Microcomputers

Number Systems and Digital Logic Review

Outline

• Number systems and formats
  – Common number systems
  – Base Conversion
  – Integer representation
  – Signed integer representation
  – Binary coded decimal (BCD) representation
  – Binary and Hex arithmetic

• Logic circuits/Boolean algebra
  – Switch inputs
  – NMOS and PMOS transistors and circuits
  – Truth tables
  – Logic networks
  – Tri-state devices
  – Basic combinatorial circuits
  – Clock signals
  – Storage elements
  – Basic sequential circuits
Review of Number Systems and Formats

- Computers are based on devices that can take only two states (0 or 1)
  - A single 0 or 1 is called a bit (Binary Digit)
  - Bits may be concatenated to represent many different entities
    - Characters, integers, floating point numbers, etc.
- In general, $n$ bits can represent $2^n$ distinct entities (values)

Review of Number Systems and Formats

- Example
  - $n=0$ $2^n=1$
  - $n=1$ $2^n=2$
  - $n=2$ $2^n=4$
  - $n=3$ $2^n=8$
  - $n=4$ $2^n=16$
- Strings of bits can be used to represent numbers, letters, etc.
- Numbers are associated with bit combinations according to specific number formats.
Basic Number Formats

- Binary (integer)
  - Positive only
  - Signed/unsigned
- Fractional representation
  - Fixed point notation
- Floating point (real)
  - In the C programming language this would be data types `float` and `double`
- Binary coded decimal (BCD)

Binary (Integer) Format

- Non-negative integers are represented by choosing a base, \( x \), and \( x \) different symbols called digits
- A string of these digits will represent a number
- In general, the string of digits:

\[
a_{n-1}a_{n-2}a_{n-3}......a_2a_1a_0
\]

would represent the, base \( x \), number

\[
a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + a_{n-3}x^{n-3} + ...... + a_2x^2 + a_1x^1 + a_0x^0
\]
**Binary (Integer) Format**

- example: base=10 number=65308 would be written as:
  \[6 \times 10^4 + 5 \times 10^3 + 3 \times 10^2 + 0 \times 10^1 + 8 \times 10^0\]
- But, computers are built on 2-state devices and work with BINARY (i.e. BASE 2) numbers:
- example: base=2 number=10110 would be written as:
  \[1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0\]

---

**Number Systems**

- To talk about binary data, we must first talk about number systems
- The decimal number system (base 10) you should be familiar with
  - A digit in base 10 ranges from 0 to 9.
  - A digit in base 2 ranges from 0 to 1 (binary number system). A digit in base 2 is also called a ‘bit’.
  - A digit in base R can range from 0 to R-1
  - A digit in Base 16 can range from 0 to 16-1 (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F). Use letters A-F to represent values 10 to 15. Base 16 is also called Hexadecimal or just ‘Hex’.
Common Number Systems (BASES) Used In Computers

<table>
<thead>
<tr>
<th>BINARY (Base 2)</th>
<th>OCTAL (Base 8)</th>
<th>DECIMAL (Base 10)</th>
<th>HEXADECIMAL (Base 16)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 - 000</td>
<td>0 - 0000</td>
<td>A - 1010</td>
</tr>
<tr>
<td>1</td>
<td>1 - 001</td>
<td>1 - 0001</td>
<td>B - 1011</td>
</tr>
<tr>
<td>2 - 010</td>
<td>2 - 0010</td>
<td>2 - 0010</td>
<td>C - 1100</td>
</tr>
<tr>
<td>3 - 011</td>
<td>3 - 0011</td>
<td>3 - 0011</td>
<td>D - 1101</td>
</tr>
<tr>
<td>4 - 100</td>
<td>4 - 0100</td>
<td>4 - 0100</td>
<td>E - 1110</td>
</tr>
<tr>
<td>5 - 101</td>
<td>5 - 0101</td>
<td>5 - 0101</td>
<td>F - 1111</td>
</tr>
<tr>
<td>6 - 110</td>
<td>6 - 0110</td>
<td>6 - 0110</td>
<td></td>
</tr>
<tr>
<td>7 - 111</td>
<td>7 - 0111</td>
<td>7 - 0111</td>
<td></td>
</tr>
<tr>
<td>8 - 1000</td>
<td>8 - 1000</td>
<td>8 - 1000</td>
<td></td>
</tr>
<tr>
<td>9 - 1001</td>
<td>9 - 1001</td>
<td>9 - 1001</td>
<td></td>
</tr>
</tbody>
</table>

Positional Notation

- Value of number is determined by multiplying each digit by a weight and then summing
- The weight of each digit is a POWER of the BASE and is determined by position

- \(953.78 = 9 \times 10^2 + 5 \times 10^1 + 3 \times 10^0 + 7 \times 10^{-1} + 8 \times 10^{-2}\)
  \[= 900 + 50 + 3 + .7 + .08 = 953.78\]

- \(0b1011.11 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2}\)
  \[= 8 + 0 + 2 + 1 + 0.5 + 0.25 = 11.75\]

- \(0xA2F = 10 \times 16^2 + 2 \times 16^1 + 15 \times 16^0\)
  \[= 10 \times 256 + 2 \times 16 + 15 \times 1\]
  \[= 2560 + 32 + 15 = 2607\]
**Base Conversion**

- **By expansion**
  - For converting base $x$ to base 10

- example:
  
  \[
  (1011011)_2 \quad = \\
  1\times2^7+0\times2^6+1\times2^5+1\times2^4+1\times2^3+0\times2^2+1\times2^1+1\times2^0 \\
  = 128+0+32+16+8+0+2+1 = (187)_{10}
  \]

- example:
  
  \[
  (51A)_{16} = 5\times16^2+1\times16^1+10\times16^0 \\
  (51Ah) = (1306)_{10}
  \]

**Base Conversion**

- **Conversion of base 10 to base X**
  - Use a successive division approach

Convert from base=10 to base=2

\[
\begin{array}{c|c|c}
2 \hspace{5mm} 38 & 0 & \text{Remainder} \\
2 \hspace{5mm} 19 & 1 & \\
2 \hspace{5mm} 9 & 1 & \\
2 \hspace{5mm} 4 & 0 & \\
2 \hspace{5mm} 2 & 0 & \\
2 \hspace{5mm} 1 & 1 & \\
0 & & \\
\end{array}
\]

\[
(100110)_2 = (38)_{10}
\]

Convert from base=10 to base=16

\[
\begin{array}{c|c|c}
16 \hspace{5mm} 587 & 11 & \text{Remainder} \\
16 \hspace{5mm} 36 & 4 & \\
16 \hspace{5mm} 2 & 2 & \\
\hspace{5mm} 0 & & \\
\end{array}
\]

\[
(24B)_{16} = (587)_{10}
\]

**Conversion of base 10 to base 16**

\[
(10011011)_2 = (187)_{10}
\]
Conversion Of Binary To Hexadecimal Or Octal

- Group binary digits into groups of four and assign each group a hexadecimal digit

<table>
<thead>
<tr>
<th>0110</th>
<th>1011</th>
<th>0111</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>B</td>
<td>7</td>
</tr>
</tbody>
</table>

- Binary-to-octal:

<table>
<thead>
<tr>
<th>011</th>
<th>010</th>
<th>110</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

- Hexadecimal-to-binary:

<table>
<thead>
<tr>
<th>A</th>
<th>1</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>110</td>
<td>001</td>
<td>1001</td>
</tr>
</tbody>
</table>

- Octal-to-binary:

<table>
<thead>
<tr>
<th>5</th>
<th>0</th>
<th>3</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>000</td>
<td>011</td>
<td>001</td>
</tr>
</tbody>
</table>

Negative Integer Representation

- Sign-magnitude format
  - Attach an extra bit to indicate the sign of the number (0=+, 1=−)
  - example:

\[(01011)_2 = (11)_{10} \quad (11011)_2 = -(11)_{10}\]
Negative Integer Representation

• 2's complement format
  – Negative numbers are assigned as follows:
    \[-b = 2^n - b\]
  – where \( b \) is the magnitude and \( n \) is the number of bits representing the number

• example: \( n=16 \) \( b=1234_{10} \)
  
  \[1234_{10} = (0000010011010010)_2 = (04D2)_{16}\]

  \(-b \) (in 2's complement)
  
  \[2^{16} - 1234_{10} = (1111101100101110)_2 = (FB2E)_{16}\]


Negative Integer Representation

• In general:
  – If \( n \) bits are used in the 2's complement number system, then from \(-2^{n-1}\) to \(2^{n-1}-1\) can be represented.

  example: \( n=16 \) integers from \(-2^{16-1}\) to \(2^{16-1}-1\) \(= -32768\) to \(32767\)

• Forming 2's complement numbers
• 2 methods
  
  – Exchange all 1's and 0's and add 1
    • example: \( b=0101 \) \((2^{4-1})-b = 1010 \) (1's complement)+1 \(= 1011 \) (2's complement)
  
  – Exchange all 1's and 0's after the first 1
**Binary Coded Decimal (BCD) Format**

- Numbers are stored in terms in their 4-bit binary equivalent.
  - 2 basic BCD formats
    - Packed BCD - a string of decimal digits are stored in a sequence of 4-bit groups.
      - example: \( 9502_{10} \) would be stored as:
        
        \[
        1001 \ 0101 \ 0000 \ 0010
        \]
    - Unpacked BCD - digits are stored in the low-order half of an 8-bit group (what is in the high half is undefined - usually zero)
      - example: \( 9502_{10} \) would be stored as:
        
        \[
        uuuu1001 \ uuuu0101 \ uuuu0000 \ uuuu0010
        \]

**Codes for Characters**

- Also need to represent Characters as digital data.
- The ASCII code (American Standard Code for Information Interchange) is a 7-bit code for Character data.
  - Typically 8 bits are actually used with the 8th bit being zero or used for error detection (parity checking).
    - ‘A’ \( = \%01000001 = 0x41 \)
    - ‘&’ \( = \%00100010 = 0x26 \)
### ASCII: American Standard Code for Information Interchange

#### ASCII TABLE

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hex</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>NUL</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>SOH</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>STX</td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td>ETX</td>
</tr>
<tr>
<td>4</td>
<td>04</td>
<td>EOT</td>
</tr>
<tr>
<td>5</td>
<td>05</td>
<td>ENQ</td>
</tr>
<tr>
<td>6</td>
<td>06</td>
<td>ACK</td>
</tr>
<tr>
<td>7</td>
<td>07</td>
<td>BEL</td>
</tr>
<tr>
<td>8</td>
<td>08</td>
<td>BS</td>
</tr>
<tr>
<td>9</td>
<td>09</td>
<td>HT</td>
</tr>
<tr>
<td>10</td>
<td>0A</td>
<td>LF</td>
</tr>
<tr>
<td>11</td>
<td>0B</td>
<td>VT</td>
</tr>
<tr>
<td>12</td>
<td>0C</td>
<td>FF</td>
</tr>
<tr>
<td>13</td>
<td>0D</td>
<td>CR</td>
</tr>
<tr>
<td>14</td>
<td>0E</td>
<td>SO</td>
</tr>
<tr>
<td>15</td>
<td>0F</td>
<td>SI</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>DLE</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>DC1</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>DC2</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>DC3</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>DC4</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>NA</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>-space</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>DEL</td>
</tr>
</tbody>
</table>

https://commons.wikimedia.org/w/index.php?curid=10388973

### Binary Arithmetic

- Recall that N binary digits (N bits) can represent unsigned integers from 0 to \(2^N - 1\)
  - 4 bits = 0 to 15
  - 8 bits = 0 to 255
  - 16 bits = 0 to 65535
- Besides simply representation, we would like to also do arithmetic operations on numbers in binary form
- Principle operations are addition and subtraction
# Binary Addition and Subtraction

The rules for binary addition are:

- $0 + 0 = 0$, carry = 0
- $1 + 0 = 1$, carry = 0
- $0 + 1 = 1$, carry = 0
- $1 + 1 = 0$, carry = 1

The rules for binary subtraction are:

- $0 - 0 = 0$, borrow = 0
- $1 - 0 = 1$, borrow = 0
- $0 - 1 = 1$, borrow = 1
- $1 - 1 = 0$, borrow = 0

Borrows, Carries from/to digits to left of current of digit.

Binary subtraction, addition works just the same as decimal addition, subtraction.

---

# Binary, Decimal addition

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>$0b\ 101011$</td>
</tr>
<tr>
<td>+ 17</td>
<td>+ $0b\ 00001$</td>
</tr>
<tr>
<td>-------</td>
<td>---------------</td>
</tr>
<tr>
<td>51</td>
<td>$101100$</td>
</tr>
</tbody>
</table>

From LSB to MSB:
- $1 + 1 = 0$, carry of 1
- $1 + 0 = 1$
- $0 + 0 = 0$
- $1 + 0 = 1$

From LSD to MSD:
- $7 + 4 = 1$; with carry out of 1 to next column
- $1 (carry) + 1 + 0 = 0$, carry of 1
- $1 (carry) + 0 + 0 = 1$, no carry
- $1 + 0 = 1$

$1 (carry) + 3 + 1 = 5$.  
Answer = 51.
### Subtraction

**Decimal**

\[
\begin{array}{c}
900 \\
- 001 \\
\hline
899
\end{array}
\]

0-1 = 9; with borrow of 1 from next column
0 -1 (borrow) - 0 = 9, with borrow of 1
9 - 1 (borrow) - 0 = 8.
Answer = 899.

**Binary**

\[
\begin{array}{c}
0b 100 \\
- 0b 001 \\
\hline
011
\end{array}
\]

0-1 = 1; with borrow of 1 from next column
0 -1 (borrow) - 0 = 1, with borrow of 1
1 - 1 (borrow) - 0 = 0.
Answer = %011.

### Hex Addition

\[
\begin{array}{c}
0x3A \\
+ 
0x28 \\
\hline
0x62
\end{array}
\]

A+8 = 2; with carry out of 1 to next column
1 (carry) + 3 + 2 = 6.
answer = 0x62

**Decimal check.**

\[
\begin{align*}
0x3A &= 3 \times 16 + 10 \\
&= 58 \\
0x28 &= 2 \times 16 + 8 \\
&= 40 \\
58 + 40 &= 98 \\
0x62 &= 6 \times 16 + 2 \\
&= 96 + 2 = 98!!
\end{align*}
\]
**Hex addition again**

Why is $0xA + 0x8 = 2$ with a carry out of 1?

The carry out has a weight equal to the BASE (in this case 16). The digit that gets left is the excess.

$$Ah + 8h = 10 + 8 = 18.$$ 

18 is GREATER than 16 (BASE), so need a carry out!

Excess is $18 - BASE = 18 - 16 = 2$, so ‘2’ is digit.

Exactly the same thing happens in Decimal.

$$5 + 7 = 2, \text{ carry of } 1.$$ 

$$5 + 7 = 12, \text{ this is greater than } 10!.$$ 

So excess is $12 - 10 = 2, \text{ carry of } 1.$

---

**Hex Subtraction**

0x34
- 0x27
---
0x0D

4 - 7 = D; with borrow of 1 from next column

3 - 1 (borrow) - 2 = 0.

answer = 0x0D.

Decimal check.

$$0x34 = 3 \times 16 + 4 = 52$$ 
$$0x27 = 2 \times 16 + 7 = 39$$ 
$$52 - 39 = 13$$ 
$$0x0D = 13 !!$$
Hex subtraction again

Why is 0x4 – 0x7 = 0xD with a borrow of 1?

The borrow has a weight equal to the BASE (in this case 16).

BORROW +0x4 – 0x7 = 16 + 4 - 7 = 20 - 7 = 13 = 0xD.

0xD is the result of the subtraction with the borrow.

Exactly the same thing happens in decimal.
3 - 8 = 5 with borrow of 1
borrow + 3 - 8 = 10 + 3 - 8 = 13 - 8 = 5.

Unsigned Overflow

• In this class we use 8 bit or 16 bit precision most of the time.
• Overflow occurs when I add or subtract two numbers, and the correct result is a number that is outside of the range of allowable numbers for that precision.
  – I can have both unsigned and signed overflow (more on signed numbers later)
• 8 bits -- unsigned integers
  – 0 to 2^8 -1 or 0 to 255
• 16 bits -- unsigned integers
  – 0 to 2^16 -1 or 0 to 65535
Unsigned Overflow Example

Assume 8 bit precision; i.e. I can’t store any more than 8 bits for each number.

Let’s add $255 + 1 = 256$. The number 256 is OUTSIDE the range of 0 to 255! What happens during the addition?

$$255 = \text{0xFF}$$
$$+ 1 = \text{0x01}$$
__________________________
$$256 /= \text{0x00}$$ /= means Not Equal

$0xF + 1 = 0$, carry out
$0xF + 1 \text{ (carry)} + 0 = 0$, carry out
Carry out of MSB falls off end, No place to put it!!!
Final answer is WRONG because could not store carry out.

Unsigned Overflow

• A carry out of the Most Significant Digit (MSD) or Most Significant Bit (MSB) is an OVERFLOW indicator for addition of UNSIGNED numbers.
  – The correct result has overflowed the number range for that precision, and thus the result is incorrect.

• If we could STORE the carry out of the MSD, then the answer would be correct.
  – But we are assuming it is discarded because of fixed precision, so the bits we have left are the incorrect answer.
Logic Circuits

- Logic circuits perform operations on digital signals
  - Implemented as electronic circuits where signal values are restricted to a few discrete values
- In binary logic circuits there are only two values, 0 and 1
- The general form of a logic circuit is a switching network

Boolean Algebra

- Direct application to switching networks
  - Work with 2-state devices \( \rightarrow \) 2-valued Boolean algebra (switching algebra)
  - Use a Boolean variable (X, Y, etc.) to represent an input or output of a switching network
  - Variable may take on only two values (0, 1)
  - \( X=0, \ X=1 \)
  - These symbols are not binary numbers, they simply represent the 2 states of a Boolean variable
  - They are not voltage levels, although they commonly refer to the low or high voltage input/output of some circuit element
Representing ‘1’ and ‘0’

- In the electrical world, two ways of representing ‘0’ and ‘1’ are (these are not the only ways):
  - Presence or absence of electrical current
  - Different Voltage levels
- Different voltage levels are the most common
  - Usually 0v for logic ‘0’, some non-zero voltage for logic ‘1’ (i.e. > 3 volts)
- Can interface external sources to digital systems in many ways
  - Switches, buttons, other human controlled input devices
  - Transducers (change a physical quantity like temperature into a digital quantity).

Switch Inputs

High True switch

Vdd

L

Gnd

Switch open (negated), output is L

Vdd is power supply voltage, typically 5V or 3.3V

Gnd is 0 V

Switch closed (asserted), output is H
Examples of high, low signals

Switch open (negated), output is H  
Switch closed (asserted), output is L

CMOS transistors (P, N)

S: source  
G: gate  
D: drain  

transistor operation of P, N types is complementary to each other
Inverter gate - takes 2 transistors

VDD (logic 1)

A

Y = ~A

PMOS is open (off)

PMOS is closed (on)

A = 1

Y = 0

A = 0

Y = 1

NMOS is Closed (on)

NMOS is Open (off)

Truth tables

• Tabular listing that fully describes a logic function
  – Output value for all input combinations (valuations)

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_1' \cdot x_2$</th>
<th>$x_1 \cdot x_2$</th>
<th>$x_1' + x_2$</th>
<th>$x_1 \cdot x_1'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

AND

OR

NOT
Logic gates and networks

- Each basic logic operation (AND, OR, NOT) can be implemented resulting in a circuit element called a logic gate.
- A logic gate has one or more inputs and one output that is a function of its inputs.

\[ x_1 \cdot x_2 \]

AND gates

\[ x_1 + x_2 + \ldots + x_n \]

OR gates

\[ \overline{x} \]

NOT gate
Logic gates and networks

- A larger circuit is implemented by a network of gates
  - Called a logic network or logic circuit

\[ f = (x_1 + x_2) \cdot x_3 \]

Analysis of a logic network

- The function of a logic network can be described by a timing diagram (gives dynamic behavior of the network)
**Tri-State Buffer**

A common way to drive a line or bus from multiple sources is to use a TRISTATE buffer.

When $EN = 1$, then $Y = A$.

When $EN = 0$, then $Y = \text{undriven}$.

$Y$ is undriven, this is called the high impedance state.

Designate high impedance by a 'Z'.

When $EN = 0$, then $Y = 'Z'$ (high impedance)

---

**Using Tri-State Buffers (cont)**

Only $A$ or $B$ is enabled at a time.

If $S=0$ then $Y = A$
If $S=1$ then $Y = B$
Combinational Building Blocks, Mux

Binary Adder

F (A,B,C) = A xor B xor C      G = AB + AC + BC

These equations look familiar. These define a
Binary Full Adder:

Sum = A xor B xor Cin
Cout = AB + Cin A + Cin B
      = AB + Cin (A + B)

Full Adder (FA)
4-Bit Ripple Carry Adder

Incrementer
**Combinational Right Shifter**

A combinational block that can either shift right or pass data unchanged

![Diagram of 4-bit Right Shifter](Image)

Copyright 2005. Thomson/Delmar Learning, All rights reserved.

**Understanding the shift operation**

<table>
<thead>
<tr>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x85 = 0x1011</td>
<td>0x1011</td>
</tr>
<tr>
<td>SI = 0</td>
<td>1st right shift</td>
</tr>
<tr>
<td>0x42 = 0x1001</td>
<td>0x1001</td>
</tr>
<tr>
<td>SI = 0</td>
<td>2nd right shift</td>
</tr>
<tr>
<td>0x21 = 0x0101</td>
<td>0x0101</td>
</tr>
<tr>
<td>SI = 0</td>
<td>3rd right shift</td>
</tr>
<tr>
<td>0x10 = 0x0100</td>
<td>0x0100</td>
</tr>
</tbody>
</table>

Etc....
Clock Signal Review

- τ - period (in seconds)
- $P_w$ - pulse width (in seconds)
- $f$ - frequency pulse width (in Hertz)  \( f = \frac{1}{\tau} \)
- Duty cycle - ratio of pulse width to period (in %)  \( \text{duty cycle} = \frac{P_w}{\tau} \)

<table>
<thead>
<tr>
<th>Unit</th>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>millisecond (ms)</td>
<td></td>
<td>$10^{-3}$</td>
</tr>
<tr>
<td>Kilo Hz (KHz)</td>
<td></td>
<td>$10^3$</td>
</tr>
<tr>
<td>Micro sec (µs)</td>
<td></td>
<td>$10^{-6}$</td>
</tr>
<tr>
<td>Mega Hz (MHz)</td>
<td></td>
<td>$10^6$</td>
</tr>
<tr>
<td>Nano sec (ns)</td>
<td></td>
<td>$10^{-9}$</td>
</tr>
<tr>
<td>Giga Hz (GHz)</td>
<td></td>
<td>$10^9$</td>
</tr>
</tbody>
</table>

Storage Element: The D Flip-Flop

- D: data input
- CK: clock input
- S: set input (asynchronous, low true)
- R: reset input (asynchronous, low true)
**Synchronous vs. Asynchronous Inputs**

Synchronous input: Output will change after active clock edge  
Asynchronous input: Output changes independent of clock  

![Synchronous vs. Asynchronous Inputs Diagram](image)

State elements often have async set, reset control.  
D input is synchronous with respect to Clk  
S, R are asynchronous. Q output affected by  
S, R independent of C. Async inputs are  
dominant over Clk.

---

**Registers**

The most common sequential building block is the register. A register is \( N \) bits wide and has a load line for loading in a new value into the register.

![Registers Diagram](image)

Note that DFF simply loads old value when \( LD = 0 \). DFF is loaded  
every clock cycle.
Counter

Very useful sequential building block. Used to generate memory addresses, or keep track of the number of times a datapath operation is performed.

Shift Register

Very useful sequential building block. Used to perform either parallel to serial data conversion or serial to parallel data conversion.
What You Need to Know

- Convert hex, binary integers to decimal
- Convert decimal integers to hex, binary
- Convert hex to binary, binary to hex
- Formats/codes and number/character representation
- Addition, subtraction of binary, hex numbers
- Detecting unsigned overflow
- Basic two-input Logic Gate operation
- NMOS/PMOS Transistor Operations
- Inverter/NAND transistor configurations
- Tri-state buffer operation
- Mux, Adder operation
- Clock signal definition
- DFF, Register, Counter, Shifter register operation