<table>
<thead>
<tr>
<th>Subject Sequence</th>
<th>Marcovitz Textbook Reading Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Digital World</td>
<td>Howstuffworks.com</td>
</tr>
<tr>
<td>Number System Conventions</td>
<td>1.1-1.3</td>
</tr>
<tr>
<td>Boolean algebra</td>
<td>2.2-2.4, 2-7-2.8</td>
</tr>
<tr>
<td>Logic Gates and Circuits</td>
<td>2.6</td>
</tr>
<tr>
<td>minterms and K-maps</td>
<td>2.5, 3.1</td>
</tr>
<tr>
<td>Maxterms and Kmaps</td>
<td>3.2-3.4</td>
</tr>
<tr>
<td>Important types of CLCs</td>
<td>4.2-4.5</td>
</tr>
<tr>
<td>ROM, PLD &amp; RAM Structures</td>
<td>4.6</td>
</tr>
<tr>
<td>Sequential Logic Circuits</td>
<td>5.1-5.2</td>
</tr>
<tr>
<td>Flip-flops and Clocks</td>
<td>5.3</td>
</tr>
<tr>
<td>Sequential logic circuit analysis and design</td>
<td>6.1-6.4</td>
</tr>
<tr>
<td>Important Types of SLCs</td>
<td>7.1-7.3</td>
</tr>
</tbody>
</table>
THE DIGITAL WORLD

Major Application of Digital Logic: the design of microprocessor chips in computers & mobile devices.

iPod Architecture
From: electronics.howstuffworks.com/ipod3.htm
1. Microprocessor chip
2. Memory (SDRAM 256 MB)
3. Peripherals
   a. Click Wheel (capacitive sensing controller)
      electronics.howstuffworks.com/ipod4.htm
   b. Hard drive (30 GB)
   c. Display (320 x 240 pixel LCD)
      electronics.howstuffworks.com/lcd2.htm
4. iPod Touch differences
   a. Smaller electronics
   b. Touch display (480 x 320 pixel x 2)
      electronics.howstuffworks.com/ipod-touch2.htm

Digital Data Types
1. Numeric
   a. Integer
      i. Unsigned (count values)
      ii. Signed (add or subtract)
   b. Floating Point
      i. Radix (decimal) points
      ii. Sign, fraction & exponent
   c. BCD (binary-coded decimal)
2. Nonnumeric
   a. Characters
      i. ASCII: 1 B for each of $2^8 = 256$ English and control characters (Latin-1)
      From: howstuffworks.com/bytes2.htm
      ii. UNICODE: 2 B for each of $2^{16} = 64K$ International characters.
   b. Audio
      i. Analog waveforms
      From: howstuffworks.com/analog-digital2.htm
      ii. A/D Conversion to 16 bits (CD)
      From: howstuffworks.com/analog-digital3.htm
      iii. Compression.
      From: computer.howstuffworks.com/mp32.htm

Notes:

B = byte = 8 bits (b)
K = $2^{10} = 1024 \approx 10^3$
   $2^9 = 512$
   $2^8 = 256$
   … etc.
M = K x K = $2^{20} \approx 10^6$
G = K x M = $2^{30} \approx 10^9$

A motherboard is where all the circuit components are mounted. Major components are connected by bus ribbons.

One large square chip is numbered PP5020E. Google this number and see if you can identify this chip.

Open a text file and type: "ECE 2500". When you save the file, find the size of the file in bytes. It should be 8 B.

It is estimated that there are about 200,000 international characters. UNICODE can handle only ¼ of them.

Codecs such as MP3 or AAC (iPod) use psycho acoustics and perceptual coding to compress the file to 10% of its original size.
Color codes
From: howstuffworks.com/lcd5.htm
iv. Red 1 B => 256 shades
v. Green 1 B => 256 shades
vi. Blue 1 B => 256 shades

Digital Logic Components (Process digital data)
1. **Register** (holds various forms of digital data)
2. **Port** (a register interfacing data to/from the outside world)
3. **ALU** (adds contents of 2 registers)
4. **Bus** (A path by which data may flow from one register to another in parallel)
5. **Encoder** (Encodes or compresses data)
6. **Decoder** (Decodes or expands data. Also used to make memory location selections)
7. **MUX** (Selects between many data sources)
8. **ROM** (An storage array that can be read word by word, chosen by an address)
9. **RAM** (A storage array that also can be written)
10. **USB cable** (A path by which data packets may be transferred serially to ports from a hub)
11. **Optical Disc Storage** (CD/DVD ROM from which blocks of data can be read)
12. **Hard Disk Drive** (A magnetic storage device from which blocks of data can be stored & read)
13. **USB drive** (A ROM device which can transfer data in blocks over a USB cable)
14. **Microcontroller** (A processing device consisting of an ALU, registers, ports and RAM)
15. **Microprocessor** (More powerful processor that has extensive memory and multiple ALUs)
16. **LCD display** (a display device which uses a 2d decoder array and a controller)

**NUMBER SYSTEM CONVENTIONS**

**Number systems table**: comparison of important number systems, including:
1. **decimal** (base 10)
2. **binary** (base 2)
3. **octal** (base 8), and
4. **hexadecimal** (base 16) number systems

The total number of possible shades of colors with a 24 bit color format is:
\[ 256^3 = 2^{24} = 16 \text{ M colors} \]
General Number Systems Representation
1. Juxtapositional notation for representation of a number = \( N \).
2. Polynomial representation for \( N \).

Number-base Conversion: The process of converting \( N \) from one number-base representation, to another. There are three cases to consider:
1. Power series method to convert \( N \) to base 10.
2. Divide/multiply method to convert base 10 \( N \) to any other base. (i.e. music sampling, above)
3. Base \( 2^k \) conversions: binary to hex (or octal).

Binary Addition of Integers:
1. Bit by bit addition right to left, with carry bits
2. Subtraction is done by adding numbers encoded in 2’s complement format.

2’s Complement Format: (i.e. “how to take a 2’s complement” of a number \( N \))
1. Complement all the bits of \( N \)
2. Add 1 to the result. This is \( N' \).
3. The sign of \( N \) (or \( N' \)) is shown by the most significant bit: 0 = “+”; 1 = “-“.

Note: \( N + N' = 0 \)

Error Correction Codes (ECC)
1. Provides self-correction of errors that occur in the data when transporting data.
2. Hamming code
   a. Compute HC for data-in
   b. Compute HC for data-out
   c. Compare differences in HC bits and add those positions to form bit error position
3. Other ECC: Reed-Solomon code

ECC in an iPod hard drive
From: [computer.howstuffworks.com/hard-disk7.htm](http://computer.howstuffworks.com/hard-disk7.htm)
1. Toshiba 1.8” platter stores up to 7,500 songs
2. Tracks are divided up into a number of fixed length sectors, consisting of
   a. Preamble (for head synchronizing)
   b. Data field
   c. ECC field (Hamming or Reed-Solomon)
3. Solid-state Drive: Memory is broken up into are divided up into a number of fixed length blocks, similar to the tracks of magnetic disks. Blocks consist only of the Data field and ECC field

Error correction is one of the most important advantages that digital systems provide over analog systems.

A preamble field is not necessary in a solid state drive because there is no mechanical motion to sync to.
**BOOLEAN ALGEBRA**

Binary numbers are also used as truth values, or logic values.

Logic defined: the process of classifying information.

Binary logic (or more commonly, digital logic) is the process of classifying information into two distinct classes, e.g.

(TRUE, FALSE) = truth values
(Yes, No)
(CLOSE, OPEN) = relay positions
blown, intact = fuse state
(ON, OFF) = switch positions
(1, 0) = binary numbers, or (Logic 1, Logic 0)

Logic design is based upon the three logic operators

Binary Logic Operations (Variables)
1. **AND:** \( z = x \cdot y = xy \)
2. **OR:** \( z = x + y \)
3. **NOT:** \( z = \bar{x} \), also written as \( z = x' \) in the text

Binary Logic Operations

<table>
<thead>
<tr>
<th>OR</th>
<th>XOR</th>
<th>AND</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 + 0 = 0</td>
<td>0 ( \oplus ) 0 = 0</td>
<td>0 ( \cdot ) 0 = 0</td>
</tr>
<tr>
<td>0 + 1 = 1</td>
<td>0 ( \oplus ) 1 = 1</td>
<td>0 ( \cdot ) 1 = 0</td>
</tr>
<tr>
<td>1 + 0 = 1</td>
<td>1 ( \oplus ) 0 = 1</td>
<td>1 ( \cdot ) 0 = 0</td>
</tr>
<tr>
<td>1 + 1 = 1</td>
<td>1 ( \oplus ) 1 = 0</td>
<td>1 ( \cdot ) 1 = 1</td>
</tr>
</tbody>
</table>

Two Level Logic Circuits with AND/OR gates:
From: [computer.howstuffworks.com/boolean1.htm](computer.howstuffworks.com/boolean1.htm)
Examples will be given to describe
1. **AND-OR** circuits (**sum of product = SOP**)
2. **OR-AND** circuits (**product of sum = POS**)
3. **XOR-XOR** circuits

These circuits can also be described algebraically with the use of an algebra system for logic variables called…

Notes:

1 and 0 no longer represent numbers, but logical values like true and false.

The word “gate” comes from the usage of a fence gate which can be open or closed.

A digital logic circuit is a combination of gates that do a specified logic function
Boolean Algebra

Fundamental properties of Boolean Algebra: Each x, y and z are elements of $B = \{0, 1\}$

1. **Identities**: (P3, P4) (Dual)
   \[
   x + 0 = x \quad x \cdot 1 = x \\
   x + 1 = 1 \quad x \cdot 0 = 0
   \]

2. **Commutativity**: (P1)
   \[
   x + y = y + x \quad x \cdot y = y \cdot x
   \]

3. **Associativity**: (P2)
   \[
   x + (y + z) = (x + y) + z \quad x \cdot (y \cdot z) = (x \cdot y) \cdot z
   \]

4. **Distributivity**: (P8)
   \[
   x + (y \cdot z) = (x + y)(x + z) \quad x \cdot (y + z) = x \cdot y + x \cdot z
   \]

5. **Existence of the complement**: (P5) There exists an element called $\bar{x}$ such that
   \[
   x + \bar{x} = 1 \quad x \cdot \bar{x} = 0
   \]

6. **Involution**: (P7)
   \[
   \bar{\bar{x}} = x
   \]

7. **Absorption**: (P12)
   \[
   x + xy = x \quad x \cdot (x + y) = x
   \]

8. **Adjacency**: (P9)
   \[
   xy + \bar{x}y = x \quad (x + y)(x + \bar{y}) = x
   \]

9. **DeMorgan's Law**: (P11)
   \[
   x + y + z = \bar{x} \cdot \bar{y} \cdot \bar{z} \quad x \cdot y \cdot z = \bar{x} + \bar{y} + \bar{z}
   \]

Boolean Functions and Logic Circuits
Examples will now be given to show
1. Drawing a logic circuit from a Boolean function
2. Simplifying a Boolean functions by pattern matching the Boolean properties and theorems.
3. Developing truth tables

**LOGIC GATES AND CIRCUITS**

DeMorgan’s Laws Shows Equivalent Graphical Symbols for Logic Gates: Examples will be given to describe
1. **NAND** gate drawn with an **OR** symbol
2. **NOR** gate drawn with an **AND** symbol
3. **NOTs** built from **NANDs** and **NORs**
Two Level Logic Circuits with Other Gates:
Examples will be given to describe
1. **AND-OR** circuits (sum of product = SOP)
2. **OR-AND** circuits (product of sum = POS)
3. **NAND-NAND** circuits = **AND-OR** circuits
   (Leading NAND looks like an OR)
4. **NOR-NOR** circuits = **OR-AND** circuits
   (Leading NOR looks like an AND)
5. **XOR-XOR** circuits = larger **XOR** gates

MOS Implementation of Logic Gates
Examples will be shown how to implement **NAND**, **NOR**, and **NOT** gates from elementary **NMOS** transistors.
1. **NMOS** transistors: the Gate voltage controls determines the three states of the transistor as seen between the Source and Drain: OFF state, **ON** state and Resistive state.
2. **NAND**, **NOR** and **NOT** gates can be constructed from two or three transistors. **AND** and **OR** gates require at least five **NMOS** transistors.
3. **CMOS** (complementary MOS) technology utilizes **NMOS** transistors supplemented with **PMOS** transistors (made from a complementary process) to limit current flow and thus power consumption in the logic gates. More transistors are required, because **NMOS** and **PMOS** occur in pairs.

**MINTERMS AND K-MAPS**

Minterm Properties and Notation
1. A **minterm** is a product term which produces a single 1 in a truth table.
2. The minterm which yields a 1 in row i is denoted as minterm $m_i$, $0 \leq i \leq 2^n - 1$
3. Minterm list form for a Boolean function:
   \[ f = \sum m(\text{row# s where } f = 1) \]
   \[ \bar{f} = \sum m(\text{row# s where } f = 0) \]

**Notes:**
A function having $m$ 1s in a truth table has $m$ minterms.
The importance of minterms is in their ability to be able to form SOP algebraic expressions from the 1s in a truth table.
Karnaugh Map (K-Map) Properties
1. Each cell in a K-map for a function \( f \) corresponds to a row of the truth table describing \( f \).
2. Cell \( i \) is a placemark for minterm \( m_i \). K-map labels identify the coincidence of literals for each minterm.
3. Adjacent minterms can be combined into a simpler product term, also by using the coincidence of literals technique.
4. Cells over the left and right edges, or the upper and lower edges are defined to be adjacent.

Procedure for Plotting SOP Functions on a K-map
1. Determine the minterms \( m_i \) contained in \( f \) (found by observing the rows where \( f = 1 \) in the truth table).
2. Plot the 1’s of the function to be minimized on the K-map. That is, for each minterm \( m_i \) in \( f \), enter a 1 in cell \( i \).
3. For each don’t care contained in \( f \), enter a \( d \) (or \( x \)) in the associated K-map cell (see below).

Procedure for reading minimal SOP expressions of functions from the K-map
1. Draw loops around adjacent 1-entries (cells with 1’s) in largest groups possible. Group size must be a power of two (e.g. 1 cell, 2 cells, 4 cells, 8 cells, etc.)
2. 1-entries not adjacent to other 1-entries are circled as groups of one.
3. Discard redundant groupings (those entries entirely covered by other groups.)
4. For each group, read off the coincident literals covering the group, by exploiting K-map labels. AND those literals together to form products; OR the resulting products to create a sum.

Map Simplification Resulting from Don’t Cares
1. Don’t care = \( d \) (or \( x \)) = \{0,1\} (either a 0 or a 1)
2. Group 1-entries as before, but also include any \( d \)-entries which serve to increase the group size of the 1-entries. Treat unused \( d \)-entries as 0-entries.
3. Never group cells consisting entirely of don’t care entries. This results in a redundant group.
MAXTERMS & K-MAPS

Maxterm Properties and Notation
1. A Maxterm is a sum term which produces a single 0 in a truth table.
2. Maxterm which yields a 0 in row i is denoted as maxterm $M_i$, $0 \leq i \leq 2^n - 1$
3. Maxterm list form for a Boolean function:
   \[ f = \prod M(\text{row} \# \text{s where } f = 0) \]
   \[ \bar{f} = \prod M(\text{row} \# \text{s where } f = 1) \]

Other Properties of Minterms and Maxterms
1. $\sum m(\text{all row} \# \text{s}) = 1$
   $\prod M(\text{all row} \# \text{s}) = 0$
2. $m_i = M_i$
   $\bar{M_i} = m_i$

Procedure for plotting and reading minimal POS expressions of functions from the K-map
1. Plot the 0’s of the function to be minimized on a K-map. That is, for each maxterm $M_i$ in $f$, enter a 0 in cell $i$.
2. Draw loops around adjacent 0-entries.
3. For each group, read off the complement of the coincident literals covering the group, by exploiting K-map labels. OR those literals together to form sums; AND the resulting sums to create a product.

Notes:
A function having $M$ 0s in a truth table has $M$ Maxterms.
The importance of Maxterms is in their ability to be able to form POS algebraic expressions from the 0s in a truth table.
Boolean functions may be synthesized for any given truth table by writing that function in minterm or maxterm list form. The result then can be simplified using Boolean algebra to obtain a simple expression in SOP (for minterms) or POS (for maxterms) form.