Sergio Garcia Tapia
Digital Design and Computer Architecture: RISC-V
January 30, 2024

Exercise 1.4. An analog voltage is in the range of 0-5 V. If it can be measured with an accuracy of ±50 mV, at most how many bits of information does it convey?

Solution: According to *Section 1.3: Digital Abstraction*, the *amount of information* D in a discrete valued varaible with N distinct states is measured in units of *bits* as

$$D = \log_2 N$$

There are 6 digits in the 1s place (0 through 5), 10 digits in the first decimal (0 through 10), and 6 digits in the second decimal place (0 through 5). Overall, that is $6 \cdot 10 \cdot 6$ states, or $N = 360$. Therefore, at most $D = \log_2 360 = 8.49$ bits of information are conveyed.

Exercise 1.5. A classroom has an old clock on the wall whose minute hand broke off.

(a) If you can read the hour hand to the nearest 15 minutes, how many bits of information does the clock convey about the time?

(b) If you know whether it is before or after noon, how many additional bits of information do you know about the time?

Solution:

(a) Since an hour has 60 minutes, the described conditions means there are 4 distinguishable states per hour. Since a wall-clock is only able to distinguish between 12 hours in a day (not 24), there are $12 \cdot 4 = 48$ total states. Therefore, the amount of information D is $D = \log_2 48 = 5.58$ bits of information.

(b) If we know that it is before noon or after noon, we have one additional bit of information. If we were to do the math, note that we would have double the states, so now we would have

$$D = \log_2 48 \cdot 2 = \log_2 48 + \log_2 2 = \log_2 48 + 1$$

Exercise 1.6. The Babylonians developed the *sexagesimal* (base 60) number system about 4000 years ago. How many bits of information is conveyed with one sexagesimal digit? How do you write the number the number 4000_{10} in sexagesimal?

Solution: Since there are 60 values for a sexagesimal digit, a total of $D = \log_2 60 = 5.91$ bits of information are conveyed. To write 4000_{10} in sexagesimal, we repeatedly divide by 60 and assign the remainder to the current bit, starting from the right.

$$\begin{aligned} 4000 &= 60 \cdot 66 + 40 \\ 66 &= 60 \cdot 1 + 6 \end{aligned}$$

Hence,

$$\begin{aligned} 4000_{10} &= 60 \cdot (60 \cdot 1 + 6) + 40 \\ &= 60^2 \cdot 1 + 60 \cdot 6 + 60^0 \cdot 40 \end{aligned} \tag{1}$$

Therefore, starting from the right, the first digit is 40, followed by 6, followed by 1.

Exercise 1.7. How many different numbers can be represented with 16 bits?

Solution: Each bit has two possible values: 0 or 1. Therefore, with 6 bits, we have 2^{16} different numbers, which is 65,536 possible values.

Exercise 1.8. What is the largest unsigned 32-bit binary number?

Solution: If unsigned, the largest 32-bit binary number is represented by all 1s. This is precisely $2^{32} - 1 = 4,294,967,295$, or about 4.3 billion.

Exercise 1.9. What is the largest 16-bit binary number that can be represented with:

(a) unsigned numbers?

(b) two's complement numbers?

(c) sign/magnitude numbers?

Solution:

(a) $2^{16} - 1 = 65,535$

(b) $2^{15} - 1 = 32,767$

(c) $2^{15} - 1 = 32,767$

Exercise 1.10. What is the largest 32-bit number that can be represented with

(a) unsigned numbers?

(b) two's complement numbers?

(c) sign/magnitude numbers?

Solution:

(a) $2^{32} - 1 = 4,294,967,295$, or about 4.3 billion

(b) $2^{31} - 1 = 2,147,483,647$, or about 2.1 billion

(c) $2^{31} - 1 = 2,147,483,647$, or about 2.1 billion

Exercise 1.11. What is the smallest (most negative) 16-bit number that can be represented with

(a) unsigned numbers?

(b) two's complement numbers?

(c) sign/magnitude numbers?

Solution:

(a) 0 is the smallest; negative numbers are not represented in this system.

(b) $-2^{15} = -32,768$

(c) $-2^{15} + 1 = -32,767$

Exercise 1.12. What is the smallest (most negative) 32-bit number that can be represented with

(a) unsigned numbers?

(b) two's complement numbers?

(c) sign/magnitude numbers?

Solution:

(a) 0 is the smallest; negative numbers are not represented in this system.

(b) $-2^{31} = -1 = 2,147,483,648$

(c) $-2^{31} + 1 = -1 = 2,147,483,647$

Exercise 1.13. Convert the following unsigned binary numbers to decimal. Show your work.

(a) 1010_2

(b) 110110_2

(c) 11110000_2

(d) 000100010100111_2

Solution:

(a) $1010_2 = 2^3 \cdot 1 + 2^2 \cdot 0 + 2^1 \cdot 1 + 2^0 \cdot 0 = 8_{10} + 2_{10} = 10_{10}$

(b) $110110_2 = 2^5 + 2^4 + 2^2 + 2^1 = 32 + 16 + 4 + 2 = 54_{10}$

(c) $11110000_2 = 2^7 + 2^6 + 2^5 + 2^4 = 128 + 64 + 32 + 16 = 240_{10}$

(d) $000100010100111_2 = 2^{11} + 2^7 + 2^5 + 2^2 + 2^1 + 2^0 = 2215_{10}$

Exercise 1.14. Convert the following unsigned binary numbers to decimal. Show your work.

(a) 1110_2

(b) 100100_2

(c) 11010111_2

(d) 011101010100100_2

Solution:

(a) $1110_2 = 2^3 + 2^2 + 2^1 = 8 + 4 + 2 = 14_{10}$

(b) $100100_2 = 2^5 + 2^2 = 32 + 4 = 36_{10}$

(c) $11010111_2 = 2^7 + 2^6 + 2^4 + 2^2 + 2^1 + 2^0 = 215_{10}$

(d) $011101010100100_2 = 2^{13} + 2^{12} + 2^{11} + 2^9 + 2^7 + 2^5 + 2^2 = 15,012_{10}$

Exercise 1.15. Repeat Exercise 1.13, but convert to hexadecimal.

Solution:

(a) $1010_2 = \mathrm{A}_{16}$

(b) $110110_2 = (11_2)(0110_2) = 36_{16}$

(c) $11110000_2 = (1111)_2(0000)_2 = \mathrm{F0}_{16}$

(d) $0001000101001111_2 = (000)_2(1000)_2(1010)_2(0111)_2 = (8\mathrm{A}7)_{16}$

Exercise 1.16. Repeat Exercise 1.14, but convert to hexadecimal.

Solution:

(a) $1110_2 = \mathrm{E}_{16}$

(b) $100100_2 = (10)_2(0100)_2 = 24_{16}$

(c) $11010111_2 = (1101)_2(0111)_2 = \mathrm{D}7_{16}$

(d) $011101010100100_2 = (011)_2(1010)_2(1010)_2(0100)_2 = 3\mathrm{AA}4_{16}$

Exercise 1.17. Convert the following hexadecimal numbers to decimal. Show your work.

(a) $\mathrm{A}5_{16}$

(b) $3\mathrm{B}_{16}$

(c) FFFF_{16}

(d) $\mathrm{D}000000_{16}$

Solution:

(a) $\mathrm{A}5_{16} = 16^1 \cdot 10 + 16^0 \cdot 5 = 160 + 5 = 165_{10}$

(b) $3\mathrm{B}_{16} = 16^1 \cdot 3 + 16^0 \cdot 11 = 48 + 11 = 59_{10}$

(c) $\mathrm{FFFF}_{16} = 16^3 \cdot 15 + 16^2 \cdot 15 + 16^1 \cdot 15 + 16^0 \cdot 15 = 235,929,855_{10}$

(d) $\text{D0000000}_{16} = 16^7 \cdot 13 = 3,489,660,928_{10}$

Exercise 1.18. Convert the following hexadecimal numbers to decimal. Show your work.

(a) 4E_{16}

(b) 7C_{16}

(c) ED3A_{16}

(d) 403FB001_{16}

Solution:

(a) $\text{4E}_{16} = 16^1 \cdot 4 + 16^0 \cdot 14 = 78_{10}$

(b) $\text{7C}_{16} = 16^1 \cdot 7 + 16^0 \cdot 12 = 124_{10}$

(c) $\text{ED3A}_{16} = 16^3 \cdot 14 + 16^2 \cdot 13 + 16^1 \cdot 3 + 16^0 \cdot 10 = 60,730_{10}$

(d) $\text{403FB001}_{16} = 16^7 \cdot 4 + 16^5 \cdot 3 + 16^4 \cdot 15 + 16^3 \cdot 11 + 16^0 \cdot 1 = 1,077,915,664_{16}$

Exercise 1.19. Repeat exercise 1.17, but convert to unsigned binary.

Solution:

(a) A5_{16}: Since $A_{16} = 1010_2$ and $5_{16} = 0101_2$, we concatenate them to get 10100101_2.

(b) 3B_{16}: Concatenate 0011_2 and 1011_2 to get $00111011_2 = 111011_2$.

(c) $\text{FFFF}_{16} = 1111111111111111_2$

(d) $\text{D000000}_{16} = 11010000000000000000000000_2$

Exercise 1.20. Repeat exercise 1.18, but convert to unsigned binary.

Solution:

(a) $\text{4E}_{16} = 01001110_2$

(b) $\text{7C}_{16} = 01111100_2$

(c) ED3A_{16}: Concatenate $\text{E}_{16} = 1110_2$, $\text{D}_{16} = 1101_2$, $3_{16} = 0011_2$ and $\text{A}_{16} = 1010_2$.

(d) 403FB001_{16}: Concatenate $4_{16} = 0100_2$, $0_{16} = 0000_2$, $3_{16} = 0011_2$, $\text{F}_{16} = 1111_2$, $\text{B}_{16} = 1011_2$, $0_{16} = 0000$, $0_{16} = 0000$, and $1_{16} = 0001_2$:

$$01000000001111111010000000000001_2$$

Exercise 1.21. Convert the following two's complement binary numbers to decimal.

(a) 1010_2

(b) 110110_2

(c) 01110000_2

(d) 10011111_2

Solution:

(a) 1010_2: Since this is a two's complement binary representation, the most significant bit represents the sign. Since the most significant bit is a 1, this is a negative number. To find the magnitude, we compute its two's complement by inverting its bits and adding 1:

$$1010 \to 0101_2 + 1_2 \to 0110_2 = 6_{10}$$

Hence, the decimal equivalent is -6_{10}.

(b) 110110_2: This is also negative. Its two's complement is:

$$110110_2 \to 001001_2 + 1_2 \to 001010_2 = 2^3 + 2^1 = 10_{10}$$

The decimal equivalent is thus -10_{10}.

(c) 01110000_2: This time, the most significant bit is 0, so this number is positive. We do not have to find the two's complement this time, and obtain:

$$2^6 + 2^5 + 2^4 = 112$$

The decimal equivalent is 112_{10}.

(d) 10011111_2: This number is negative because of its most significant bit being 1. We find the two's complement:

$$10011111_2 \to 01100000_2 + 1_2 \to 01100001_2 = 2^6 + 2^5 + 2^0 = 97_{10}$$

The decimal equivalent is -97_{10}.

Exercise 1.22. Convert the following two's complement binary numbers to decimal.

(a) 1110_2

(b) 100011_2

(c) 01001110_2

(d) 10110101_2

Solution:

(a) 1110_2: Since the msb is 1, this number is negative. Its two's complement is:

$$1110_2 \to 0001_2 + 1_2 \to 0010_2 = 2_{10}$$

The decimal equivalent is -10_2.

(b) 100011_2: The msb is 1, so this number is negative. Its two's complement is:

$$100011_2 \rightarrow 011100_2 + 1_2 \rightarrow 011101_2 = 2^4 + 2^3 + 2^2 + 2^0 = 29_{10}$$

The decimal equivalent is -29_{10}.

(c) 01001110_2: Since the msb is 0, this number is positive. Calculating the two's complement is not necessary, and we can start with the conversion:

$$2^6 + 2^3 + 2^2 + 2^1 = 78_{10}$$

(d) 10110101_2: The msb is 1, so this number is negative. Its two's complement is

$$10110101_2 \rightarrow 01001010_2 + 1_2 \rightarrow 01001011_2 = 2^6 + 2^3 + 2^1 + 2^0 = 75_{10}$$

Its decimal equivalent is -75_{10}.

Exercise 1.41. How many 5-bit two's complement numbers are greater than 0? How many are less than 0? How would your answers differ for sign/magnitude numbers?

Solution: There are $2^5 = 32$ numbers in a 5-bit two's complement system. Of these, half (16) are negative. Therefore, there are 15 numbers greater than 0. If it were sign/magnitude, we would have 15 numbers greater than 0, and 15 less than 0, with the remaining two being -0 and +0.

Exercise 1.42. How many 7-bit two's complement numbers are greater than 0? How many are less than 0? How would your answers differ for sign/magnitude numbers?

Solution: There are $2^7 = 128$ numbers in a 5-bit two's complement system. Of these, half (64) are negative. Therefore, there are 63 numbers greater than 0. If it were sign/magnitude, we would have 63 numbers greater than 0, and 63 less than 0, with the remaining two being -0 and +0.

Exercise 1.43. How many bytes are in a 32-bit word? How many nibbles are in the 32-bit word?

Solution: A byte is 8 bits. Therefore, a 32-bit word has 4 bytes. A nibble is 4 bits, which represents one hexadecimal. THEREFORE, A 32-bit word has 8 nibbles.

Exercise 1.44. How many bytes are in a 64-bit word?

Solution: There 8 bytes in a 64-bit word.

Exercise 1.45. A particular DSL modem operates at 768 kbits/sec. How many bytes can it receive in 1 minute?

Solution: Since 768 bits is equivalent to 96 bytes, this is equivalent to 96 kilo bytes per second. Multiplying by 60, we get 5,760 kilo bytes per minute, or 5.76 mega bytes per minute, or 5,760,000 bytes in 1 minute.

Exercise 1.46. USB 3.0 can send data at 5 Gbits/sec. How many bytes can it send in 1 minute?

Solution: We multiply by 60 obtain 300 Gbits in one minute, or 300,000,000,000 bits, which is equivalent to 37,500,000,000 bytes.

Exercise 1.47. Hard drive manufacturers use the term "megabyte" to mean 10^6 bytes and "gigabyte" to mean 10^9 bytes. How many real GBs (i.e., GiBs) of music can you store on a 50 GB hard drive?

Solution: Since a manufacturer means 50 gigabyte, this means $50 \cdot 10^9$ bytes. However, since 1 GiB=1024MiB, and 1 MiB=1024 KiB, and 1 KiB=1024 bytes, we divide this by $(1024)^3$ to obtain approximately 46.6 GiB.

Exercise 1.48. Estimate 2^{31} without using a calculator.

Solution: Since $2^{10} \approx 1,000$, $2^{20} \approx 1,000,000$, and $2^{30} \approx 1,000,000,000$, it follows that $2^{31} \approx 2,000,000,000$, or 2 billion.

Exercise 1.49. A memory on the Pentium II microprocessor is organized as a rectangular array of bits with 2^8 rows and 2^9 columns. Estimate how many bits it has without using a calculator.

Solution: The memory has $2^8 \cdot 2^9 = 2^{17}$ bits. Since $2^{10} \approx 1,000$ and $2^7 = 128$, this amounts to about $128,000$ bits, or 128 kbits.

Exercise 1.52. Perform the following additions of unsigned binary numbers. Indicate whether the sum overflows a 4-bit result.

(a) $1000_2 + 0100_2$

(b) $1101_2 + 1011_2$

Solution:

$$\text{(a)} \quad \begin{array}{r} 1\,0\,0\,0 \\ +\,0\,1\,0\,0 \\ \hline 1\,1\,0\,0 \end{array}$$

The result does not overflow.

$$\text{(b)} \quad \begin{array}{r} 1\,1\,0\,1 \\ +\,1\,0\,1\,1 \\ \hline 1\,1\,0\,0\,0 \end{array}$$

The result overflows.

Exercise 1.53.

(a) $10011001_2 + 01000100_2$

(b) $11010010_2 + 10110110_2$

Solution:

(a) The result does not overflow:

$$\begin{array}{r} 1\,0\,0\,1\,1\,0\,0\,1 \\ +\,0\,1\,0\,0\,0\,1\,0\,0 \\ \hline 1\,1\,0\,1\,1\,1\,0\,1 \end{array}$$

(b) The result overflows:

$$\begin{array}{r} 1\,1\,0\,1\,0\,0\,1\,0 \\ +\,1\,0\,1\,1\,0\,1\,1\,0 \\ \hline 0\,1\,1\,0\,1\,0\,0\,0 \end{array}$$

Exercise 1.66. A flying saucer crashes in a Nebraska cornfield. The FBI investigates the wreckage and finds an engineering manual containing an equation in the Martian number system: 325+42=411. If this equation is correct, how many fingers would you expect martians to have?

Solution: The equation is true in base 6. Therefore, martians likely have 6 fingers.

Exercise 1.67. Ben Bitdiddle and Alyssa P. Hacker are having an argument. Ben says, 'All integers greater than zero and exactly divisible by six have exactly two 1's in their binary representation." Alyssa disagrees. She says, "No, but all such numbers have an even number of 1's in their binary representation". Do you agree with Ben or Alyssa or both or neither? Explain

Solution: We can experiment by writing some multiples of 6_{10} in binary. Certainly:

$$6_{10} = 110_2, \quad 12_{10} = 1100, \quad 18_{10} = 10010_2, \quad 24_{10} = 11000_2, \quad 30_{10} = 11110_2$$

Since 30_{10} has more than two 1's, it turns out that Ben is wrong. We continue:

$$36_{10} = 100100_2, \quad 42_{10} = 101010_2$$

Given 42_{10} has three 1 bits, it turns out Alyssa is wrong, too.

Exercise 1.68. Ben Bitdiddle and Alyssa P. acker are having another argument. Ben says, "I can get the two's complement of a number by subtracting 1, then inverting all the bits of the result." Alyssa says, "No, I can do it by examining each bit of the number, starting with the least significant bit. When the first 1 is found, invert each subsequent bit." Do you agree with Ben or Alyssa or both or neither? Explain.

Solution: The two's complement of an N-bit number is obtained by subtracting the number from 2^N. This number by an $(N+1)$-bit number with a 1 in the most significant bit and 0 everywhere else. It can be obtained by adding 1 to an N-bit -1 (represented by all 1 bits). Hence, the N-bit two's complement of x is

$$2^N - x = (11\cdots1 + 1) - x = (11\cdots1 - x) + 1$$

In words, we invert the bits of x (by subtracting it from -1) and then add 1. Ben's description, subtracting 1 and then inverting the bits of the result, is essentially:

$$1 - (x - 1) = x + 2$$

This leads to an incorrect answer. To demonstrate, given the 4-bit 1001_2, two's complement is

$$1001_2 \to 0110_2 + 1 \to 0111_2$$

According to Ben's method, first we subtract 1 (with overflow):

$$1001_2 \to (1001_2 - 1) = 1001_2 + 1111_2 \to 1100_2$$

and then inverting its bits we get 0011_2, which is not correct. Alyssa's method, however, works correctly on this number, and it holds true. Suppose we have a nonzero, N-bit binary number, which therefore has a least significant 1-bit. Since finding the two's complement involves inverting all the bits and then adding 1, this means the rightmost 0 inverts to 1, and adding 1 to it inverts it back to 0, with a 1 carried to the next place. This happens for all 0's to the right of the least significant 1-bit. The least significant 1 is inverted to a 0, and when the carry bit reaches it, it becomes 1, but this time there is no carry. Therefore, all the bits to the left of it are unaffected after the bit inversion operation.

Exercise 1.69. Write a program in your favorite language (e.g., C, Java, Perl) to convert numbers from binary to decimal. The user should type in an unsigned binary number. The program should print the decimal equivalent.

Solution:

The following standalone C program, `binarytodecimal.c`, does the job. Instead of accepting command-line arguments, it prompts the user for a binary string, and upon receiving a valid input, it outputs the decimal result.

```
#include <stdio.h> /* printf, getchar, EOF */
#include <ctype.h> /* isspace */
#define MAXBITS 64

int readbits(char s[]);
unsigned long bintodecimal(char *s);

/* read unsigned binary number with up MAXBITS bits and display its decimal
   value */
int main(int argc, char *argv[])
```

```
{
    char s[MAXBITS + 1];

    printf("Enter an %d-bit binary number: ", MAXBITS);
    if (!readbits(s))
    return 1;  /* exit unsuccessfully */

    unsigned long n = bintodecimal(s);
    printf("\nThe unsigned decimal equivalent is %lu\n", n);
}

/* readbits: reads contiguous binary number of up to MAXBITS bits into s until
    newline or EOF is encountered, skipping white space
* returns 1 if successful, and 0 otherwise
*/
int readbits(char s[])
{
    int c, i;
    i = 0;
    while (isspace(c = getchar())) /* skip leading white space */
    ;

    if (c != '1' && c != '0') {
        printf("Invalid binary digit: %c\n", *s);
        s[0] = '\0';
        return 0;
    }

    s[i++] = c;
    while (i < MAXBITS && ((c = getchar()) == '0' || c == '1')) /* read rest of
        bits */
    s[i++] = c;

    if (i == MAXBITS)
    c = getchar();

    while (c != '\n' && c != EOF && isspace(c))  /* skip trailing white space
        besides newline */
    c = getchar();

    if (c == '1' || c == '0') {
        printf("Error: either already received %d bits, or encountered additional
            bit separated by space: %c\n", MAXBITS, c);
        return 0;
    }
    else if (c != '\n' && c != EOF) {
        printf("Encountered invalid character: %c\n", c);
```

```
        return 0;
    }
    s[i] = '\0';
    return 1;
}

/* bintodecimal: converts unsigned binary string in s to decimal. Expects MSB to
    be at s[0] */
unsigned long bintodecimal(char *s)
{
    unsigned long n;
    for (n = 0; *s != '\0'; s++)
    n = 2 * n + (*s - '0');
    return n;
}
```

Exercise 1.70. Repeat Exercise 1.69, but convert from an arbitrary base b_1 to another base b_2, as specified by the user. Support bases up to 16, using the letters of the alphabet for digits greater than 9. The user should enter b_1, b_2, and then the number to convert in base b_1. The program should print the equivalent number in base b_2.

Solution: The following standalone C program, `baseconverter.c`, expects three command-line arguments: `b1`, `b2`, `n`. It is expected that `n` is an unsigned numerical string in base `b1` that fits in a 64-bit word. It converts `n` from base `b1` to base `b2` by first converting to base 10:

```
#include <stdio.h> /* printf, getchar, EOF */
#include <ctype.h> /* toupper, tolower */
#include <stdlib.h> /* strtol */
#include <string.h> /* strlen */
#define DECIMAL 10
#define BINARY 2
#define HEX 16
#define MAXBITS 64

int readbase(char *s);
int checkbase(int c, int base);
int validatenumber(char *s, int base);
unsigned long todecimal(char *s, int base);
void tobase(unsigned long n, int base, char *s);
void reverse(char *s);

/* given number bases b1 and b2, followed by an unsigned number in base b1,
    outputs the number in base b2
*
* Assumes the given number fits in 64 bits.
* */
```

```
int main(int argc, char *argv[])
{
    int b1, b2;
    if (!(b1 = readbase(*++argv)) || !(b2 = readbase(*++argv))||
        !validatenumber(*++argv, b1)) {
        printf("Usage: <base1> <base2> <n>\n");
        printf("\t base1 and base2 must be %d through %d, and n must be an
            unsigned number in base2 that can fit in a 64-bit word\n", BINARY,
            HEX);
        return 1;
    }
    unsigned long n;
    char s[MAXBITS + 1];

    n = todecimal(*argv, b1); /* b1 to decimal */
    tobase(n, b2, s);     /* decimal to b2 */
    printf("%s\n", s);
}

/* todecimal: converts unsigned integer string a given base to decimal: supports
    bases BINARY through HEX */
unsigned long todecimal(char *s, int base)
{
    unsigned long n;
    for (n = 0; *s != '\0'; s++)
    n = base * n + (isdigit(*s)? (*s - '0'): (10 + (tolower(*s) - 'a')));
    return n;
}

/* tobasE: converts unsigned integer n to a string representation in the given
    base */
void tobase(unsigned long n, int base, char *s)
{
    int r;
    char *p;
    for (p = s; n ; n /= base, p++) {
        r = n % base;
        *p = (char) ((r < 10) ? '0' + r : ('A' + (r-10)) );
    }
    *p = '\0';
    reverse(s);
}

/* readbase: parses numerical base from s and returns it, returning 0 if invalid
    */
int readbase(char *s)
{
```

```
    if (s == NULL)
    return 0;
    char *endp;
    int base = (int) strtol(s, &endp, DECIMAL);

    if (*endp != '\0') {
        printf("Error: invalid character encountered when reading base: %s\n",
            endp);
        return 0;
    }
    if (base < BINARY || base > HEX) {
        printf("Error: invalid base %d, expected base between %d and %d\n", base,
            BINARY, HEX);
        return 0;
    }
    return base;
}

/* checkdigit: returns 1 if c is a valid digit in the given base, 0 otherwise.
   Supported bases: binary through hexadecimal */
int checkdigit(int c, int base)
{
    if (base < BINARY || base > HEX) { /* invalid base */
        printf("Error: Only bases %d through %d are supported, but got: %d\n",
            BINARY, HEX, base);
        return 0;
    }

    if (base <= DECIMAL && (!isdigit(c) || (c - '0') >= base))
    return 0;

    int maxletter = 'a' + (base - DECIMAL) - 1;
    return isdigit(c) || (base > DECIMAL && tolower(c) <= maxletter);
}

/* validatenumber: returns 1 if s is valid in the given base, 0 otherwise */
int validatenumber(char *s, int base)
{
    if (s == NULL)
    return 0;
    while (*s)
    if (!checkdigit(*s++, base))
    return 0;
    return 1;
}

/* reverse: reverse string s in-place */
```

```
void reverse(char *s)
{
    int c, i, j;
    for (i = 0, j = strlen(s) - 1; i < j; i++, j--) {
        c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}
```

Exercise 1.71. Draw the symbol, Boolean Equation, and truth table for

(a) a three-input OR gate

(b) a three-input exclusive OR (XOR) gate

(c) a four-input XNOR gate

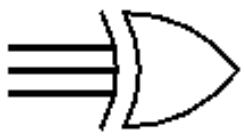
Solution:

(a) Given inputs A,B,C, and output Y, the equation would be $Y = A + B + C$. The truth table and symbol are below:

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

(b) Given inputs A,B,C, and output Y, the equation would be $Y = A \oplus B \oplus C$. The truth table and symbol are below:

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0



A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

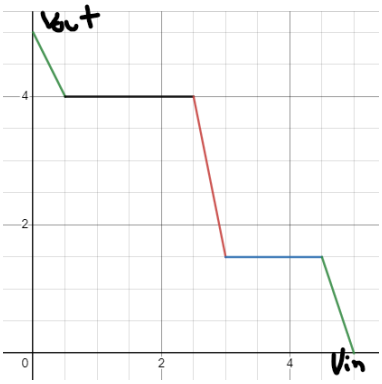


Figure 1: DC Transfer Characteristics: Exercise 1-78

(c) Given inputs A,B,C, D and output Y, the equation would be $Y = \overline{A \oplus B \oplus C \oplus D}$. The truth table and symbol are below:

Exercise 1.77. How many different truth tables exist for Boolean functions of N variables.?

Solution: When $N = 2$, the result is 16. The reason is that an input is a pair (A, B), and there are 4 such pairs, and hence 4 unique outputs. Since each output is 0 or 1, this means there are 2^4 unique functions. If $N = 3$, there are 2^3 possible outputs, and hence 2^8 possible functions. In general, there are 2^{2^N} Boolean functions of N variables.

Exercise 1.78. Is it possible to assign logic levels so that a device with the transfer characteristics shown in Figure 1 would serve as an inverter? If so,what are the input and output low and high levels (V_{IL}, V_{OL}, V_{IH}, and V_{OH}) and noise margins (NM_L and NM_H)? If not, explain why not.

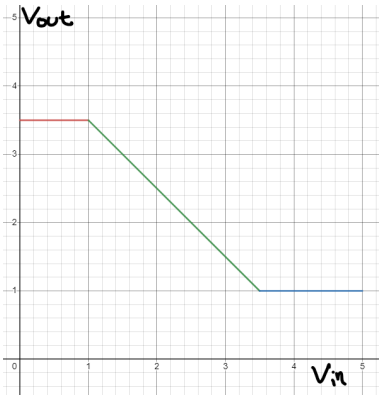


Figure 2: DC Transfer Characteristics: Exercise 1-79

Solution: Yes. We can choose $V_{OH} = 4$, $V_{IH} = 3$, $V_{IL} = \frac{5}{2}$, and $V_{OL} = \frac{3}{2}$. Then $NM_H = NM_L = 1$.

Exercise 1.79. Repeat Exercise 1.78 for the transfer characteristics shown in Figure

Solution: Because the slope of the transfer characteristics is never better than -1, the system does not have gain. Put another way, we require $V_{OL} < V_{IL}$ and $V_{OH} > V_{IH}$, but here they would be equal.