

ECE120: Introduction to Computer Engineering

Notes Set 1.6 Summary of Part 1 of the Course

This short summary provides a list of both terms that we expect you to know and and skills that we expect you to have after our first few weeks together. The first part of the course is shorter than the other three parts, so the amount of material is necessarily less. These notes supplement the Patt and Patel textbook, so you will also need to read and understand the relevant chapters (see the syllabus) in order to master this material completely.

According to educational theory, the difficulty of learning depends on the type of task involved. Remembering new terminology is relatively easy, while applying the ideas underlying design decisions shown by example to new problems posed as human tasks is relatively hard. In this short summary, we give you lists at several levels of difficulty of what we expect you to be able to do as a result of the last few weeks of studying (reading, listening, doing homework, discussing your understanding with your classmates, and so forth).

This time, we'll list the skills first and leave the easy stuff for the next page. We expect you to be able to exercise the following skills:

- Represent decimal numbers with unsigned, 2's complement, and IEEE floating-point representations, and be able to calculate the decimal value represented by a bit pattern in any of these representations.
- Be able to negate a number represented in the 2's complement representation.
- Perform simple arithmetic by hand on unsigned and 2's complement representations, and identify when overflow occurs.
- Be able to write a truth table for a Boolean expression.
- Be able to write a Boolean expression as a sum of minterms.
- Know how to declare and initialize C variables with one of the primitive data types.

At a more abstract level, we expect you to be able to:

- Understand the value of using a common mathematical basis, such as modular arithmetic, in defining multiple representations (such as unsigned and 2's complement).
- Write Boolean expressions for the overflow conditions on both unsigned and 2's complement addition.
- Be able to write single `if` statements and `for` loops in C in order to perform computation.
- Be able to use `scanf` and `printf` for basic input and output in C.

And, at the highest level, we expect that you will be able to reason about and analyze problems in the following ways:

- Understand the tradeoffs between integer and floating-point representations for numbers.
- Understand logical completeness and be able to prove or disprove logical completeness for sets of logic functions.
- Understand the properties necessary in a representation: no ambiguity in meaning for any bit pattern, and agreement in advance on the meanings of all bit patterns.
- Analyze a simple, single-function C program and be able to explain its purpose.

You should recognize all of these terms and be able to explain what they mean. Note that we are not saying that you should, for example, be able to write down the ASCII representation from memory. In that example, knowing that it is a 7-bit representation used for English text is sufficient. You can always look up the detailed definition in practice.

- universal computational devices / computing machines
  - undecidable
  - the halting problem
- information storage in computers
  - bits
  - representation
  - data type
  - unsigned representation
  - 2's complement representation
  - IEEE floating-point representation
  - ASCII representation
- operations on bits
  - 1's complement operation
  - carry (from addition)
  - overflow (on any operation)
  - Boolean logic and algebra
  - logic functions/gates
  - truth table
  - AND/conjunction
  - OR/disjunction
  - NOT/logical complement/ (logical) negation/inverter
  - XOR
  - logical completeness
  - minterm
- mathematical terms
  - modular arithmetic
  - implication
  - contrapositive
  - proof approaches: by construction, by contradiction, by induction
  - without loss of generality (w.l.o.g.)
- high-level language concepts
  - syntax
  - variables
    - declaration
    - primitive data types
    - symbolic name/identifier
    - initialization
  - expression
  - statement
- C operators
  - operands
  - arithmetic
  - bitwise
  - comparison/relational
  - assignment
  - address
  - arithmetic shift
  - logical shift
  - precedence
- functions in C
  - `main`
  - function call
  - arguments
  - `printf` and `scanf`
    - format string
    - escape character
  - `sizeof`
- transforming tasks into programs
  - flow chart
  - sequential construct
  - conditional construct
  - iterative construct/iteration/loop
  - loop body
- C statements
  - statement: null, simple, compound
  - `if` statement
  - `for` loop
  - `return` statement