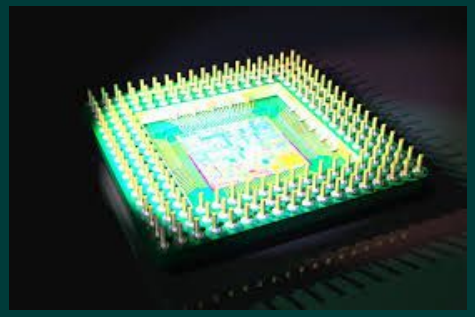



Computer Organization and Design

Data Representation in the Hardware

Units, Binary, Characters, Integers,

Hexadecimal, Numeric Strings, Number Conversions

Chapter 1, page 6

Chapter 2, Section 2.4 & 2.9 (page 114)

Units of Measurement

- Base 2 (binary) vs. Base 10 (decimal) Prefixes
  - Some confusion and ambiguity in usage
  - Talking about data (base 2) vs. data transmission (base 10)
- IEEE-1541 defines standard for representing binary vs. decimal quantities and unit symbols unambiguously

Decimal term	Abbreviation	Value	Binary term	Abbreviation	Value	% Larger
kilobyte	KB	10^3	kibibyte	KiB	2^{10}	2%
megabyte	MB	10^6	mebibyte	MiB	2^{20}	5%
gigabyte	GB	10^9	gibibyte	GiB	2^{30}	7%
terabyte	TB	10^{12}	tebibyte	TiB	2^{40}	10%
petabyte	PB	10^{15}	pebibyte	PiB	2^{50}	13%
exabyte	EB	10^{18}	exbibyte	EiB	2^{60}	15%
zettabyte	ZB	10^{21}	zebibyte	ZiB	2^{70}	18%
yottabyte	YB	10^{24}	yobibyte	YiB	2^{80}	21%

IEEE (Institute of Electrical and Electronic Engineers)

The Bit

- A computer manipulates electrical signals
- Electrical signals in digital computers exist in two states – off and on
  - These are referred to as discrete values
  - Off is generally considered zero volts and on is some positive voltage, generally the power supply voltage
- Conveniently, we use the digits 0 and 1 to represent the two states off and on
- Thus, computers work in binary
- Each 0 and 1 is called a binary digit – **bit** for short

Bits to Bytes and Beyond

- Single bits allow you to represent two discrete values (states)
  - True or false
  - Yes or no
  - The number zero or the number 1

Logic perspective

- Representing more than two states requires using multiple bits
  - **Byte** = 8 bits
  - **Nybble** = 4 bits (half byte)
  - **Word** = 32 bits (four bytes)
  - **Halfword** = 16 bits (two bytes)
  - **Doubleword** = 64 bits (eight bytes)

Words and More…

- Word size is dependent on the system architecture
  - Older systems defined word = 16 bits
  - Current systems generally define word = 32 bits or 4 bytes (usually referred to as 32-bit systems)
  - 64-bit systems are typical today but a word could be either 32 bits or 64 bits – architecture dependent
  - **For us, a word is always a 32-bit value**
- Other definitions of data sizes
  - Quadword (usually 128 bits)
  - Octaword (usually 256 bits)
  - Not all systems have formal definitions for these data sizes

Data Representation

- It is very important to understand how information is represented in a computer system.
- Data representation is fundamental to understanding how and why computers work the way they do.
- Computers work in binary but humans work with languages and decimal-based numbers
  - words and phrases composed of alphabetic characters
- High-Level Programming
  - Variable types
  - Objects, records, composite data, etc.

It's All In The Numbers

- All information inside a computer is a number:
  - Alphabetic characters are numbers.
  - Sentences and phrases (character strings) are numbers.
  - Numbers in all forms are numbers.
  - Punctuation and special characters are numbers.
- How do we know what's what if everything is a number?
  - **It's how we use it that defines what it is!**
  - **Values "may" be used differently from how they were originally defined.**

Example: What Could A Number Be?

- The binary number

  00000000 01010010 10000010 10110011

- Could represent:
  - Integer value: 5,407,411
  - Character string: “ ⁰Ré| “
  - Real Number: 7.577397×10^{-39}
  - Instruction: add t0, t0, t0

Character Representations

- ASCII (American Standard Code for Information Interchange)
  - Original ASCII defined a 7-bit code for letters, numbers, punctuation & non-printables (aka control characters)
  - Numeric values 0 – 127 can be represented with 7 bits
  - Each letter (upper & lower case, numeric digit, punctuation was assigned a number
  - Extended ASCII adds additional characters by using an 8 bit numeric code (numeric values 0 – 255)
  - More than one version of extended ASCII
    - ISO 8859 (aka ISO Latin 1)
    - And others…

Character Representations (cont)

- EBCDIC (Extended Binary Coded Decimal Interchange Code)
  - 8-bit code for letters, numbers, punctuation
  - Implemented by IBM on first mainframe systems, still used and supported on mainframe class systems
  - Completely different code from ASCII – numbers represent different characters than those of ASCII
  - IBM was a proponent of ASCII standardization committee and wanted to adopt it for all their systems
  - However, they did not have enough time to adapt all their peripheral components for ASCII
  - So they stayed with EBCDIC and with the success of the System/360, EBCDIC also became successful

Character Representation (cont)

- Unicode
  - A computing industry standard for the consistent encoding, representation, and handling of text expressed in most of the world's writing systems
  - Provides a unique number for every character for every language regardless of platform
  - Uses a 7 – 32 bit encoding methodology
  - Most popular encoding formats:
    - UTF-8 used in Linux by default, and mostly data on Internet
    - UTF-16 used by Microsoft Windows, Mac OS X's file systems, Java programming language, etc.
  - First 127 encodings of Unicode incorporate ASCII codes
- Unicode has become the dominant scheme for internal processing and storage of text

What Do We Use For This Class?

- Unicode is the default on the systems we typically use
- For this class, we typically only need to work with the basic character set (what you see on the keyboard & some control characters)
  - ASCII character set
  - One byte is the basic unit for all character storage
  - Every individual character, numeric digit, special character, etc. requires a full 8 bits (1 byte)
- When working with character data in the hardware, the numerical representation is presented.

ASCII Chart

Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D	36	24	$	68	44	D	100	64	d
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f
7	7	Bell	BEL	CTRL-G	39	27	'	71	47	G	103	67	g
8	8	Backspace	BS	CTRL-H	40	28	(	72	48	H	104	68	h
9	9	Horizontal tab	HT	CTRL-I	41	29	)	73	49	I	105	69	i
10	0A	Line feed	LF	CTRL-J	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	VT	CTRL-K	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	FF	CTRL-L	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage feed	CR	CTRL-M	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	SO	CTRL-N	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	SI	CTRL-O	47	2F	/	79	4F	O	111	6F	o
16	10	Data line escape	DLE	CTRL-P	48	30	0	80	50	P	112	70	p
17	11	Device control 1	DC1	CTRL-Q	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	DC2	CTRL-R	50	32	2	82	52	R	114	72	r
19	13	Device control 3	DC3	CTRL-S	51	33	3	83	53	S	115	73	s
20	14	Device control 4	DC4	CTRL-T	52	34	4	84	54	T	116	74	t
21	15	Neg acknowledge	NAK	CTRL-U	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	SYN	CTRL-V	54	36	6	86	56	V	118	76	v
23	17	End of xmit block	ETB	CTRL-W	55	37	7	87	57	W	119	77	w
24	18	Cancel	CAN	CTRL-X	56	38	8	88	58	X	120	78	x
25	19	End of medium	EM	CTRL-Y	57	39	9	89	59	Y	121	79	y
26	1A	Substitute	SUB	CTRL-Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	ESC	CTRL-[	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	FS	CTRL-\	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	GS	CTRL-]	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	RS	CTRL-^	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	US	CTRL-_	63	3F	?	95	5F	_	127	7F	DEL

https://www.commfront.com/pages/ascii-chart

Every ASCII character has a number associated with it.

This table shows the decimal and hex value for the basic ASCII character set.

Note the first 32 characters are referred to as control characters.

The RARS program displays character data as hexadecimal values .

Number Representations

- First, let's just consider the counting numbers
  - Positive integers starting with zero and counting up
  - These are referred to as natural numbers (no negatives)
  - In computing, these are called **unsigned integers**
- Humans like decimal – base 10 (0 – 9)
- Computers work in binary – base 2 (0 – 1)
  - Each bit position in a binary number represents a power of 2 reading from right to left

. . .	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	(512)	(256)	(128)	(64)	(32)	(16)	(8)	(4)	(2)	(1)

- A 1 in a bit position adds to the value of the number based on its position

Binary Number Structure

- Identifying bit positions
  - LSB = least significant bit (right-most bit of number)
  - MSB = most significant bit (left-most bit of number)

- Byte:

$$\underset{\substack{2^7 \\ \text{MSB}}}{0}\ 1\ 0\ 1\ 0\ 0\ 1\ \underset{\substack{2^0 \\ \text{LSB}}}{1}$$

- Word:

$$\underset{\substack{2^{31} \\ \text{MSB}}}{0}\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ \ldots\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ \underset{\substack{2^0 \\ \text{LSB}}}{1}$$

Binary Place Values

- $2^0 = 1$
- $2^1 = 2$
- $2^2 = 4$
- $2^3 = 8$
- $2^4 = 16$
- $2^5 = 32$
- $2^6 = 64$
- $2^7 = 128$
- $2^8 = 256$
- $2^9 = 512$
- $2^{10} = 1024$

Counting in binary (using 4-bits):

Learn the binary counting sequence!

Binary		Decimal
0000	=	0
0001	=	1
0010	=	2
0011	=	3
0100	=	4
0101	=	5
0110	=	6
0111	=	7
1000	=	8
1001	=	9
1010	=	10
1011	=	11
1100	=	12
1101	=	13
1110	=	14
1111	=	15

Example – Binary to Decimal

- 8-bit binary number:

  0 1 1 0 0 1 1 1

  2^6 2^5 2^2 2^1 2^0

  - 64 + 32 + 4 + 2 + 1
  - = decimal 103

- 16-bit binary number:

  0 0 0 1 1 0 0 0 0 1 1 0 0 1 1 1

  2^{12} 2^{11} 2^6 2^5 2^2 2^1 2^0

  - 4096 + 2048 + 64 + 32 + 4 + 2 + 1
  - = decimal 6,247

Converting Decimal to Binary

- Successive divide by 2 until quotient = 0 and then read remainder values in reverse:
- Ex. 37 37 / 2 = 18 R 1

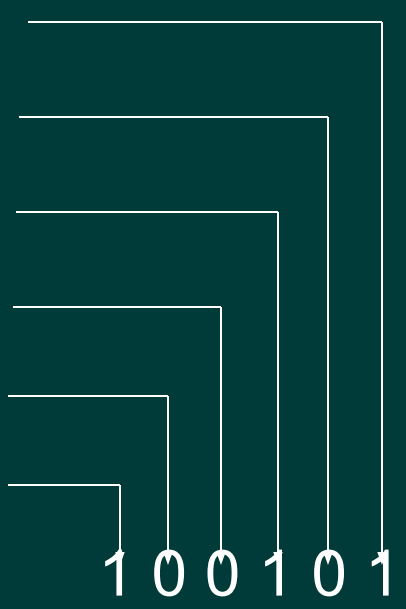
18 / 2 = 9 R 0

9 / 2 = 4 R 1

4 / 2 = 2 R 0

2 / 2 = 1 R 0

1 / 2 = 0 R 1



Binary value = 1 0 0 1 0 1

- **State binary values as byte (for 8 bits), halfword (for 16-bits) or word (for 32 bits)**
  - Place leading zeros in front: **0 0** 1 0 0 1 0 1

    for unsigned integers

Alternative Method by Subtraction

- Determine largest power of 2 value less than or equal the decimal number to convert
  - Ex. 37 (largest power of 2 value = 32 = 2^5)
- Subtract power of 2 value from number
  - 37 – 32 = 5
- Repeat by subtracting the largest power of 2
  - Largest power of 2 less than or equal to 5 = 4 (2^2)
  - Subtract: 5 – 4 = 1
- Continue subtracting largest power of 2 less than or equal to 1 = 1 (2^0)
  - 2^0 is the terminal value

		2^5	2^4	2^3	2^2	2^1	2^0	
		32	16	8	4	2	1	
0	0	1	0	0	1	0	1	$= 2^5 + 2^2 + 2^0 = 37$

Hexadecimal

- Large binary numbers are cumbersome for us to work with.
- Traditionally, hexadecimal (base 16) provides a convenient conversion
  - Hex digits 0 – 9, a – f representing values 0 through fifteen
  - Each hex digit represents a 4-bit nybble
  - 4-bits allows representing values 0 – 15
  - Two hex digits = 1 byte
  - Four hex digits = 1 halfword
  - Eight hex digits = 1 word

Numbers: Decimal, Binary, Hex

Decimal	Binary	Hex
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	a
11	1011	b
12	1100	c
13	1101	d
14	1110	e
15	1111	f

The alphabetic hex digits a – f may also be expressed capitalized.

The RARS program will always use lower case letters.

Hex values by convention should always be prefixed with 0x and stated as byte, halfword or word;

e.g. 0x01 (byte)
0x0001 (halfword)
0x00000001 (word)

Converting Decimal to Hexadecimal

- Successive divide by 16 and read remainder values in reverse:
- Ex. 37 37 / 16 = 2 R 5

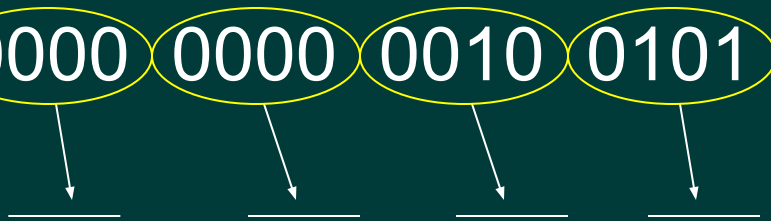
  2 / 16 = 0 R 2

  Hex value = 0x25 [(2 x $16^{1)}$ + (5 x 16^{0})]

  Binary value = 0010 0101
- **Values should be stated based on the appropriate storage size**
  - Byte: 0x25 (2 hex digits = 8 bits)
  - Halfword: 0x0025 (4 hex digits = 16 bits)
  - Word: 0x00000025 (8 hex digits = 32 bits)

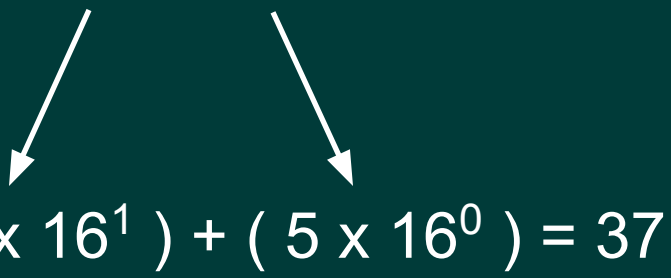
Hexadecimal Notation

- Remember, hex numbers are preceded by **0x**
- 0x0025 = binary 0000 0000 0010 0101 = 37_{10}



Hexadecimal to decimal by nybble:

0000 = 0	1000 = 8
0001 = 1	1001 = 9
0010 = 2	1010 = a
0011 = 3	1011 = b
0100 = 4	1100 = c
0101 = 5	1101 = d
0110 = 6	1110 = e
0111 = 7	1111 = f



You Try It

Convert 101_{10} to binary stated as a byte value

101 / 2 = 50 R 1
50 / 2 = 25 R 0
25 / 2 = 12 R 1
12 / 2 = 6 R 0
6 / 2 = 3 R 0
3 / 2 = 1 R 1
1 / 2 = 0 R 1

0 1 1 0 0 1 0 1

101	
64	(2^6)
37	
32	(2^5)
5	
4	(2^2)
1	(2^0)

Restate the byte value as hexadecimal

0 1 1 0 0 1 0 1

6 5 → 0x65

Positive & Negative Integers
(Signed Integers)

- Must represent both positive (+) and negative (-)
  - In binary, can't use +/- symbols; only have 0s and 1s
- All digital systems use a format called **two's complement**
  - Leftmost bit represents the most negative value you can have for the number of bits
- Example using byte value

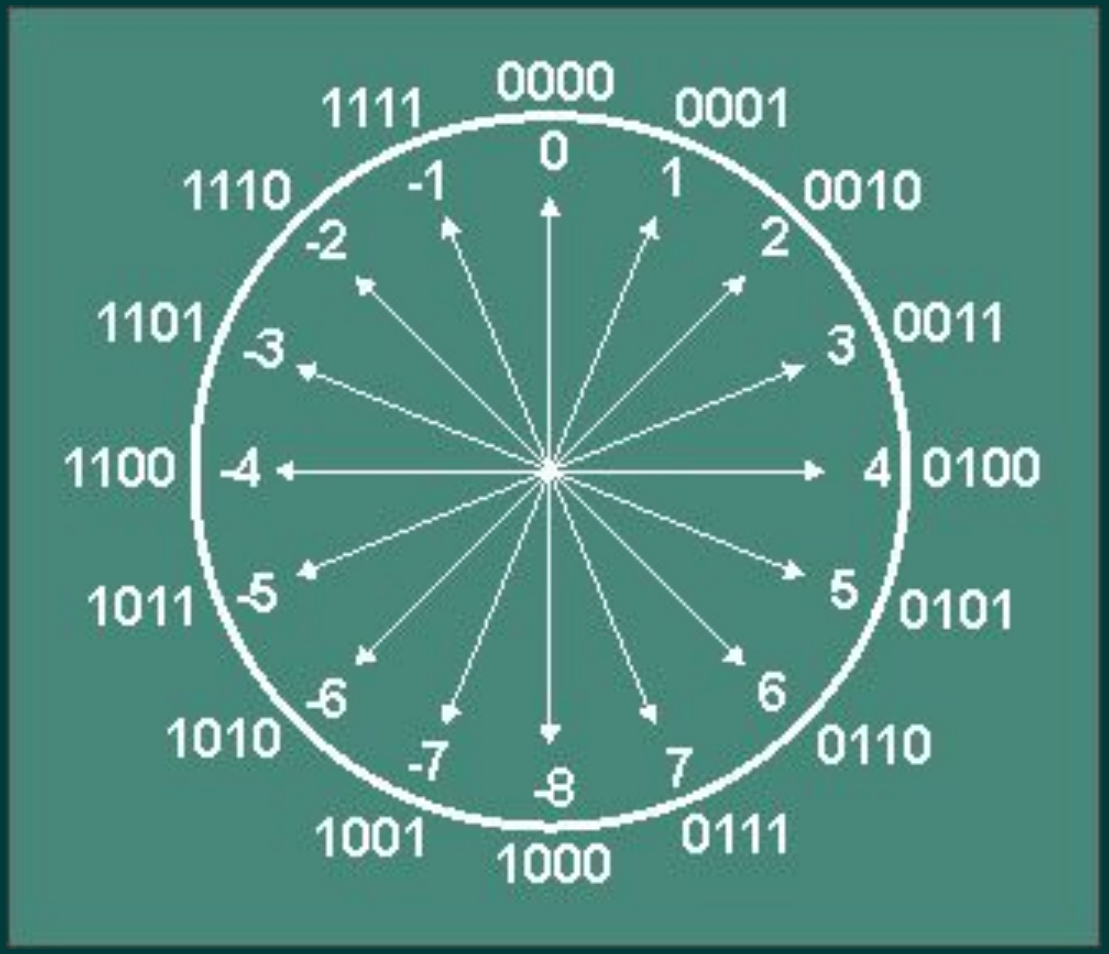
1	0	0	1	0	0	1	1
-2^7			2^4			2^1	2^0

$-128 + 16 + 2 + 1 = -109_{10}$

Signed integers are the default and most commonly used integer types in general computing.

Signed Integers

- Range of values for signed integers centers around zero



-8 = 1000
-7 = 1001
-6 = 1010
-5 = 1011
-4 = 1100
-3 = 1101
-2 = 1110
-1 = 1111
0 = 0000
1 = 0001
2 = 0010
3 = 0011
4 = 0100
5 = 0101
6 = 0110
7 = 0111

Note: the 2's complement of zero = zero!
The 2's complement of the most negative number is itself.

Positive to Negative and Back Again

- To form the negative value (if given the positive),
  - Invert (complement) all the bits of the positive number
  - Then add 1
  - Example: +25 = 00011001 (represented as byte)
    - Invert bits: 11100110 (0s to 1s; 1s to 0s)
    - Add 1: 11100111 (this value is -25)
- To get back to positive, it's the same process
  - Invert all bits
  - Add 1
- How to tell if a binary number is positive or negative
  - The leftmost bit (most significant bit) = 0 for positive integers
  - The leftmost bit = 1 for negative integers

Another Example

- Binary 00101100
  - First bit = 0, so it's a positive number
  - Value = 44_{10} ($2^5 + 2^3 + 2^2$)
  - Hex = 0x2c
- Compute negative value
  - Complement binary bits: 11010011
  - Add 1: + 1
  - Value: -44 ($-2^7 + 2^6 + 2^4 + 2^2$) 11010100
  - Hex = 0xd4

11 (carries)

- Convert back to positive
  - Complement binary bits: 00101011
  - Add 1: + 1
  - Value: +44 00101100
  - Hex = 0x2c

11 (carries)

Another Example

- Binary 11100111
  - First bit = 1, so it's a negative number
  - Value = -25_{10} ($-2^7 + 2^6 + 2^5 + 2^2 + 2^1 + 2^0$)
  - Hex = 0xe7
- Compute positive value
  - Complement binary bits: 00011000
  - Add 1: + 1
  - Value: +25 ($2^4 + 2^3 + 2^0$) 00011001
  - Hex = 0x19
- Convert back to negative
  - Complement binary bits: 11100110
  - Add 1: + 1
  - Value: -25 11100111
  - Hex = 0xe7

Negative Numbers (cont)

- **Sign extension** – replicating the sign bit to the left when a number uses more bits
- Example: -25
  - As 1 byte (8 bit) value: 11100111
  - As 1 word (4 byte) value: 11111111111111111111111111100111
  - Must keep the same value regardless of the number of bits
- For positive numbers, since the sign bit is 0, the zero would be replicated to the left
- Example: +25
  - As 1 byte (8 bit) value: 00011001
  - As 1 word (4 byte) value: 00000000000000000000000000011001
- Why use two's complement?
  - Makes the digital hardware for arithmetic logic circuits easier to construct

Numeric Strings

- Textual representation of numbers
- Often the way stored in document or text files or during input/output operations
- Example: “1234” ≠ 1234
  - Binary string: 00110001001100100011001100110100
  - Binary integer: 00000000000000000000010011010010
- Numeric strings that will be used in arithmetic operations must be converted to integer representation
  - HLLs (Java, Python, C, etc.) have library functions or methods to convert
  - Low-level programming (assembly language) will require you code the conversion without library support

Numeric String to Integer Conversion Algorithm

1. Initialize a starting result value to zero
2. Loop over the characters in the string, left to right
3. Convert the ASCII character to its equivalent digit
    1. ASCII number – ASCII zero (can use either decimal or hex)
4. Compute: result value = result value * 10 + digit
5. Repeat steps 3 & 4 for all remaining characters
6. Result value will contain the correct converted integer value

This algorithm is an example of when it is necessary to perform arithmetic on numbers that represent character data.

Example Conversion

- String: “1234”
- Initial result value = 0
- First character to digit: 49 - 48 = 1 (0x31 – 0x30 = 1)
- Result value = 0 * 10 + 1 = 1
- Second character to digit: 50 – 48 = 2 (0x32 – 0x30 = 2)
- Result value = 1 * 10 + 2 = 12
- Third character to digit: 51 – 48 = 3 (0x33 – 0x30 = 3)
- Result value = 12 * 10 + 3 = 123
- Fourth character to digit: 52 – 48 = 4 (0x34 – 0x30 = 4)
- Result value = 123 * 10 + 4 = 1234
- Done!

Other Numerical Types

- Numbers with a fractional component (real numbers); e.g. 123.456
  - Potentially infinite number of values
  - How to represent infinite possibilities with finite discrete 0s and 1s?
  - Special and somewhat complex encoding format allows representation of numbers with integer and fractional components within defined limits.
  - Coverage of this topic is included in Chapter 3 which covers computer arithmetic and additional rules and restrictions for dealing with numeric data.
  - We will defer discussing this topic until later in the semester.

Terminology to Know

- bit
- nybble
- byte
- word
- word size
- Halfword
- doubleword
- ASCII
- EBCDIC
- Unicode
- binary number
- hexadecimal number
- binary/decimal/hex number conversion
- signed integers
- unsigned integers
- sign extension
- two’s complement
- units of measurement
- Numerical strings & conversion to integer