed
   (a) It also contains the address of the next instruction, stored within the program counter (PC), and a finite state machine to direct all activity taking input from the IR and from a clock (CLK)
      i. The clock is made up of a piezoelectric crystal oscillator to produce oscillating voltage, with a RUN latch connected to it and an AND gate, determining if the computer runs
      ii. Older computers used the HALT instruction to turn off the RUN latch, while newer computers, such as the LC-3 use the operating system to do so
   (b) The finite state machine has a series of command outputs, such as, within the LC-3, the two bit ALUK, controlling the operation of the ALU, or the GateALU, determining if the ALU output is sent out of the computer by the processor bus during that cycle
      i. The LD.MAR is the write-enabled signal for the MAR register, allowing the PC to be written into the MAR to get the subsequent command
      ii. The GatePC determines if the PC is connected to the processor bus, such that the

data from it can be accessed at a particular point

iii. The PCMUX allows it to choose the select line for modifying the PC in a specific manner, such as incrementing, giving the new value

iv. The LD.PC is the write-enabled signal for the PC register, while the LD.IR is the write-enabled signal for the IR, allowing the values to be modified

v. The GateMDR determines if the MDR is connected to the processor bus, such that the value from it can be sent to other parts of the system

vi. The MARMUX determines where the address supplied to the MAR is from, either from a register or the PC, specified by the ADDR1MUX, which then determines what length immediate/zero to add to it by the ADDR2MUX

A. The decision of which is controlled by a zero extended (positive sign) trap vector sent by the computer to determine which method is done

B. The important of using the correct immediate length is noted by the risk of removing sections as overflow should it not be chosen correctly

vii. It is noted that while these are used for LC-3, they are specific to a particular ISA, such that they are not universally created components

4.2 Instruction Processing

1. Instructions are made up of an opcode (command) on the left, written as 4 bits [15 : 12], in a 16 bit word, with the remaining 12 bits as the operands (data executing on)
   (a) Functions such as ADD or NOT are called an operate instruction, acting to process the data
   (b) Functions such as LDR are called a data movement instruction, due to moving data from one location to another
2. The commands are processed in the six phase instruction cycle, taking up a some number of machine/clock cycles, controlled by the finite state machine
   (a) It begins with the FETCH phase, taking the next instruction from memory by the address in the PC, loading it into the IR, after which the PC is incremented to point toward the next instruction, taking multiple cycles to access memory, the remaining steps each taking one
   (b) After, the DECODE phase studies the opcode to direct the command, sending an output line from the DECODE box to the processor in one clock cycle made up of all control signals (distinct from control commands), then the EVALUATE ADDRESS phase, which calculates any memory addresses needed
   (c) After, the FETCH OPERANDS phase takes the operands from either the memory from the EVALUATE ADDRESS phase, or from temporary storage, and then are used in the EXECUTE phase, finally using the STORE RESULT phase to finish the cycle
   (d) It is noted that while there are six phases, most commands such as the ADD (EVALUATE ADDRESS) and LDR (EXECUTE) do not require specific phases each
3. Control instructions are those which change the sequence of instructions, changing the PC after it is incremented, during the EXECUTE phase

5 Chapter 5 - LC-3 and LC-4

1. The LC-4 is a variation on the LC-3, using different formatting and cycle implementation in certain circumstances, but with generally similar conceptuals
2. The ISA specifies all functions of the computer that can be written for the computer in machine language, meaning Instructions Set Architecture for conversion from logic or high-level languages
   (a) The ISA is defined by the set of opcodes, data types of the operands, and addressing modes for a particular machine
3. The opcodes present in a particular ISA depend on the purpose of the computer, with varying amounts based on the level of complexity
   (a) Conversely, most computers prefer fewer large commands, rather allowing more variation in the instructions sent to maximize efficiency
   (b) The LC-3 has 19 different opcodes, with 1101 unspecified, each for a particular command, while the LC-4 uses 36 different opcodes, each with only 4 bits for it, such that it has certain opcodes with the first four bits overlapping, using additional bits to differentiate
4. Data types are representations of information that opcodes are able to operate on, supporting a specific format of data, such that LC-3 only supports 2's complement integers
5. Addressing modes specify where the operand is located, found either in the instruction mode (literal/immediate operand), register mode, or within memory
   (a) Addressing modes for within memory are either PC-relative, indirect, or base + offset modes, used only for data movement operands generally, using the other two for all operand types
   (b) Immediate operands are limited by the length of bits allowed within the command
6. Condition codes are single-bit registers found in the LC-3 and LC-4, set for any TEMP register writing function in LC-3, for any CMP function or any TEMP register writing in LC-4
   (a) The three registers are N (negative), Z (zero), P (positive), setting automatically to allow instruction sequencing as a result of the returned value
7. General purpose registers (GPR) are the registers/memory locations used for TEMP storage within the processor, each generally word-addressable, for the register addressing mode
   (a) It is noted that the same register can be used for both the source and destination
   (b) Loading is the process of moving memory to a register from the memory, while the process of moving away is storing
   (c) The LC-3 has seven data movement instructions, LD, LDR, LDI, LEA, ST, STR, and STI, while the LC-4 only has LDR and STR commands, with the first three bits after the opcode being the temporary register being used for loading/storing (destination register/DR)
      i. The remaining bits $[8:0]$ are the address generation bits, encoding the instructions for the data movement location/addressing mode
      ii. LDR and STR are the base + offset mode commands, with the leftmost three as the register for the base location (source register/SR), the remaining six bits as the literal offset value
      iii. LD and ST are PC-relative mode commands, providing all 9 bits as displacement from the current PC, after it has been incremented already
      iv. LDI and STI are indirect address mode, providing all 9 bits as the displacement from the current PC, to a location which contains the address of the data in memory

v. LEA is immediate mode, only able to load into a register, adding the incremented PC counter to the immediate value, which is then loaded into the register, not needing memory access

(d) The original valuing within an algorithm of variables is called the initialization

8. Control instructions are used to change the sequence of instructions, allowing conditional branches, unconditional jumps, subroutine/function calls, TRAP, and return from interrupt
   (a) The BR command is used for conditionals, checking some combination of NZP registers, if found, adding the incremented PC by the immediate offset given by the remaining bits for the new PC
      i. The use of all three registers is called an unconditional branch
      ii. Loops can either be run by iteratively, decreasing some counter each time the body of the loop executes, by iteration, or by sentinel, testing for an occurrence of some sentinel variable to stop the loop
         A. Sentinels are often unexpected character to search for, signifying the end of the sequence of expected characters
   (b) The JMP command allows movement to an address as an unconditional BR, with increased immediate length, such that there is a wider range
      i. The JMPR command allows changing the PC to any address contained within a register, to allow not using the offset within a wider range
   (c) The TRAP command sets R7 to the incremented PC, moving the PC to a memory address in the operating system, called an OS service call, the rightmost 8 bits as the trapvector
      i. The trapvector is an unsigned immediate that identifies the service call that the operating system is supposed to use
      ii. If the trapvector is 0, 0x8000 is used in the LC-4, while in the LC-3, an input character from the keyboard is x23, output to the monitor is x21, and ending a program is x25
      iii. The output and input character must be stored in R0 for the LC-3, which is the register character automatically stored and displayed
9. Coding in LC-3/LC-4 are done by using the mnemonic forms with spaces between the separate register codes, rather than the binary encoding or the semantic explanation
10. The global bus of the data path allows any structure within the computer to transmit 16 bits of information to any other structure, modifying the connections within the bus to determine the source and output, transferring one at a time, though some computers have multiple buses
    (a) Each component of the LC-3 has a tri-state device, drawn as a triangle, to allow the computers control to allow one device to send data through the bus at once, designated as the Gate control
    (b) On the receiving side, the LD (load enabled) control determines where the signal is being received

6 Chapter 7 - Assembly Language

6.1 Assembly Programming

1. For programming-ease, machine language given by the ISA can also be denoted by assembly language, contained within each ISA, each command representing one ISA command
   (a) This is then often translated into a high-level programming language such as C, moving it closer toward human language, which are ISA-independent
      i. High-level languages provide less control over the individual commands, sacrificing it for readability
   (b) Each assembly command has a mnemonic name for opcodes, and allows addresses to be set equivalent to text symbolic addresses
2. The assembler program translates assembly language, also called assembling it, into machine code for the computer
3. Instruction lines begin with a label, then the symbolic opcode, symbolic or explicit (such as register codes) operands, and finally comments (written after a semi-colon)
   (a) The label acts as a symbolic address for the memory address at that label, either storing an instruction or data
      i. In LC-3, labels can be placed on assembler commands, setting that label as the current location within the assembler, while this cannot be done in LC-4
      ii. Labels are often used to designate the start of loops or functions, as well as the end of the program overall in LC-4
   (b) Immediate values are written non-symbolically, such that binary is prefaced by a b, hex by an x, and decimal by an #
   (c) Comments are purely to make the code more readable to other programmers, ignoring all after the semicolon by the assembler
      i. Additional spaces and tabs are also used similarly, making the program more generally readable, though new lines can only be used at the end of a line properly
      ii. Comments are generally used at the start of the program to describe inputs, outputs, limitations, and the purpose of the program
4. Pseudo-ops/assembler directives are instructions directly to the assembler, not representing ISA opcodes, but rather to clarify the code for the assembler, denoted by a dot at the start
   (a) In LC-3, the .END command functions to tell the assembler when to stop reading, while the .ORIG replaces the .ADDR command
   (b) .FILL in LC-3 increments the current location within memory, then fills it with the value given, before incrementing again, while it fills the current value then increments in LC-4

6.2 Assembly

1. The assembly process converts the assembly language into machine language by the assembler software, using the "assemble *filename*.asm *outfile*" command in LC-3 assembler
   (a) In LC-4, "as *filename output*" is used instead in the assembler, producing an object (.obj) file
   (b) The assembly and setup process in the LC-4 can be shortened by a script file, called by "script *filename*.txt", containing commands
      i. The reset command is used to reset all LC-4 registers to 0, while "set *register/PC*

value is used to set the values of the PC and register before the program is run

ii. The object file is loaded into the LC-4 by "ld *filename*"

iii. Breakpoints in the program are set by "break set *label/address*"

2. The assembler requires a two-pass process due to the labels not necessarily being created before they are used
   (a) The first pass creates a symbol table, identifying the binary addresses corresponding to each label, while the second pass translates to machine code
   (b) During each pass, the current address code being created is kept track of by a location counter, incremented after
      i. As a result, the incremented PC for each command is the LC + 1
3. As a computer program is executed, it is done in executable image, made up of several object files linked together
   (a) Object files are either user-written and assembled, compiled from a higher level language, or built in libraries
   (b) Each object file is made up of ISA binary commands as a result
   (c) After it is linked into an executable file, the instruction cycle is used to run the file