LOGIC DESIGN AND NUMBER SYSTEMS

Purpose: These course notes are intended for the instructor. It is not a replacement for the notes taken by the students in the classroom. The students can use these notes as a complement to their own notes or to the text-book.


COMPUTER ORGANIZATION

Computer consists of: CPU, Main Memory, Input/Output Processor and System Interconnection Circuits. CPU consists of: ALU (Arithmetic Logic Unit), Control Unit, Registers, and Internal Inconnection Circuits. I/O Processor: Video controller, harddisk and floppy controllers, keyboard and mouse controllers, ... Input Devices: Keyboard, mouse, harddisk, floppy disk, microphone, modem, scanner, ... Output Devices: Terminal, printer, harddisk, floppy disk, sound speaker, modem, ...
LOGIC GATES

<table>
<thead>
<tr>
<th>Name</th>
<th>Graphic Symbol</th>
<th>Algebraic Function</th>
<th>Truth Table</th>
</tr>
</thead>
</table>
| AND             | ![AND Symbol]  | \( x = A \cdot B \)  | \[
|                 |                | or \( x = A \overline{B} \) | \[
|                 |                | \[
| OR              | ![OR Symbol]   | \( x = A + B \)    | \[
| Inverter        | ![Inverter Symbol] | \( x = A' \) | \[
| Buffer          | ![Buffer Symbol] | \( x = A \) | \[
| NAND            | ![NAND Symbol] | \( x = (A \cdot B)' \) | \[
| NOR             | ![NOR Symbol]  | \( x = (A + B)' \) | \[
| Exclusive-OR    | ![XOR Symbol]  | \( x = A \oplus B \) | \[
| Exclusive-NOR   | ![XOR Symbol]  | \( x = (A \oplus B)' \) | \[

BOOLEAN ALGEBRA

Boolean Algebra is an algebra that deals with binary variables and logic operations. The three basic logic operations are AND, OR and complement (invert).

Boolean Function is a function that can be expressed algebraically with binary variables, the logic operation symbols, parentheses and equal sign. For a given value of variables, the Boolean function can be either 1 or 0.

Example:  \( F = x + y' \cdot z \)

Truth Table and Logic Diagram for \( F = x + y' \cdot z \)

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Basic Identities of Boolean Algebra:

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
| 1 | \(x + 0 = x\) | 2 | \(x . 0 = 0\) | 3 | \(x + 1 = 1\) | 4 | \(x . 1 = x\) | 5 | \(x + x = x\) | 6 | \(x . x = x\) | 7 | \(x + x' = 1\) | 8 | \(x . x' = 0\) | 9 | \((x')' = x\) | 10 | \(x + y = y + x\) | 11 | \(x y = y x\) | 12 | \(x + (y + z) = (x + y) + z\) | 13 | \(x (y z) = (x y) z\) | 14 | \(x (y + z) = x y + x z\) | 15 | \(x + y z = (x + y) (x + z)\) | 16 | \((x + y)' = x' y'\) | 17 | \((x y)' = x' + y'\) | 18 | De Morgan's Rules

Algebraic Manipulation: Application of Basic Identities of Boolean Algebra.

Simplify the following Boolean function by using basic identities:

\[ F = X' Y Z + X' Y Z' + XZ \]

(1st make the truth table for \(F\), then again after simplification.)

Priority of Boolean Operators:

1. ( )
2. NOT (invert)
3. AND
4. OR

Complement of a Boolean Function:

The complement of a function \(F\) is obtained from the interchange of 1’s to 0’s and 0’s to 1’s in the values of \(F\) in the truth table.

In finding the boolean form of the complement of a function, we apply the De Morgan's rules.

Example:

Find the complement of the following functions:

\[ F_1 = X' Y Z' + X' Y' Z \]
\[ F_2 = X(Y' Z' + Y Z) \]

Standard Forms:

- Sum of Products. Format: OR of AND’s.
- Product of Sums. Format: AND of OR’s.
  - Example: \(F = (X' + Y + Z') . (X' + Z) . (X + Y')\)

Minterms and Maxterms:

Maxterm is the complement of Minterm.

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>Product Term</th>
<th>Symbol</th>
<th>Sum Term</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(X' Y' Z')</td>
<td>(m_0)</td>
<td>(X + Y + Z)</td>
<td>(M_0)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>(X' Y' Z)</td>
<td>(m_1)</td>
<td>(X + Y + Z')</td>
<td>(M_1)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>(X' Y Z')</td>
<td>(m_2)</td>
<td>(X + Y' + Z)</td>
<td>(M_2)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>(X' Y Z)</td>
<td>(m_3)</td>
<td>(X + Y' + Z')</td>
<td>(M_3)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>(X Y' Z')</td>
<td>(m_4)</td>
<td>(X' + Y + Z)</td>
<td>(M_4)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>(X Y' Z)</td>
<td>(m_5)</td>
<td>(X' + Y + Z')</td>
<td>(M_5)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>(X Y Z')</td>
<td>(m_6)</td>
<td>(X' + Y' + Z)</td>
<td>(M_6)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>(X Y Z)</td>
<td>(m_7)</td>
<td>(X' + Y' + Z')</td>
<td>(M_7)</td>
</tr>
</tbody>
</table>

Representation using minterms:

\[ F (x, y, z) = \Sigma m (1, 4, 5, 6, 7) \]

(Make the truth table for this function, and write sum of products.)
MAP SIMPLIFICATION

Karnaugh Maps:

2-Variable Karnaugh Map

\[
\begin{array}{c|cc}
A & 0 & 1 \\
\hline
B & 0 & 1 \\
& 2 & 3 \\
\end{array}
\]

4-Variable Karnaugh Map

\[
\begin{array}{c|c|c|c|c|c}
A & 00 & 01 & 11 & 10 \\
\hline
B & 0 & 1 & 3 & 2 \\
C & 4 & 5 & 7 & 6 \\
D & 12 & 13 & 15 & 14 \\
& 8 & 9 & 11 & 10 \\
\end{array}
\]

3-Variable Karnaugh Map

\[
\begin{array}{c|c|c|c|c}
A & 0 & 1 & 3 & 2 \\
\hline
B & 4 & 5 & 7 & 6 \\
C & 00 & 01 & 0011 & 0010 \\
D & 0100 & 0101 & 0111 & 0110 \\
& 1100 & 1101 & 1111 & 1110 \\
& 1000 & 1001 & 1011 & 1010 \\
\end{array}
\]

4-Variable Karnaugh Map

Sum-of-products Simplification by Karnaugh map:
- Make rectangular or square groups of 8, 4, 2 cells;
- Start with biggest groups (8 for example), then continue with smaller groups;
- Make the groups as big as possible;
- If all elements of a group exist in other groups, remove (do not use) that group;
- If there are "don't cares", use them to make bigger groups;
- All don't cares need not be used, use only if don't cares serve to make bigger groups.

Example: \[ F (x, y, z) = \sum m (1, 4, 5, 6, 7) \]
\[ F (x, y, z) = x + y' z \]

Example: \[ F (A, B, C, D) = \sum m (0, 1, 2, 5, 8, 9, 10) \]
\[ F (A, B, C, D) = B' C' + B' D' + A' C' D \]

Product of Sums Simplification:

Example: \[ F (A, B, C, D) = \sum m (0, 1, 2, 5, 8, 9, 10) \]
\[ F' = A B + C D + B' D' \quad \text{(by grouping zero's)} \]
\[ F = (A B + C D + B' D')' \quad \text{(invert } F' \text{ to get } F) \]
\[ F = (A B)' . (C D)' . (B' D')' \quad \text{(apply De Morgan)} \]
\[ F = (A' + B') . (C' + D') . (B' + D) \quad \text{(re-apply De Morgan)} \]

Don't Cares:
\[ F (A, B, C) = \sum m (0, 2, 6) \]
\[ d (A, B, C) = \sum m (1, 3, 5) \]

Insert the "Don't Cares" into the groups if it helps to make bigger groups.
It is not mandatory to include all the "Don't Cares". Some of them may be unused.
Implementing the Logic Circuit for a Boolean Function with only NAND gates or NOR gates:

To implement only with NAND gates:
1. Find the function as sum-of-product expression:
2. Invert twice,
3. Apply De Morgan only once to inner invert. De Morgan’s rules convert "Sum" expressions to "Product" expressions.

Example:

\[ F = A' \cdot C' \cdot D + B' \cdot C' + B' \cdot D' \]

Invert twice:
\[ F = ((A' \cdot C' \cdot D + B' \cdot C' + B' \cdot D')')' \]

Apply De Morgan for inner invert:
\[ F = ((A' \cdot C' \cdot D)' \cdot (B' \cdot C')' \cdot (B' \cdot D')')' \]

\[ F = y' + w' \cdot x' + x' \cdot z' \]

To implement only with NOR gates:
4. Find the function as product-of-sums expression:
   - write the complement \( F' \) of the function as sum-of-products,
   - write \( F \) as the complement of \( F' \),
   - by applying De Morgan rules twice, find the function as product-of-sums expression,
5. then invert twice,
6. then apply De Morgan only once to inner invert.

Example:

\[ F' = A \cdot B' + A' \cdot C + B \cdot C \]
\[ F = (A \cdot B' + A' \cdot C + B \cdot C)' \]
\[ F = (A \cdot B')' \cdot (A' \cdot C)' \cdot (B \cdot C)' \]
\[ F = (A' + B) \cdot (A + C') \cdot (B' + C') \]
\[ F = ((A' + B) \cdot (A + C') \cdot (B' + C')')' \]
\[ F = ((A' + B)' + (A + C')' + (B' + C')')' \]

\[ F \] as sum-of-products
\[ F' \]
\[ F = (F')' \]
Apply De Morgan for the 1st time
Apply De Morgan for the 2nd time: \( F \) is product-of-sums
Double invert
Apply De Morgan to inner invert. Here is the result.
COMBINATIONAL CIRCUITS

One boolean function or truth table per output.
The combinational circuit can be described verbally, and in this case, one has to find the boolean functions or the truth tables.

HALF ADDER

Half adder is a combinational circuit which is capable of adding two bits.

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>C</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ S = x' y + x y' = x \oplus y \]
\[ C = x y \]

FULL ADDER

Full adder is a combinational circuit which is capable of adding three bits.

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>C</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ S = x \oplus y \oplus z \]
\[ C = x y + (x' y + x y') z = x y + (x \oplus y) z \]
DECODERS

A decoder is a combinational circuit that converts binary information from \( n \) coded inputs to a maximum of \( 2^n \) unique outputs.

Example: 3-to-8 Line Decoder.

Truth Table for 3-to-8 Line Decoder:

<table>
<thead>
<tr>
<th>Enable</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>A2</td>
<td>A1</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

NAND Gate Decoder: NAND gates are used instead of AND gates.

Example: 2-to-4 Line Decoder with NAND gates.

Decoder Expansion:

A certain size decoder can be constructed using two half-sized decoders. Example: 3x8 decoder constructed by two 2x4 decoders. (Figure 2-3, page 46)

ENCODERS

An encoder is a circuit which converts a maximum of \( 2^n \) unique inputs to \( n \) outputs which are equivalent to binary code for corresponding input. Performs inverse operation of a decoder. \( 2^n \) inputs \( \rightarrow \) \( n \) outputs

Truth Table for 3-to-8 Line Encoder:

<table>
<thead>
<tr>
<th>Enable</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>D0</td>
<td>D1</td>
</tr>
<tr>
<td>0</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Lecture Notes – Logic Design and Number Systems

Halil Özmen - 13/05/2009
MULTIPLEXERS (MUX)

A multiplexer is a combinational circuit that receives binary information from one of $2^n$ input data lines and directs it to a single output line. A multiplexer with $2^n$ input lines requires "n" selection lines.

Example: 4-to-1 Multiplexer (Figure 2-4, page 48)

Function Table for 4-to-1 Line Multiplexer:

<table>
<thead>
<tr>
<th>Select</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_1</td>
<td>S_0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Example Combinational Circuits:

Example 1:
Built a combinational circuit that generates the 2's complement of its 3-bit inputs.
(a) Prepare the truth table, (b) Find the most simplified sum-of-products form of the output equations (using Karnaugh map method), (c) Draw combinational circuit using the found output functions.

Example 2:
For the below combinational circuit, (a) Prepare the truth table, (b) Find the most simplified sum-of-products form of the output equations (using Karnaugh map method), (c) Draw combinational circuit using the found output functions.
SEQUENTIAL CIRCUITS

Definition:
A sequential circuit consists of a combinational circuit and storage elements that together form a feedback system.

Storage Elements:
The storage elements are devices capable of storing binary information within them. The binary information stored at any given time defines the state of the sequential circuit.

Synchronous sequential circuit: is a system whose behaviour can be defined from the knowledge of its signals at discrete instants of time.

Asynchronous sequential circuit: The behaviour of an asynchronous sequential circuit depends upon the order in which the inputs change, and the state of the circuit can be affected at any instant of time.

A synchronous sequential circuit employs signals that affect the storage elements only at discrete instants of time. Synchronization is achieved by a timing device called a clock generator that produces a periodic train of clock pulses. The outputs of storage elements change only when clock pulses are present.

FLIP-FLOPS
Each flip-flop is capable of holding one bit of information. Flip-flops are clocked storage elements.

Flip-flops are not combinational circuits, because they have clock inputs, and the results change only at the time of clock pulses.

The characteristic tables of flip-flops are used to determine the next state when the flip-flop input(s) are known.

SR Flip-flop
Characteristic Table:

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Q(t+1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q(t)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>X</td>
</tr>
</tbody>
</table>

D Flip-flop
Characteristic Table:

<table>
<thead>
<tr>
<th>D</th>
<th>Q(t+1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
**JK Flip-flop**

Characteristic Table:

<table>
<thead>
<tr>
<th>J</th>
<th>K</th>
<th>Q(t+1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q(t)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Q'(t)</td>
</tr>
</tbody>
</table>

No Change
Clear to 0
Set to 1
Complement

**T Flip-flop**

Characteristic Table:

<table>
<thead>
<tr>
<th>T</th>
<th>Q(t+1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Q(t)</td>
</tr>
<tr>
<td>1</td>
<td>Q'(t)</td>
</tr>
</tbody>
</table>

No Change
Complement

**Flip-Flop Input Equations:**

In a sequential circuit, the boolean functions of the flip-flop inputs are called "Flip-flop Input Equations".

The "Flip-flop Input Equations" are boolean functions that are in terms of inputs and flip-flop outputs (present state).

When a sequential circuit is given, by examining (tracing) the combinational part, Flip-flop Input Equations can be determined.

\[
J_A = X \oplus Y \\
K_A = Y \\
D_B = A' + Y
\]

**Excitation Tables:**

The excitation table shows what should be the inputs of flip-flops for various transitions between current state to the next state.

Excitation tables are used to find the flip-flop input values when the present and next states are known.

**SR Flip-flop:**

Excitation Table:

<table>
<thead>
<tr>
<th>Q(t)</th>
<th>Q(t+1)</th>
<th>S</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>d</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>d</td>
<td>0</td>
</tr>
</tbody>
</table>

**JK Flip-flop:**

Excitation Table:

<table>
<thead>
<tr>
<th>Q(t)</th>
<th>Q(t+1)</th>
<th>J</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>d</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>d</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>d</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>d</td>
<td>0</td>
</tr>
</tbody>
</table>

**D Flip-flop:**

Excitation Table:

<table>
<thead>
<tr>
<th>Q(t)</th>
<th>Q(t+1)</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**T Flip-flop:**

Excitation Table:

<table>
<thead>
<tr>
<th>Q(t)</th>
<th>Q(t+1)</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
State Table:
The states of a sequential circuit are determined by the outputs of the Flip-flops in that sequential circuit. If there are "n" flip-flops in a sequential circuit, there are at most $2^n$ states.
The state table of a sequential circuit consists of columns for: Inputs, Present States, Next States and Outputs.
In the state table, the next states and outputs can be deduced from inputs and present states.
If there are $K$ inputs, and $N$ flip-flops, then the state table has $2^{(K+N)}$ rows.
For the following sequential circuit, and write the state table. First find the flip-flop input equations!

### J = X A + X' Y
### K = B
### D = X' Y + A'
### Z = Y A

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Present State</th>
<th>Next State</th>
<th>Outp.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X$</td>
<td>$Y$</td>
<td>$A$</td>
<td>$B$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
State Diagram:
All the information in a state table can be mapped to a state diagram, in which it is easier to see the transitions among states depending on inputs and generating the outputs.
For the circuit and state table of the above example, the state diagram is as follows:

Sequential Circuit Analysis:
When a sequential circuit is given, do the followings:
1. Find flip-flop input equations;
2. Prepare the state table;
3. Draw the state diagram.

Sequential Circuit Synthesis (Design):
When the description of a sequential circuit is given verbally, the followings has to be done to draw (or build) the circuit:
1. Draw the state diagram;
2. Make the state table, and fill the inputs, the present states, the next states and the outputs;
3. Using the present states, next states and the flip-flop excitation tables, fill the flip-flop inputs columns in the state table;
4. Using the flip-flop inputs columns in the state table, find the flip-flop input equations by the Karnaugh Map method;
5. Draw the sequential circuit using the found flip-flop input equations.
TIMING DIAGRAMS:
At the beginning, the signals for flip-flop inputs are drawn till the first clock pulse. Then for each clock period:
- the states (flip-flop outputs) are found,
- and then signals for flip-flop inputs are drawn,

Example: Draw the timing diagram for $A$, $B$, $J_A$, $K_A$, and $D_B$ of the sequential circuit given below, according to the given inputs $X$ and $Y$, and initial states of $A$ and $B$ (which are both 0).
(Hint: First, write the boolean expressions for the inputs of the flip-flops.)

\[ J_A = X + B \]
\[ K_A = Y \]
\[ D_B = Y \oplus A \]
REGISTERS

A register is a group of flip-flops with each flip-flop capable of storing one bit of information. An "n-bit" register has n flip-flops, and is capable of storing any binary information of "n" bits. A register may also have extra combinational circuit to perform certain data processing tasks.

Example: 4-bit Register with asynchronous clear (Figure 2-6, page 51)

"Registers with Parallel-Load" have the capability of loading their content parallely in one clock cycle.
Example: 4-bit Register with parallel load (Figure 2-7, page 52)

SHIFT REGISTERS

A register capable of shifting its binary information in one or both directions is called a shift register.

Example: 4-bit Shift Right Register
Inputs: I: Serial input, S: Shift Right input
State is determined by: \( A_3, A_2, A_1, A_0 \).
S is the "Shift Right" signal.
If S = 1, the register shifts to right by one bit, I is input from left.
If S = 0, the register keeps its content.
Therefore: if S = 0: \( A_3(t+1) = A_3(t) \), \( A_2(t+1) = A_2(t) \), \( A_1(t+1) = A_1(t) \), \( A_0(t+1) = A_0(t) \)
If S = 1: \( A_3(t+1) = I \), \( A_2(t+1) = A_3(t) \), \( A_1(t+1) = A_2(t) \), \( A_0(t+1) = A_1(t) \)
Therefore: \( A_3(t+1) = S'.A_3(t) + S.I \), \( A_2(t+1) = S'.A_2(t) + S.A_3(t) \)
\( A_1(t+1) = S'.A_1(t) + S.A_2(t) \), \( A_0(t+1) = S'.A_0(t) + S.A_1(t) \)

4-bit Shift Right Register using D flip-flops (without Shift signal) (Figure 2-8, page 53)

4-bit Shift Right Register using D flip-flops (with Shift signal)
A shift register which shifts its information in only one direction is called **Unidirectional** Shift Register. If it shifts in both directions, it is called **Bidirectional** Shift Register.

**GENERAL PURPOSE REGISTERS**

The most general shift register has the following capabilities:

1. An input for clock pulses to synchronize all operations (Clock input).
2. A shift-right operation and a serial input line associated with the shift-right.
3. A shift-left operation and a serial input line associated with the shift-left.
4. A parallel load operation and "n" input lines associated with the parallel transfer.
5. "n" parallel output lines.
6. A control state input that leaves the information in the register unchanged even though clock pulses are applied continuously.

**COUNTERS**

A register that goes through a predetermined sequence of states upon the application of input pulses is called a counter. Binary counters are counters that follows the binary number sequence.

Example: 4-bit Synchronous Binary Counter (Figure 2-10, page 57)

Binary Counter with Parallel Load: capable of setting the binary counter from input lines.
Example: 4-bit Binary Counter with parallel load and synchronous clear (Figure 2-11, page 59)
MEMORY UNIT, RAM, ROM

Bit
Byte - 8 bits
Word: Word size depends on computer (processor).
  2-byte word (16-bit computer), 4-byte word (32-bit computer), etc.

RAM: Random Access Memory
ROM: Read Only Memory

RAM (block Diagram):

\[ 2^k \times n \text{ memory unit (i.e. number of words x word length)} \]

ROM: Asynchronously, ROM data output lines automatically provide the "n" bits of the word selected by address lines.

ROM types: PROM (Programmable ROM), EPROM (Erasable PROM), EEPROM (Electrically Erasable PROM).

Building memories with smaller capacity memory units:

Example: Build a memory of 1024 x 16 by using 256 x 8 memory units.
DATA TYPES

In digital computers, information is stored in **binary formats**, in **memory** and **processor registers**. Various data types are represented in binary-coded form in computer registers and in memory.

Data types may be:
1. Numbers used in arithmetic computations.
2. Letters of alphabet (and other characters including punctuation marks and special symbols !, @, $, %, ^, & , etc.) used in data processing
3. Other discrete symbols used for specific purposes.

Registers contain either data or control information.
Control information is a bit or group of bits used to specify the sequence of command signals needed for manipulation of the data in other registers. Usually there is one or more special registers dedicated to hold the control information.

Except binary numbers (integers), all types of data are represented in binary-coded forms.

NUMBER SYSTEMS

A number system of base "r" (or radix "r") is a system that uses distinct symbols for "r" digits.
Numbers are represented by a string of digit symbols.

Decimal Number System has 10 digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
Binary Number System has 2 digits: 0 and 1.
Other number systems used in computer science are: Base 8 (Octal) and Base 16 (Hexadecimal).

Value of a digit is dependent on the position it has in the number.

**Decimal:**

\[
\begin{align*}
72045 &= 5 \times 10^0 = 5 \\
&= 4 \times 10^1 = 40 \\
&= 0 \times 10^2 = 0 \\
&= 2 \times 10^3 = 2000 \\
&= 7 \times 10^4 = 70000 \\
\end{align*}
\]

**Binary:**

\[
\begin{align*}
110101 &= 1 \times 2^6 = 1 \\
&= 0 \times 2^5 = 0 \\
&= 1 \times 2^4 = 4 \\
&= 0 \times 2^3 = 0 \\
&= 1 \times 2^2 = 16 \\
&= 1 \times 2^1 = 32 \\
\text{Total} &= 53
\end{align*}
\]

**Hexadecimal:**

\[
\begin{align*}
(AF3)_{16} &= 3 \times 16^0 = 3 \times 1 = 3 \\
&= F \times 16^1 = 15 \times 16 = 240 \\
&= A \times 16^2 = 10 \times 256 = 2560 \\
\text{Total} &= 2803
\end{align*}
\]

**Decimal to Binary Number Conversion:**

\[
\begin{align*}
217 &= 11010111 \\
108 &= 10001000 \\
54 &= 110110 \\
27 &= 11011 \\
13 &= 1101 \\
6 &= 110 \\
3 &= 11 \\
1 &= 1 \\
0 &= 0 \\
\text{(217)}_{10} &= (11011001)_{2}
\end{align*}
\]
Binary to Hexadecimal and Octal Number Conversions:

<table>
<thead>
<tr>
<th>Octal:</th>
<th>2</th>
<th>7</th>
<th>1</th>
<th>5</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary:</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Hexadecimal:</td>
<td>2</td>
<td>E</td>
<td>6</td>
<td>A</td>
<td></td>
</tr>
</tbody>
</table>

\[(10111001101010)_{8} = (2E6A)_{16} = (27152)_{10} = (11882)_{10}\]

Binary Addition and Multiplication:
Give examples.

<table>
<thead>
<tr>
<th>[10110101]</th>
<th>[11011110]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[\text{1011}]</td>
<td>[\text{1101}]</td>
</tr>
<tr>
<td>[\text{------}]</td>
<td>[\text{------}]</td>
</tr>
<tr>
<td>[110010011]</td>
<td></td>
</tr>
</tbody>
</table>

Decimal Representation:

BCD = Binary Coded Decimal
In BCD, every decimal digit is represented by 4-bits.

<table>
<thead>
<tr>
<th>Decimal Number</th>
<th>BCD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
</tr>
<tr>
<td>10</td>
<td>0001 0000</td>
</tr>
<tr>
<td>50</td>
<td>0101 0000</td>
</tr>
<tr>
<td>99</td>
<td>1001 1001</td>
</tr>
<tr>
<td>248</td>
<td>0010 0100 1000</td>
</tr>
<tr>
<td>7249</td>
<td>0111 0010 0100 1001</td>
</tr>
</tbody>
</table>

Alphanumeric Representation:

Characters: Alphabetic: A - Z, a - z
Numeric: 0 - 9
Special Characters: \.,;":;\?%^&()[]+=+-*/<>~|#$

ASCII character representation.

COMPLEMENTS OF NUMBERS

\((r-1)\)'s Complement:
9's complement in Decimal Number System, 1's complement in Binary Number System.
Number \(N\) having \(n\) digits in base \(r\): \((r-1)'s\ Complement = (r^n - 1) - N\)

Example: 9's complement. \(830594\)
\[169405 = (10^6 - 1) - 830594 = 999999 - 830594\]

Example: 1's complement. \(10110010110101\)
\[01001101001010 = (2^{14} - 1) - 10110010110101\]

\(r's\ Complement:
10's complement in Decimal Number System, 2's complement in Binary Number System.
Number \(N\) having \(n\) digits in base \(r\): \(r's\ Complement = r^n - N\)
Converts \((r-1)'s\ complement to \(r's\ complement by adding 1.\)

Example: 10's complement. \(830594\)
\[169406\]

Example: 2's complement. \(10110010110101\)
\[01001101001011\]
Substraction of Unsigned Numbers:
M - N (in base r)
1. Add M to r's complement of N. M + (r^n - N) = M - N + r^n
2. If M >= N, then the sum will produce an end carry r^n which is discarded.
3. If M < N, then the sum does not produce an end carry. It is equal to r^n - (N - M), which is r's complement of (N - M). To obtain the answer in a familiar form, take r's complement of the sum and place a negative sign in front.

Binary Substraction using 2's complement:
M - N: (M is minuend, N is subtrahend)
a) Make the length of the numbers equal by padding zero's at left side of the shorter number.
b) Take 2's complement of subtrahend.
c) Add minuend M and 2's complement of subtrahend N.

Example: 110100010101 - 10110001
First, increase length of subtrahend to the length of minuend: 000010110001
2's complement of 000010110001 = 111101001111

\[
\begin{array}{c}
110100010101 \\
+ 111101001111 \\
\hline
111001001000
\end{array}
\]
Discard carry:

11000100100

SIGNED NUMBERS
When an integer number is positive, the sign is represented by 0 and the magnitude by a positive binary number. When the number is negative, the sign is represented by 1 but the number may be represented in one of three possible ways:
1. Signed-magnitude representation,
2. Signed-1's complement representation,
3. Signed-2's complement representation,
Signed-2's complement representation is the most common one.
The left-most bit is used as the sign bit in signed numbers.
If a signed number is stored in 16-bit register, the register holds 15 significant bits, plus a sign bit.
A 16-bit register can hold signed numbers between the ranges \(-2^{15}\) and \(2^{15} - 1\).
Example: 8-bit signed numbers: 0 = 00000000, 1 = 00000001, 127 = 01111111, -1 = 11111111, -2 = 11111110, -128 = 10000000.

OVERFLOW:
An overflow occurs when an arithmetic operation produces a result which does not fit in the register. I.e. when the result needs more bits to represent than the number of bits available in the register.