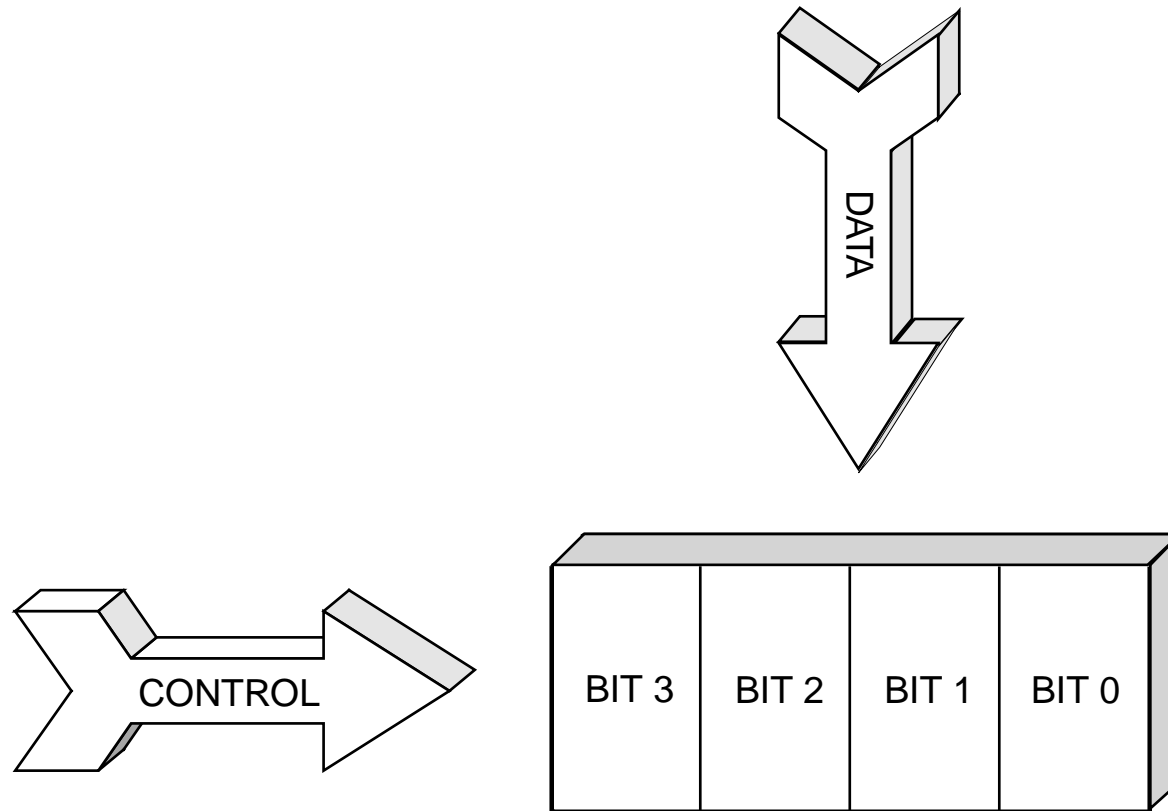


THE DATAPATH

- One of the five basic components of a computer
- All datapath operations are based on one simple idea: **Operate repetitively on chunks of data using the same algorithm**
 - ▷ A “chunk” can be a bit, a byte (8 bits), a short word (16 bits), a long word (32 bits), or a quad word (64 bits)]
 - ▷ The MIPS instruction set architecture uses words of 32 or 64 bits
- RISC machines load data chunks into registers before operating on them
 - ▷ The cycle of datapath operations is
 - Load data from main memory into registers
 - Operate on the data using the ALU, FPU, etc.
 - Store the result back to main memory

BIT SLICING: THE SAME ALGORITHM IS APPLIED TO ALL BITS IN PARALLEL



DATA REPRESENTATIONS

- Harvard architecture pioneered by Howard Aiken:
 - ▷ Separate memories for instructions and data
 - ▷ Logical extension: Separate memory for each kind of data
 - Advantage: Data types can be recognized unambiguously
 - Disadvantage: Inefficient utilization of memory
- von Neumann architecture:
 - ▷ A single memory is used for both instructions and data
 - ▷ All types of data are stored in the same memory
 - Advantage: Efficient utilization of memory
 - Disadvantage: Data types cannot be recognized unambiguously
- The correct design decision is generally to use a single kind of memory for all data
 - ▷ Software has to decide how a particular data item should be interpreted

ON THE INTERPRETATION OF WORDS...

- In a computer's memory (RAM, CPU registers, etc.) data is organized into **words**
 - ▷ A MIPS word is 32 bits long
 - ▷ A PC word is 16 bits long
 - ▷ A word is just a string of 1's and 0's
 - ▷ Software determines how the word is interpreted
- Example: 10001101001010000000000001011000
 - ▷ Integer: 2,368,208,984 (unsigned) or $-1,926,758,312$ (signed)
 - ▷ Floating point: -5.177×10^{-31}
 - ▷ 4-character string: `i(\squareX`
 - ▷ Hardware instruction: `lw $t0, 88($t1)`
- We're going to start by learning how **integers** can be represented using 1's and 0's

REPRESENTATIONS OF INTEGERS (1)

- The Roman representation of numbers:

$$\text{MCMXCVI} = 1996$$

- ▷ $M = 1000$, $C = 100$, $X = 10$, $V = 5$, $XC = 90$, etc.
- ▷ $X \times C = M$, etc. — a nightmare!

- A better idea — positional notation:

- ▷ Decimal representation (base 10)

$$1996_{10} = 1 \cdot 10^3 + 9 \cdot 10^2 + 9 \cdot 10^1 + 6 \cdot 10^0$$

- Each digit multiplies the power of the base (10) that corresponds to the digit's position, starting with 0 at the right and increasing to the left
- To multiply by the n^{th} power of the base (10 here), shift left n places

UNSIGNED INTEGERS (1)

- **Unsigned integer:** A representation of a non-negative integer using binary or hexadecimal positional notation

- Examples:

▷ Unsigned byte:

$$1011\ 0101_2 = B5_{16} = 181_{10}$$

Range of unsigned byte values is 0 to 255_{10}

▷ Unsigned 32-bit integer:

$$\begin{aligned} 1111\ 0111\ 0011\ 0001\ 1011\ 1001\ 1111\ 1100_2 &= F7\ 31\ B9\ FC_{16} \\ &= 4,147,231,228_{10} \end{aligned}$$

Range of unsigned 32-bit integer values is 0 to $4,294,967,295_{10}$

- Used to represent computer addresses, characters, image data

REPRESENTATIONS OF INTEGERS (2)

- Octal representation (base 8):

$$(255)_{10} = (377)_8 = 3 \cdot 8^2 + 7 \cdot 8^1 + 7 \cdot 8^0$$

▷ Octal digits are 0, 1, 2, 3, 4, 5, 6, 7

▷ Convert 196_{10} to the octal representation:

$$64 = 8^2 < 196 < 8^3 = 512 \Rightarrow \text{highest power of 8 is } 8^2$$

$$196 \div 64 = 3 \text{ (remainder = 4)}$$

$$1 = 8^0 < 4 < 8 = 8^1 \Rightarrow \text{next highest power of 8 is } 8^0$$

$$196_{10} = 3 \cdot 8^2 + 0 \cdot 8^1 + 4 \cdot 8^0 = 304_8$$

- ▷ For more efficient conversion methods, see slides on base conversion
- ▷ The basic arithmetic operations (addition, subtraction, multiplication and division) use shifts and carries, just as in the decimal representation

REPRESENTATIONS OF INTEGERS (3)

- Binary representation (base 2):

$$\begin{aligned}(255)_{10} &= (11111111)_2 \\ &= 1 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 \\ &\quad + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \\ &= 2^8 - 1\end{aligned}$$

- ▷ Binary digits are 0, 1
- ▷ Binary digit = bit
- ▷ The binary representation is convenient for logic design (only 2 voltage levels — high and low)
- ▷ Binary representations of numbers are inconvenient for humans to write or read (too many digits)

BINARY-TO-DECIMAL CONVERSION OF UNSIGNED 8-BIT INTEGERS

number to be converted → 1 0 1 1 0 0 1 1
powers of 2 → 128 64 32 16 8 4 2 1

$$\begin{aligned} \text{result} &= 128+32+16+2+1 \\ &= 179 \end{aligned}$$

REPRESENTATIONS OF INTEGERS (4)

- ASCII representation of characters as unsigned integers:

$$C \leftrightarrow 0100\ 0011_2 = 67_{10}$$

$$c \leftrightarrow 0110\ 0011_2 = 99_{10}$$

- ▷ A character is stored as 1 byte in computer memory
- ▷ In C, characters are 1-byte integers
- ▷ Maximum ASCII value is $127_{10} = 0111\ 1111_2$
- ▷ Largest unsigned integer representable with 8 bits is $255_{10} = 1111\ 1111_2$
- ▷ Usually one uses a hexadecimal representation to keep from writing too many digits

REPRESENTATIONS OF INTEGERS (5)

- Hexadecimal digits (base 16):

$$(0)_{16} = (0)_{10} = (0000)_2,$$

$$(1)_{16} = (1)_{10} = (0001)_2,$$

$$(2)_{16} = (2)_{10} = (0010)_2,$$

$$(3)_{16} = (3)_{10} = (0011)_2,$$

$$(4)_{16} = (4)_{10} = (0100)_2,$$

$$(5)_{16} = (5)_{10} = (0101)_2,$$

$$(6)_{16} = (6)_{10} = (0110)_2,$$

$$(7)_{16} = (7)_{10} = (0111)_2,$$

$$(8)_{16} = (8)_{10} = (1000)_2,$$

$$(9)_{16} = (9)_{10} = (1001)_2,$$

$$(A)_{16} = (10)_{10} = (1010)_2,$$

$$(B)_{16} = (11)_{10} = (1011)_2,$$

$$(C)_{16} = (12)_{10} = (1100)_2,$$

$$(D)_{16} = (13)_{10} = (1101)_2,$$

$$(E)_{16} = (14)_{10} = (1110)_2,$$

$$(F)_{16} = (15)_{10} = (1111)_2.$$

REPRESENTATIONS OF INTEGERS (6)

- Hexadecimal representation (base 16):

$$(3F)_{16} = (0011\ 1111)_2 = (63)_{10}$$

$$(FF)_{16} = (1111\ 1111)_2 = (255)_{10}$$

- ▷ 1 hexadecimal digit = 4 bits
- ▷ Easy to convert between hexadecimal and binary
- ▷ Hexadecimal representations are much more compact than binary representations
- ▷ In C, SAL and MIPS assembler, a hexadecimal representation is preceded with 0x (zero followed by lowercase x):

0x3F stands for $(3F)_{16}$

BASE CONVERSION (1)

- Conversion from binary representation (base 2) to base 2^k
 - ▷ In base 2^k , there are 2^k different digits (one for every possible pattern of k bits)
 - ▷ Rule for conversion of the binary representation of any unsigned integer n :
 - Group the binary digits of n into sets of k bits each, starting at the least significant bit
 - If necessary, add zero digits at the most left (most significant) end of the binary representation of n to make a full set of k bits
 - Now convert the sets of k bits into digits in base 2^k
 - ▷ Example: Convert 10110101 to base $8 = 2^3$:

$$10110101 = \underbrace{010}_2 \underbrace{110}_6 \underbrace{101}_5 = 265_8$$

BASE CONVERSION (2)

- Conversion from decimal representation (base 10) to base β
 - ▷ Let $d_k =$ coefficient of β^k , $n =$ number to be converted,
 $d_l d_{l-1} \cdots d_0 =$ base- β expansion of n (to be computed)
 - ▷ Recursive division algorithm:
 - l is the integer such that $\beta^l \leq n < \beta^{l+1}$
 - Set $k = l$, and compute $d_l = \lfloor \frac{n}{\beta^l} \rfloor =$ integer part of n/β^l
 - Set $n \mapsto n - d_k \beta^k$, $k \mapsto k - 1$, and repeat until $k = 0$
 - ▷ Example: Convert 2310_{10} to base 16:
 - $16^2 = 256 < 2310 < 4096 = 16^3 \Rightarrow l = 2$
 - $d_2 = \lfloor \frac{2310}{256} \rfloor = 9$
 - $n \mapsto 2310 - 9 \times 256 = 2310 - 2304 = 6$
 - $\Rightarrow d_1 = 0$ and $d_0 = 6$

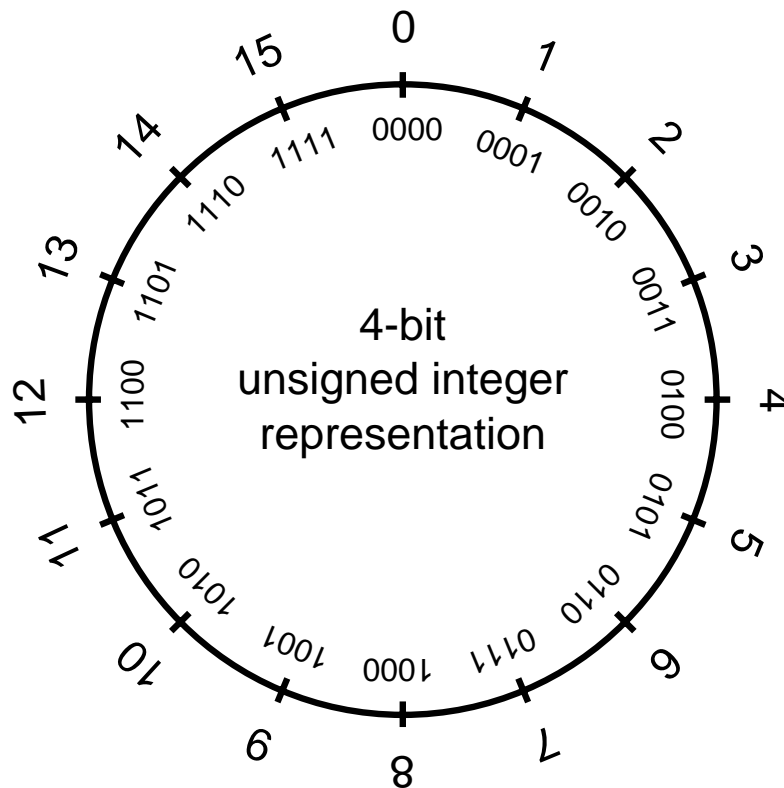
$$2310_{10} = 906_{16}$$

END OF LINE CONVENTIONS

- Conventions for ending a line of text and starting a new line:

| Operating system | End of line (ASCII) | C | Hexadecimal |
|------------------|---------------------|-------------------|-------------|
| UNIX | LF | <code>\n</code> | 0A |
| MacOS | CR | <code>\r</code> | 0D |
| VMS | CR LF | <code>\r\n</code> | 0D0A |
| MS-DOS | CR LF | <code>\r\n</code> | 0D0A |

4-BIT UNSIGNED REPRESENTATION



© C. D. Cantrell 1996

UNSIGNED INTEGERS (2)

• Unsigned integer arithmetic:

▷ Addition:

| Hex | Binary |
|---|---|
| 4D | 01001101 |
| +49 | +01001001 |
| <hr style="width: 100%; border: 0.5px solid black;"/> | <hr style="width: 100%; border: 0.5px solid black;"/> |
| 96 | 10010110 |

▷ Subtraction:

| Hex | Binary |
|---|---|
| 4D | 01001101 |
| -3F | -00111111 |
| <hr style="width: 100%; border: 0.5px solid black;"/> | <hr style="width: 100%; border: 0.5px solid black;"/> |
| 0E | 00001110 |

Subtraction can produce negative numbers (covered in later slides)

UNSIGNED INTEGERS (3)

• Integer overflow:

- ▷ Occurs when the result of an addition or multiplication has too many digits to be represented in the chosen format; for example,

| Hex | Binary |
|---|--|
| FF | 1111 1111 |
| +01 | +0000 0001 |
| $\begin{array}{r} \text{FF} \\ +01 \\ \hline 1\ 00 \end{array}$ | $\begin{array}{r} 1111\ 1111 \\ +0000\ 0001 \\ \hline 1\ 0000\ 0000 \end{array}$ |

- ▷ The extra digits are discarded, but the hardware recognizes that an overflow has occurred

UNSIGNED INTEGERS (4)

- Unsigned integer arithmetic:

- ▷ Multiplication:

| Decimal | Hex | Binary |
|---|---|---|
| 76 | 4C | 1001100 |
| ×49 | ×31 | ×110001 |
| <hr style="width: 100%; border: 0.5px solid black;"/> | <hr style="width: 100%; border: 0.5px solid black;"/> | <hr style="width: 100%; border: 0.5px solid black;"/> |
| 684 | 4C | 1001100 |
| 3040 | E40 | 00000000 |
| <hr style="width: 100%; border: 0.5px solid black;"/> | <hr style="width: 100%; border: 0.5px solid black;"/> | <hr style="width: 100%; border: 0.5px solid black;"/> |
| 3724 | E8C | 000000000 |
| | | 0000000000 |
| | | 10011000000 |
| | | 100110000000 |
| | | <hr style="width: 100%; border: 0.5px solid black;"/> |
| | | 111010001100 |

- ▷ Only shifts and additions are necessary

ADDITIVE INVERSE (1)

- The **additive inverse** of a number n is defined as the number m such that

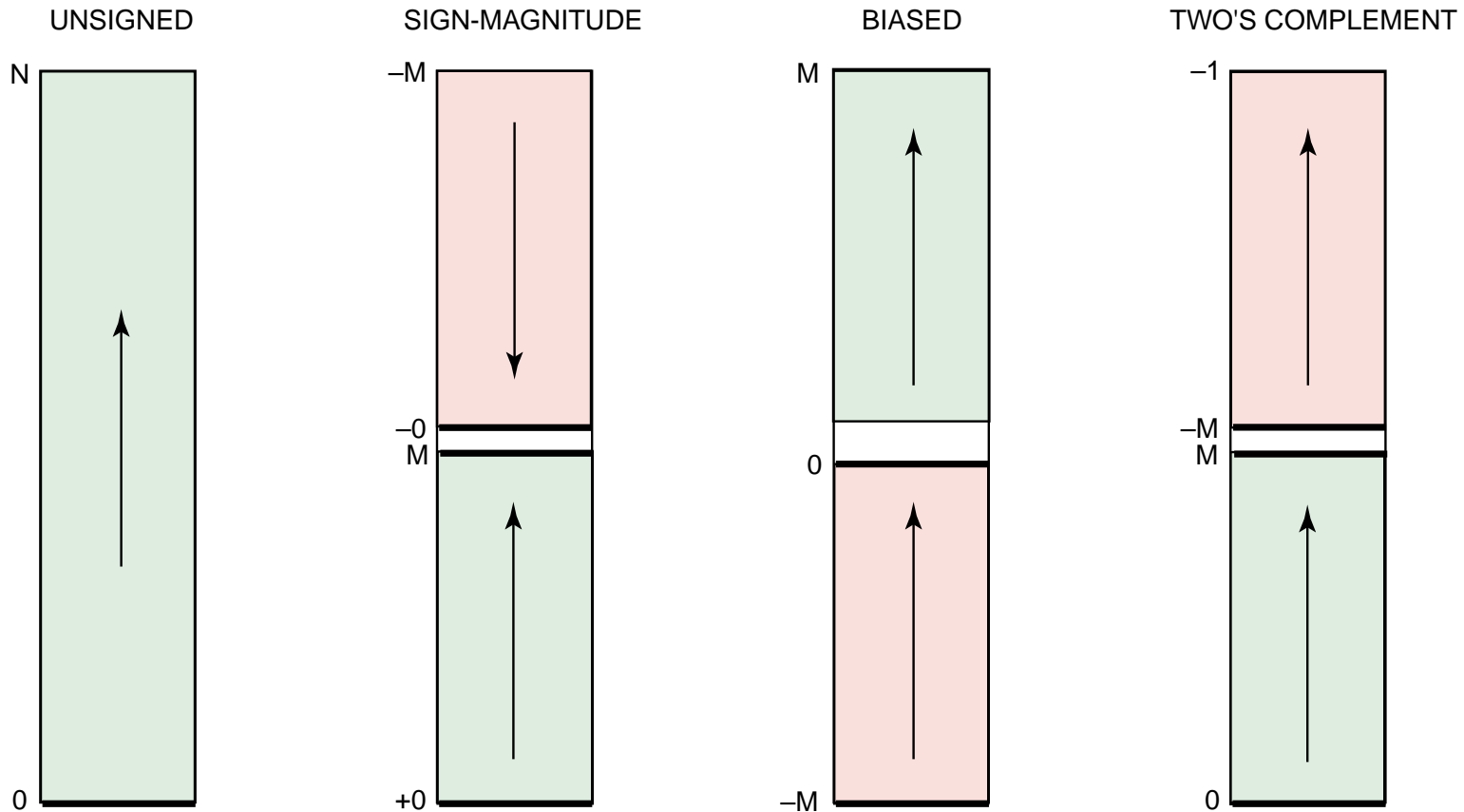
$$m + n = 0$$

- ▷ Addition by m undoes (inverts) addition by n
- ▷ Usually one writes m as

$$m = -n$$

- ▷ Subtraction is the inverse of addition
 - Simplest way to subtract n is to add $-n$
- ▷ Problems for computer architects:
 - How to represent $-n$ for any positive integer n that can be represented in the computer
 - How to maximize speed

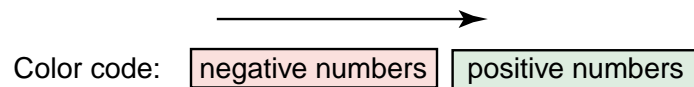
COMPARISON OF INTEGER REPRESENTATIONS



$M := 2^{n-1} - 1$

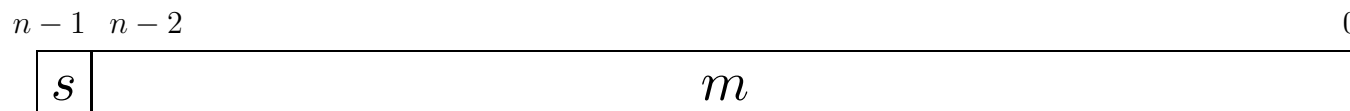
Arrows show the direction of increasing **numerical** order

$N := 2^n - 1$



SIGN-MAGNITUDE REPRESENTATION (1)

- Bit sequence:



Value assigned to this bit string in the sign-magnitude representation:

$$k = (-1)^s m$$

where m is the unsigned integer represented by bits $n - 2$ through 0

- Major disadvantage:
 - 000...0 represents +0 and 100...0 represents -0
 - ▷ Two different representations for 0 would require additional hardware for numerical comparisons
- Used in floating-point representations

SIGN-MAGNITUDE REPRESENTATION (2)

- Features of the sign-magnitude representation:

▷ Largest representable positive integer:

$$k_{\max,+} = \underbrace{11 \cdots 1}_{n-1 \text{ bits}} = 2^{n-1} - 1$$

▷ Smallest positive integer represented is $000 \cdots 01 = 1$

▷ Most negative representable integer:

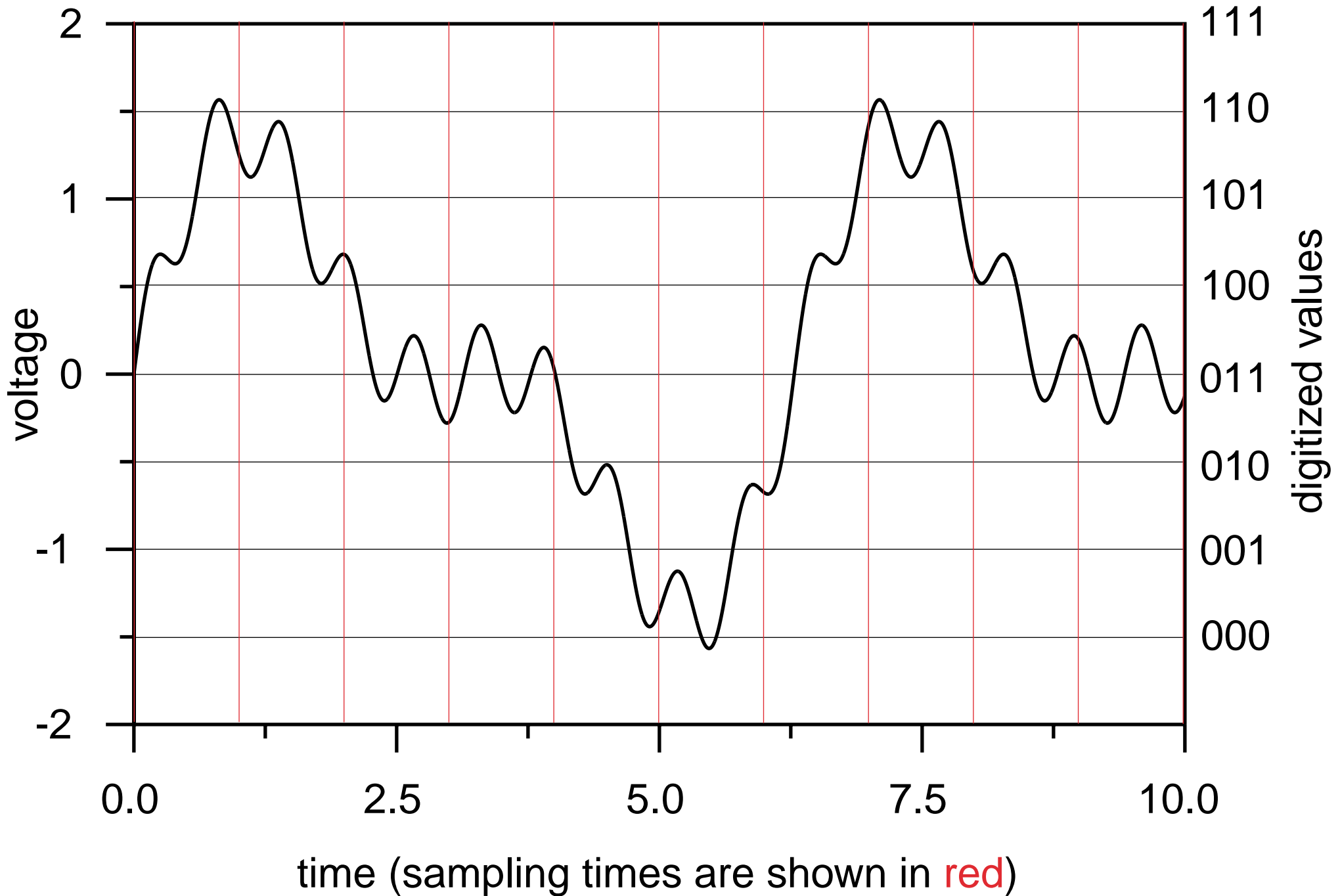
$$k_{\max,-} = -\underbrace{11 \cdots 1}_{n-1 \text{ bits}} = -(2^{n-1} - 1)$$

▷ Least negative integer less than zero is $100 \cdots 01 = -1$

- Dictionary (lexicographic) ordering:

$$+0 \prec 1 \prec \cdots \prec 2^{n-1} - 1 \prec -0 \prec -1 \prec \cdots \prec -(2^{n-1} - 1)$$

3-bit digitization of an analog signal



DIGITIZATION OF AN ANALOG SIGNAL

- How should one map a physical quantity that can be either positive or negative to a finite range of unsigned integer values?
 - ▷ This means that one needs an integer data representation in which unsigned integers can represent either positive or negative values
 - ▷ One way is to consider one of the non-zero values in the middle of the digital range as representing zero
 - ▷ Example:
 - With $n = 3$ bits one can represent $7 = 2^n - 1$ signal levels
 - ◇ The analog signal is sampled at regular time intervals
 - ◇ The most significant bit is on if the sampled voltage is positive
 - ◇ Less significant bits correspond to smaller signal ranges
 - ◇ Digitized sampled values vs. time in graph on previous slide:

| | | | | | | | | | | | |
|----------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| t_s | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| $V(t_s)$ | 011 | 101 | 100 | 010 | 011 | 000 | 010 | 110 | 100 | 011 | 011 |

BIASED REPRESENTATION (1)

- In a biased integer representation, the value k assigned to a bit string $b_{n-1} \cdots b_0$ is

$$k = -B + \underbrace{\sum_{j=0}^{n-1} b_j 2^j}_{\text{integer } b_{n-1} \cdots b_0}$$

▷ The integer B is called the **bias**

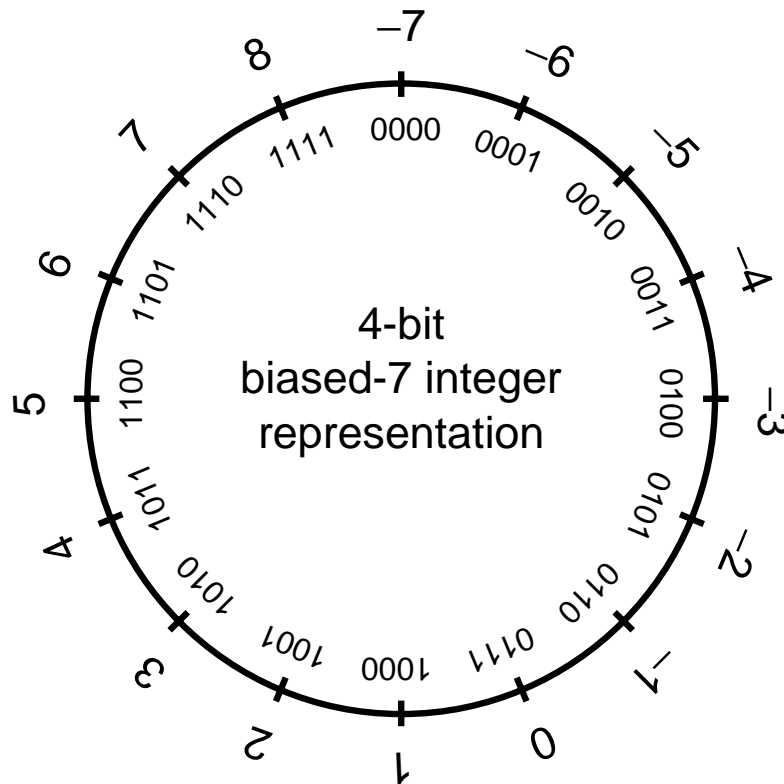
▷ Usually $B = \left\lfloor \frac{2^n - 1}{2} \right\rfloor = 2^{n-1} - 1$

▷ Example ($n = 2$, $B = 1$):

| k | $(k + B)_{10}$ | $(k + B)_2$ |
|-----|----------------|-------------|
| -1 | 0 | 00 |
| 0 | 1 | 01 |
| 1 | 2 | 10 |

▷ A biased representation is used to represent the exponent in the IEEE-754 floating-point representations

4-BIT BIASED REPRESENTATION



© C. D. Cantrell 1996

BIASED REPRESENTATION (2)

- Addition in a biased integer representation:

- ▷ Relation between a signed integer k and its unsigned representative u :

$$k = u - B \quad \Rightarrow \quad u = k + B$$

where B is the bias

- ▷ Let

$$k_1 = u_1 - B, \quad k_2 = u_2 - B$$

- ▷ Sum:

$$k_1 + k_2 = (u_1 + u_2) - 2B$$

- ▷ The unsigned integer that represents $k_1 + k_2$ is

$$u = k_1 + k_2 + B = u_1 + u_2 - B$$

BIASED REPRESENTATION (3)

- Example of addition in a biased integer representation:

▷ Let $k_1 = 1$, $k_2 = -1$, $B = 1$

▷ The unsigned integers that represent k_1 and k_2 are

$$u_1 = 10, \quad u_2 = 00$$

▷ Add the unsigned integer representatives of k_1 and k_2 , and subtract the bias $B = 01$:

$$\begin{array}{r} 10 \\ +00 \\ \hline 10 \\ -01 \\ \hline 01 \end{array}$$

▷ The result, 01, is the unsigned integer that represents 0

BIASED REPRESENTATION (4)

- Realistic example of a biased integer representation:
 - ▷ Let $n = \text{number of bits} = 8$
 - To represent equal numbers of positive and negative integers, choose the bias B equal to the integer part of $\frac{1}{2}$ the maximum unsigned integer (255) that is representable with 8 bits:

$$B = \lfloor \frac{1}{2} \times 255 \rfloor = 127_{10} = 01111111_2$$

- ▷ Numerical value of the integer represented by an 8-bit integer u in the biased-127 representation:

$$k = u - B = u - 127$$

BIASED REPRESENTATION (5)

- Biased-127, 8-bit integer representation:

▷ Example of computation of numerical value:

$$u = 11010011_2 \Rightarrow k = 01010100_2 = 84_{10}$$

$$\begin{array}{r} u = \quad 11010011 \\ -B = -01111111 \\ \hline k = \quad 01010100 \end{array}$$

▷ Add the biased integer representatives of $k_1 = 01010100$ and $k_2 = -01010100$, and subtract the bias $B = 01111111$:

$$\begin{array}{r} 11010011 \\ +00101011 \\ \hline 11111110 \\ -01111111 \\ \hline 01111111 \end{array}$$

▷ The result, 01111111, is the unsigned integer that represents 0

BIASED REPRESENTATION (6)

- Properties of an n -bit biased representation with $B = \lfloor \frac{1}{2}(2^n - 1) \rfloor$:
 - ▷ The dictionary order of the u 's is the same as the numerical order of the k 's:
 $u = 00000000_2 \prec 00000001_2 \prec \dots \prec 01111111_2 \prec \dots \prec 11111111_2$
 $k = -127_{10} \prec -126_{10} \prec \dots \prec 0 \prec \dots \prec 128_{10}$
 - ▷ **Positive values of k correspond to biased, unsigned integers u in which the most significant bit is 1**
 - This is the opposite of the sign-magnitude and two's-complement representations, in which k is negative if the most significant bit of u is 1

LOGICAL VARIABLES

- A logical variable can have only two values, which are interpreted as “true” and “false”
 - ▷ Correspondence between bits and logical values:
$$1 = \text{“true”}, \quad 0 = \text{“false”}$$
 - ▷ Correspondences between electrical signals and logical values:
 - **Positive logic** (an **active-high** signal):
high voltage = “true”, low voltage = “false”
 - **Negative logic** (an **active-low** signal):
low voltage = “true”, high voltage = “false”
 - A signal set to logical 1 is said to be **asserted** or **active** or **true**
 - A signal set to logical 0 is said to be **deasserted** or **negated** or **false**

LOGIC GATES (1)

- A **logic gate** is an electronic circuit that implements one of the basic logical functions (AND, OR, NOT)

▷ Truth tables:

| a | \bar{a} |
|-----|-----------|
| 0 | 1 |
| 1 | 0 |

| a | b | $a + b$ | $a \cdot b$ |
|-----|-----|---------|-------------|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 |

LOGICAL NEGATION

- The operation of **negation** or **inversion** changes the value of a logical variable
 - ▷ The name of the negation operator is NOT
 - ▷ The **complement** of a logical variable a is the logical variable \bar{a} such that:
 - \bar{a} is true when a is false
 - \bar{a} is false when a is true
 - Then: $\bar{a} = \text{NOT}(a)$
 - ▷ An **inverter** gate implements 1-bit negation:

$$a \text{ --- } \triangleleft \text{--- } \text{NOT}(a) = \bar{a}$$

ONE'S COMPLEMENT

- The **one's complement** of an integer m is defined as the number \bar{m} that results from bitwise negation of the binary representation of m

▷ Example:

$$m = 11010101 \Rightarrow \bar{m} = 00101010$$

▷ For a representation that uses n bits,

$$m + \bar{m} = \underbrace{111 \dots 11}_{n \text{ bits}} = 2^n - 1$$

- The one's complement \bar{m} can represent the additive inverse, $-m$
 - ▷ In the one's complement representation of signed integers, two different bit patterns represent zero:

$$\underbrace{000 \dots 00}_{n \text{ bits}} \text{ and } \underbrace{111 \dots 11}_{n \text{ bits}}$$

TWO'S COMPLEMENT

- The **two's complement** of an integer m is defined as the number $-m_{\text{two}}$ that results from finding the one's complement of the binary representation and then adding 1

▷ Example:

$$m = 00101010$$

$$-m_{\text{two}} = \bar{m} + 1 = 11010101 + 1 = 11010110$$

▷ For a representation that uses n bits,

$$\begin{aligned} m + (-m_{\text{two}}) &= m + \bar{m} + 1 \\ &= \underbrace{111 \cdots 11}_{n \text{ bits}} + 1 \\ &= \underbrace{100 \cdots 00}_{n+1 \text{ bits}} \\ &= 2^n \end{aligned}$$

BINARY-TO-DECIMAL CONVERSION OF TWO'S COMPLEMENT 8-BIT INTEGERS

Example 1: number to be converted \rightarrow 1 0 1 1 0 0 1 1

powers of 2 \rightarrow $\begin{matrix} -1 \\ 128 \\ 64 \\ 32 \\ 16 \\ 8 \\ 4 \\ 2 \\ 1 \end{matrix}$

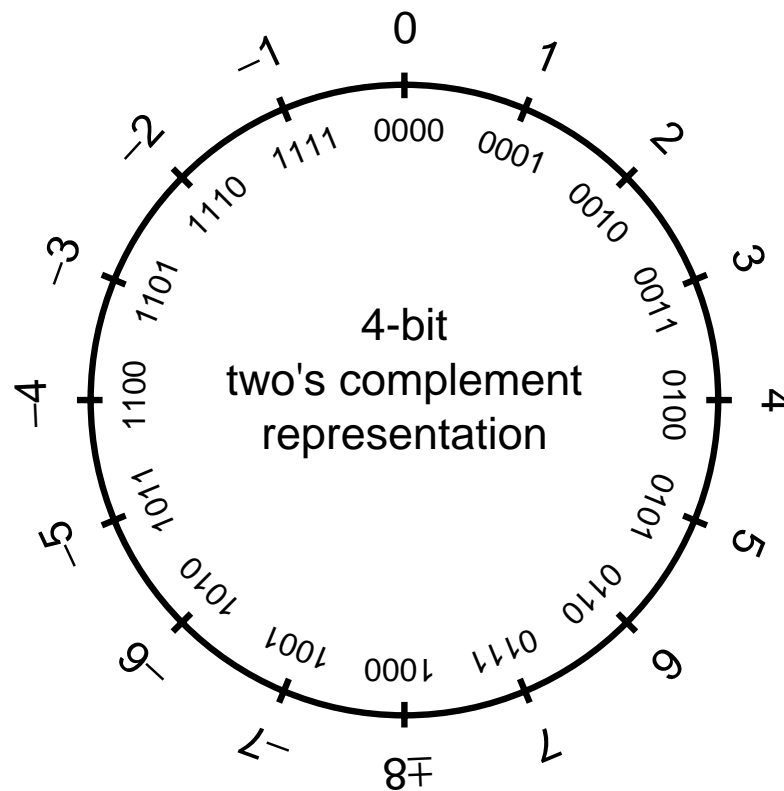
$$\begin{aligned} \text{result} &= -128 + 32 + 16 + 2 + 1 \\ &= -77 \end{aligned}$$

Example 2: number to be converted \rightarrow 0 0 1 1 0 0 1 1

powers of 2 \rightarrow $\begin{matrix} -1 \\ 128 \\ 64 \\ 32 \\ 16 \\ 8 \\ 4 \\ 2 \\ 1 \end{matrix}$

$$\begin{aligned} \text{result} &= 32 + 16 + 2 + 1 \\ &= 51 \end{aligned}$$

4-BIT 2's COMPLEMENT REPRESENTATION



© C. D. Cantrell 1996

TWO'S COMPLEMENT (2)

- The two's complement $-m_{\text{two}}$ can represent the additive inverse, $-m$
 - ▷ In the two's complement representation of signed integers, only one bit pattern represents zero:

$$\underbrace{000 \dots 00}_{n \text{ bits}}$$

This results in simple hardware for comparing a number to zero

- ▷ Computation of the additive inverse involves only two steps:
 - Invert
 - Add 1
- ▷ Since the ALU must be able to add in any case, since inversion is fast, and since there is only one representation for zero, the two's complement representation is the best choice for representing signed integers

TWO'S COMPLEMENT (3)

- Two's complement representation of an integer i :

| sign of i | two's compl. rep. of i |
|-------------|--------------------------|
| + | i |
| - | $2^n - i = 2^n + i$ |

- Example for $n = 8$:

$$i = -125_{10} \Rightarrow |i| = 7D_{16} = 0111\ 1101_2$$

$$2^n + i = 1000\ 0011_2$$

$$= \text{two's complement representation of } -125_{10}$$

BASE-64 ENCODING

- An encoding method that uses the 64 ASCII characters
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
to represent the $2^6 = 64$ possible 6-bit binary strings
 - ▷ Used to encode binary data for transmission through text-only communication channels (such as many electronic mail programs)
 - ▷ Uuencoding is more common because it uses bytes (8 bits)
 - ▷ Method:
 - The least common multiple of 6 and 8 is 24
 - 3 data bytes are used to fill a 24-bit buffer
 - A 24-bit string is encoded as 4 ASCII characters
 - Each 6-bit string represents an unsigned integer k ($0 \leq k \leq 63$)
 - The 6-bit string with value k is represented by the k th ASCII character in the list above ($000000 \mapsto A, \dots, 111111 \mapsto /$)