Unit 1: Introduction to Number Systems and Conversions

1.1 Digital Systems and Switching Circuits

Digital vs. Analog

Digital System: *physical quantities or signals can assume only discreet values, in many cases for a given input, the output is exactly correct. 0V or 5V*

Analog System: *physical quantities or signals may vary continuously over a specified range. +10 to -10*

Demo: Light Switch w/ 2 definitive states ON / OFF
Temperature of room -or- Measuring with ruler

Digital System Design – 3 parts

System Design: *breaking overall system to subsystems and specifying the characteristics of each subsystem.*
i.e. Computer: memory, cards/ I/O, etc.

Logic Design: *How to interconnect basic logic building blocks to perform a specific function.*
i.e. Connecting flip flops & gates for binary arithmetic

Circuit Design: *Specifying the interconnection of specific components to form a gate, flip flop or other logic building block.*
i.e. Resistors, diode, transistors to create gates.

Switching Circuits: *a circuit that has one or more inputs and one or more outputs that take on discrete values.*

Combinational Circuit: *a switching circuit where the output values depend only on the present values of the inputs and not on past values*.

Sequential Circuit: *a switching circuit where the output values depend on both the present and past input values. Sequential circuit are said to have "memory"*

Generally a sequential circuit is a combinational circuit with added memory.

Logic Gates: *basic building blocks of combinational circuits*

Boolean Algebra: *basic mathematics used for study of logic design, mathematically depicts relationship between inputs and outputs.*
Will be discussed in Units 3 &4

Design:
- Defining – Unit 4
- Simplification – Units 3, 5 and 6
- Implementation – Unit 7
  - Programmable Logic Devices – Unit 9

Two State Devices: *switching devices used in digital system where the output can assume only two different discreet values.*

Relay: *closed or open depending on if power to coil or not*

Diode: *conducting or nonconducting*

Transistor: *cut-off or saturated state w/ high or low output.*

Since most switching devices assume only two states, binary is the natural choice for digital systems.

1.2 Number Systems and Conversions

Number Systems

Positional Notation: *expressing a number where each digit is multiplied by a power of its base depending on its position in the number*.

Ex. Decimal (Base 10)

375.25_{10}

Similarly,

Ex. Binary (Base 2)

1101.11_2

Note: *In both the base 2 and base 10 examples the decimal point separates the positive and negative powers*

General Case: Any positive integer R (R>1) can be the *base* of a number system; containing R digits (0, 1, 2, …, R-1)

Ex. Base 8

Any number in positional notation can be written as a power series…
For a number N, where

$$N = (a_4\ a_3\ a_2\ a_1\ a_0\ .a_{-1}\ a_{-2}\ a_{-3})_R$$
$$= a_4\ X\ R^4 + a_3\ X\ R^3 + a_2\ X\ R^2 + a_1\ X\ R^1 + a_0\ X\ R^0 +$$
$$+ a_{-1}\ X\ R^{-1} + a_{-2}\ X\ R^{-2} + a_{-3}\ X\ R^{-3}$$

Where a_i is the integer coefficient of R^i and $0 \leq a_i \leq R\text{-}1$

Ex. 167.5_8

Number systems with bases > 10, letters usually represent digits above 9.
In hexadecimal (Base 16) letters A – F are used for digits 10 – 15

$A=10_{10}$ $B=11_{10}$ $C=12_{10}$ $D=13_{10}$ $E=14_{10}$ $F=15_{10}$; for digits R-1 = 15

Ex. $1AF_{16}$

Conversions

Division Method: *The base R equivalent of a decimal integer N, can be represented:*

$$N = (a_n\ a_{n-1}\ \ldots a_2\ a_1\ a_0)_R = a_n R^n + a_{n-1}R^{n-1} + \ldots + a_2R^2 + a_1R^1 + a_0$$

Dividing N by R

$$N/R = a_nR^{n-1} + a_{n-1}R^{n-2} + \ldots + a_2R^1 + a = Q_1, \text{ remainder } a_0$$

Dividing Q_1 by R

$$Q_1/R = a_nR^{n-2} + a_{n-1}R^{n-3} + \ldots\ a_3R^1 + a_2 = Q_2, \text{ remainder } a_1$$

Dividing Q_2 by R

$$Q_2/R = a_nR^{n-3} + a_{n-1}R^{n-4} + \ldots + a_3 = Q_3, \text{ remainder } a_2$$

Continue until a value for a_n is obtained

Note: *The remainder obtained at each step is a desired digit with the least significant obtained first.*

Ex. 53_{10} to Binary

Multiplication Method: *A decimal fraction F to base R equivalent can be represented*

$$F = (.a_{-1}\ a_{-2}\ a_{-3}\ \ldots\ a_{-m})_R = a_{-1}R^{-1} + a_{-2}R^{-2} + a_{-3}R^{-3} + \ldots + a_{-m}R^{-m}$$

Multiplying F by R

$$FR = a_{-1} + a_{-2}R^{-1} + a_{-3}R^{-2} + \ldots + a_{-m}R^{-m+1} = a_{-1} + F_1$$

Where F_1 represents the fractional part and a_{-1} is the integer

Multiplying F_1 by R

$$F_1R = a_{-2} + a_{-3}R^{-1} + \ldots + a_{-m}R^{-m+2} = a_{-2} + F_2$$

Multiplying F_2 by R

$$F_2R = a_{-3} + \ldots + a_{-m}R^{-m+3} = a_{-3} + F_3$$

Continue until there are a significant number of digits

Note: *The integer value of each step is the desired digit with the most significant obtained first.*

Ex. 0.625_{10} to Binary

The result may not terminate, indicating a repeating fraction

Ex. 0.7_{10} to Binary

The division and multiplication methods can be used to convert between any two bases other than decimal but the operations must be done in bases other than 10.Generally it is easier to convert to decimal first.

Ex. 231.3_4 to base 7

Binary to Hex / Hex to Binary: *Each Hex digit corresponds to exactly 4 binary digits and conversion can be done by inspection. Group in fours from each side of the decimal point. Add zeroes when necessary for the 4-bit groups*

Ex. 1001101.010111_2

1.3 Binary Arithmetic

Addition Tables: *Binary addition same as decimal only easier*

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0 and carry 1 to next column

Ex. $13_{10} + 11_{10}$ in binary

Subtraction Tables:

0 - 0 = 0
0 - 1 = 1 and borrow 1from the next column
1 - 0 = 1
1 - 1 = 0

Borrowing 1 from a column is the same as subtracting 1 from that column

Ex. a) $\overset{1}{}11101$	b) $\overset{1\,1\,1\,1}{}10000$	c) $\overset{1\,1\,1}{}111001$
$-\underline{10011}$	$-\underline{\quad 11}$	$-\underline{\quad 1011}$
1010	1101	101110

Note: *Borrowing propagates from column to column as needed*

Borrowing Review (decimal): *When borrowing 1 from a given column n, it means subtracting 1 from n and adding 10 (base) to the column to its right.*
Proof: $1 \times 10^n = 10 \times 10^{n-1}$

Solving the problem

$$\begin{array}{r} 205 \\ -\ \underline{18} \\ 187 \end{array}$$

Via power series expansion:

$$205 - 18 = [2 \times 10^2 + 0 \times 10^1 + 5 \times 10^0] - [1 \times 10^1 + 8 \times 10^0]$$

$$= [2 \times 10^2 + (0-1) \times 10^1 + (10+5) \times 10^0] - [1 \times 10^1 + 8 \times 10^0]$$

$$= [(2-1) \times 10^2 + (10+0-1) \times 10^1 + 15 \times 10^0] - [1 \times 10^1 + 8 \times 10^0]$$

$$= [1 \times 10^2 + 8 \times 10^1 + 7 \times 10^0] = 187$$

Subtraction in binary is performed using borrowing in the same way except using powers of 2 instead of 10

Ex. 111001 (c from above)
$-\ \underline{1011}$

Multiplication Tables:

0 x 0 = 0
0 x 1 = 0
1 x 0 = 0
1 x 1 = 1

Ex. 13_{10} * 11_{10} in binary

1101
1011
1101
11010
000000
1101000
10001111 = 143_{10}

Note: *Each partial product is either the multiplicand (**1101**) shifted the appropriate spaces or zero.*

Partial products method: *Method used to solve multiplication problems where the partial products are added one at a time. This avoids complications for carries >1.*

1111
1101
1111
00000
(011111)
111100
(1001011)
1111000
11000011

Division: *Just like decimal division but easier only possible quotient digits are 0 and 1*

Ex. 145_{10} divided by 11_{10} in binary

1.4 Representation of Negative Numbers

Sign and Magnitude: *method of representation of binary numbers both positive and negative where the first bit is the sign. (0 positive & 1 negative) For an n-bit word the first bit is the sign and the other n-1 bits the magnitude. See Table 1-1.*

2's Complement: *Number system representation in a binary system where a positive number N is represent by 0 followed by its magnitude (like sign & magnitude)however negative numbers –N are represented by its 2's complement N**

For a word of n bits, the 2's complement of a positive integer N is:

$$\mathbf{N^* = 2^n - N}$$

Ex.

Note: *1000 represents -8 in a 4 bit system*

1's Complement: *Same as the 2's complement, except a negative number –N is represented by its 1's complement N.*

For a word of n bits, the 1's complement of a positive integer N is

$$\mathbf{\overline{N} = (2^n - 1) - N}$$

Note: *1111 represents -0 and -8 has no definition in a 4 bit system*

Alternate method: *Equivalent definition: to form the 1's complement, simply complement N bit-by-bit by replacing 1's with 0's and 0's with 1's.*
This is true since $2^n - 1$ is all 1's for any value of n. and subtracting a bit from one is the same as complementing it.

Ex. For n=6 and N=010101

Combining the two equations yields:_

$N^* = 2^n - N = (2^n - 1 - N) + 1 = \overline{N} + 1$

So the 2's complement can be found by complementing N bit-by-bit and adding 1

Rearranging the terms for 1's and 2's complement yields:

$N = 2^n - N^*$ and $N = (2^n - 1) - N$

Therefore you can find the magnitude of a negative integer represented by its 1's or 2's complement by taking the 1's or 2's complement of the 1's or 2's complement

Addition using 2's Complement: *carry out of addition as if all numbers were positive with any carry from the sign position ignored.*

Overflow: *condition where the correct representation of the sum of an n bit word requires more than n bits.*

For n = 4

1) Addition of 2 positive numbers; sum $< 2^{n-1}$

+3	*0011*
+4	*0100*
+7	*0111*

2) Addition of 2 positive numbers; sum $\geq 2^{n-1}$

+5	*0101*
+6	*0110*
+11	*1011 – overflow! +11 requires 5 bits*

3) Addition of positive and negative numbers; (negative of larger magnitude)

+5	*0101*
- 6	*1010*
- 1	*1111*

4) Addition of positive and negative numbers; (positive of larger magnitude)

- 5	*1011*
+6	*0110*
+1	*(1)0001 – correct if carry from sign bit ignored*

5) Addition of 2 negative numbers; $|\text{sum}| \leq 2^{n-1}$

- 3	*1101*
- 4	*1100*
- 7	*(1)1001 – correct if carry from sign bit ignored*

6) Addition of 2 negative numbers; $|\text{sum}| > 2^{n-1}$

- 5	1011
- 6	1010
-11	(1)0101 *– overflow! -11 requires 5 bits*

Note: *The overflow conditions are easy to detect: when adding two positives yields a negative #2 OR adding two negatives yields a positive #6*

Proof – Throw away carry:

Addition of positive and negative numbers; (positive of larger magnitude); Item 4.

$-A + B$ *where* $(B>A)$

$A^* + B = (2^n - A) + B = 2^n + (B - A) > 2^n$

Throwing away the last carrying equals subtracting 2^n *resulting in B-A*

Addition of 2 negative numbers; $|\text{sum}| \leq 2^{n-1}$; Item 5.

$-A - B$ *where* $(A+B \leq 2^{n-1})$

$A^* + B^* = (2^n - A) + (2^n - B) = 2^n + 2^n - (A + B)$

Subtracting 2^n *yields* $2^n - (A + B) = (A + B)^* = -(A + B)$

Addition using 1's Complement: *Similar to 2's complement except instead of discarding the sign bit carry, end around carry is used.*

End-around carry: *taking any sign bit carry and adding it to the n bit sum in the furthest right position*

1) Addition of 2 positive numbers; sum $< 2^{n-1}$

Same as 2's complement

2) Addition of 2 positive numbers; sum $\geq 2^{n-1}$

Same as 2's complement

3) Addition of positive and negative numbers; (negative of larger magnitude)

+5	*0101*
- 6	*1001*
- 1	*1110*

4) Addition of positive and negative numbers; (positive of larger magnitude)

- 5	*1010*
+6	*0110*
+1	*(1)0000*
	1 – end around carry
	0001 – correct, no overflow

5) Addition of 2 negative numbers; $|\text{sum}| \leq 2^{n-1}$

- 3	*1100*
- 4	*1011*
- 7	*(1)0111*
	1 – end around carry
	1000 – correct, no overflow

6) Addition of 2 negative numbers; $|\text{sum}| > 2^{n-1}$

- 5 1010
- 6 1001
-11 (1)0011 – *overflow! -11 requires*
1 – *end around carry*
0100 – overflow!

Note: *Again the overflow condition is easy to detect, adding two negatives yields a positive*

Proof – End-around carry:

Addition of positive and negative numbers; (positive of larger magnitude); Item 4.

$-A + B$ *where* $(B>A)$

$\overline{A} + B = (2^n - 1 - A) + B = 2^n + (B - A) - 1$

End carry is same as subtract 2^n *then add . Results (B-A)*

Addition of 2 negative numbers; $|\text{sum}| \leq 2^{n-1}$; Item 5.

$-A - B$ *where* $(A+B \leq 2^{n-1})$

$\overline{A} + \overline{B} = (2^n - 1 - A) + (2^n - 1 - B) = 2^n + [2^n - 1 - (A + B)] - 1$

After end carry… $2^n - 1 - (A + B) = \overline{(A + B)} = -(A + B)$

1's and 2's Complement for n = 8.

1's) -11 -20

2's) -8 + 19

General Rule: *An overflow occurs if adding two positives results in an negative or two negatives results in a positive.*

1.5 **Binary Codes:** *Since most computers use binary but I/O is usually in decimal, the decimal numbers must be coded into binary signals.*
See Table 1-2

Binary Coded Decimal (BCD): *Replacement of each decimal by its binary equivalent*

Ex. 8-4-2-1 BCD for 937.25

1001 0011 0111 . 0010 0101
9 3 7 . 2 5

Weighted Codes: *4 bit binary code that has weights w_3, w_2, w_1, w_0 and code a_3, a_2, a_1, a_0 and represents a decimal number N*

$$N = w_3a_3 + w_2 a_2 + w_1 a_1 + w_0 a_0$$

Ex. 6311 code

2-out-of-5 Code: *Exactly 2 out of 5 bits are 1 for every valid combination. Useful for error checking because if an error occurs the number of 1's will not be exactly 2.*

Gray Code: *Codes for successive decimal digits differ by exactly one bit, Useful for analog translation since analog change will change only one bit.*

Note: *The decimal value of a digit cannot be computed by a simple formula when non-weighted code is used.*

ASCII Code: *(American Standard Code for Information Interchange) 7 bit alphanumeric to binary code representation often used to transmit alphanumeric data to and from a computer*