DATA REPRESENTATION
PART 3: CHARACTERS AND BOOLEAN

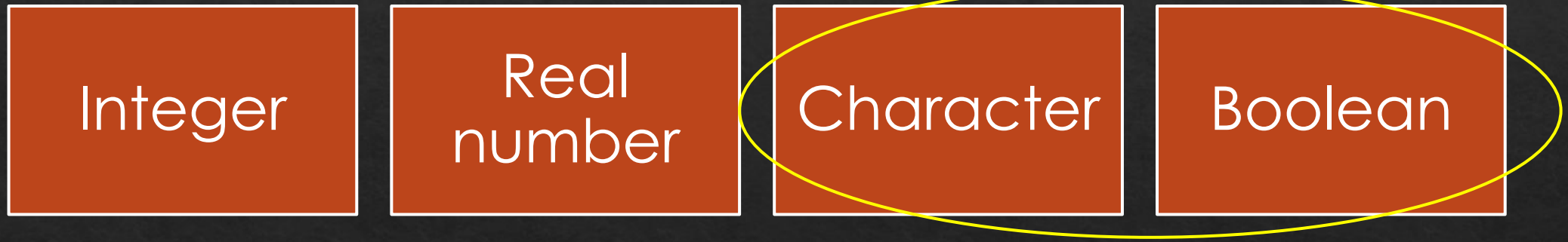
CCICOMP

OVERVIEW

- Represent characters and strings using coding methods
- Perform single-bit and bitwise logic operations on Boolean data

RECALL: CPU DATA TYPES

- The CPUs of most modern computers can represent and process at least the following primitive data types:



- The arrangement and interpretation of bits are usually different for each data type.
- Format for each data type balances compactness, range, accuracy, ease of manipulation, and standardization.

CHARACTERS

- English and many other languages use alphabetic letters, numerals, punctuation marks, and a variety of other special-purpose symbols, such as $ and &. (each symbol is a character)
- Character data can't be represented directly in a computer because computers process only bits; so such data is represented indirectly by defining a table that assigns a numeric value to each character.

CHARACTERS

- **Coding methods** for characters use table-based substitution of one set of symbols or values
- All coding methods share these important characteristics
  - All users must use the same coding and decoding methods.
  - The coded values must be capable of being stored or transmitted.
  - A coding method represents a tradeoff among compactness, number of representable characters, and standardization

COMMON CODING METHODS

- **ASCII (American Standard Code for Information Interchange)**
  - 7-bit encoding format, with 1 bit for parity. Newer computers may use all 8-bits
  - Coding table has at least 128 available characters
  - Additional 128 characters (If 8-bit version) is used by manufacturers to define multinational or graphical characters
  - Supported by majority of OS

COMMON CODING METHODS

- **Unicode**
  - Coding table that uses 16-bit or 32-bit codes to represent characters
  - Created to extend the range limit of 8-bit coding methods to represent characters of different languages (e.g. Chinese, Arabic, Japanese etc)
  - Currently, ASCII is designated as a subset of Unicode.

ASCII CODING METHOD

Hex	Value	Hex	Value	Hex	Value	Hex	Value	Hex	Value	Hex	Value	Hex	Value	Hex	Value
00	NUL	10	DLE	20	SP	30	0	40	@	50	P	60	`	70	p
01	SOH	11	DC1	21	!	31	1	41	A	51	Q	61	a	71	q
02	STX	12	DC2	22	"	32	2	42	B	52	R	62	b	72	r
03	ETX	13	DC3	23	#	33	3	43	C	53	S	63	c	73	s
04	EOT	14	DC4	24	$	34	4	44	D	54	T	64	d	74	t
05	ENQ	15	NAK	25	%	35	5	45	E	55	U	65	e	75	u
06	ACK	16	SYN	26	&	36	6	46	F	56	V	66	f	76	v
07	BEL	17	ETB	27	'	37	7	47	G	57	W	67	g	77	w
08	BS	18	CAN	28	(	38	8	48	H	58	X	68	h	78	x
09	HT	19	EM	29	)	39	9	49	I	59	Y	69	i	79	y
0A	LF	1A	SUB	2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z
0B	VT	1B	ESC	2B	+	3B	;	4B	K	5B	[	6B	k	7B	{
0C	FF	1C	FS	2C	,	3C	<	4C	L	5C	\	6C	l	7C	|
0D	CR	1D	GS	2D	-	3D	=	4D	M	5D	]	6D	m	7D	}
0E	SO	1E	RS	2E	.	3E	>	4E	N	5E	^	6E	n	7E	~
0F	SI	1F	US	2F	/	3F	?	4F	O	5F	_	6F	o	7F	DEL

DEVICE CONTROL

- When text is printed or displayed on an output device such as a monitor or printer, it's often formatted in a particular way (e.g. lines and paragraphs)
- Coding methods define device **control codes** used for text formatting by placing them immediately before or after the characters they modify.
- Common examples are:
  - Carriage return (CR) moves the print head or insertion point to the beginning of a line
  - Line feed (LF) which moves the print head or insertion point down one line

Hex	Value	Hex	Value
00	NUL	10	DLE
01	SOH	11	DC1
02	STX	12	DC2
03	ETX	13	DC3
04	EOT	14	DC4
05	ENQ	15	NAK
06	ACK	16	SYN
07	BEL	17	ETB
08	BS	18	CAN
09	HT	19	EM
0A	LF	1A	SUB
0B	VT	1B	ESC
0C	FF	1C	FS
0D	CR	1D	GS
0E	SO	1E	RS
0F	SI	1F	US

STRINGS

- Strings are complex data types that are composed of a series of character codes
- Example: The string “Hello!” encoded using ASCII is stored as

ASCII Code	**48**	**65**	**6C**	**6C**	**6F**	**21**
Equivalent character	‘H’	‘e’	‘l’	‘l’	‘o’	‘!’

EXAMPLE

What is the ASCII interpretation the hex value 434349434F4D50 as character data?

43	43	49	43	4F	4D	50
'C'	'C'	'I'	'C'	'O'	'M'	'P'

Hex	Value	Hex	Value	Hex	Value	Hex	Value	Hex	Value	Hex	Value	Hex	Value	Hex	Value
00	NUL	10	DLE	20	SP	30	0	40	@	50	P	60	`	70	p
01	SOH	11	DC1	21	!	31	1	41	A	51	Q	61	a	71	q
02	STX	12	DC2	22	"	32	2	42	B	52	R	62	b	72	r
03	ETX	13	DC3	23	#	33	3	43	C	53	S	63	c	73	s
04	EOT	14	DC4	24	$	34	4	44	D	54	T	64	d	74	t
05	ENQ	15	NAK	25	%	35	5	45	E	55	U	65	e	75	u
06	ACK	16	SYN	26	&	36	6	46	F	56	V	66	f	76	v
07	BEL	17	ETB	27	'	37	7	47	G	57	W	67	g	77	w
08	BS	18	CAN	28	(	38	8	48	H	58	X	68	h	78	x
09	HT	19	EM	29	)	39	9	49	I	59	Y	69	i	79	y
0A	LF	1A	SUB	2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z
0B	VT	1B	ESC	2B	+	3B	;	4B	K	5B	[	6B	k	7B	{
0C	FF	1C	FS	2C	,	3C	<	4C	L	5C	\	6C	l	7C	|
0D	CR	1D	GS	2D	-	3D	=	4D	M	5D	]	6D	m	7D	}
0E	SO	1E	RS	2E	.	3E	>	4E	N	5E	^	6E	n	7E	~
0F	SI	1F	US	2F	/	3F	?	4F	O	5F	_	6F	o	7F	DEL

OPERATIONS ON CHARACTERS

- Characters are usually handled similarly to unsigned integers by the CPU
- Integer instructions (add, copy, compare equal) often have similar effect to unsigned integers and characters
  - ‘A’ (ASCII 41h) + 1 = ‘B’ (ASCII 42h)
  - ‘C’ (ASCII 43h) == ‘C’ (ASCII 43h)
- Nonequality comparisons compare the characters numeric codes and hence are dependent on their **collating sequence**
  - ‘A’ (ASCII 41h) != ‘a’ (ASCII 61h)
  - ‘a’ (ASCII 61h) > ‘B’ (ASCII 42h) ‘apple’ comes after ‘Boy’ when sorting alphabetically

BOOLEAN DATA

- **Boolean data type** has only two data values—true and false
- When the processing function is a comparison operation, the output signal represents a Boolean result which is often used by other instructions as input (e.g. a conditional branch in a program).
- The Boolean data type requires only a single bit for representation, but to simplify processor design and implementation, most CPU designers use an **integer coding format** to represent Boolean values.
  - Integer value zero corresponds = false
  - Any nonzero value = true.

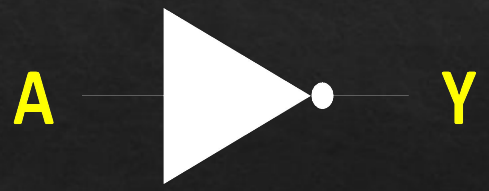
LOGIC OPERATIONS AND GATES

- Logic operations based on **Boolean algebra** may be used on Boolean data to produce a result of either ‘true’ (1) or ‘false’ (0).
- Logic operations form the basis of **logic gates** - circuits serving as basic building blocks of CPU that can perform a processing function on a single binary electrical signal, or bit
- Processing circuits for more complex operations are constructed by combining logic gates to perform Boolean functions

NOT OPERATION

A NOT operation negates the input expression and is also referred to as an inverter.

Logic Symbol



Boolean Expression

$$Y = A'$$

or

$$Y = \overline{A}$$

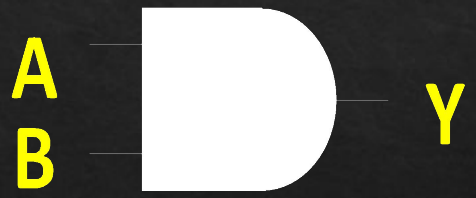
Truth Table

A	Y
0	1
1	0

AND OPERATION

The AND operation is also known as an "all or nothing" gate since it will only output a true condition if **ALL** its inputs are true

Logic Symbol



Boolean Expression

$$Y = AB$$

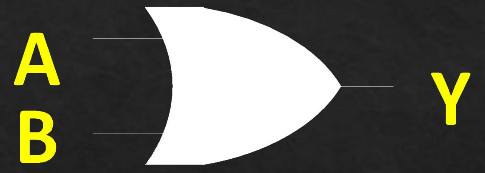
Truth Table

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

OR OPERATION

The OR operation is often referred to as an "any or all" gate. It outputs a true condition if at least one of its conditions is true.

Logic Symbol



Boolean Expression

$$Y = A + B$$

Truth Table

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

XOR OPERATION

The XOR operation is also known as exclusive-OR. It outputs a true condition if the input variables have an odd number of binary 1's (odd function).

Logic Symbol

Boolean Expression

$$Y = A \oplus B$$

or

$$Y = A'B + AB'$$

Truth Table

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

BITWISE OPERATIONS

- On computers, logic operators are often implemented as bitwise operations which perform logic operations on individual bits of a bit string.
- Examples:

NOT F3h = ?

NOT 1111 0011b

0000 1100b

= 0Ch

F3h AND 85h = ?

1111 0011b

AND 1000 0101b

1000 0001b

= 81h

BITWISE OPERATIONS

- On computers, logic operators are often implemented as bitwise operations which perform logic operations on individual bits of a bit string.
- Examples:

F3h OR 85h = ?

```
    1111 0011b
OR  1000 0101b
-------------
    1111 0111b
=   F7h
```

F3h XOR 85h = ?

```
    1111 0011b
XOR 1000 0101b
-------------
    0111 0110b
=   76h
```

APPLICATION OF BITWISE OPERATIONS

Example 1: Checking if a number is odd

23h = 0010 0011b
23h AND 01h
0010 0011
AND 0000 0001
0000 0001 □ odd (true)

22h = 0010 0010b
22h AND 01h
0010 0010
AND 0000 0001
0000 0000 □ even (false)

Example 2: Flipping bits (e.g. for getting 1's / 2's complement)

23h = 0010 0011

NOT 23h ==> 1101 1100

BOOLEAN FUNCTIONS

- A Boolean function is an expression formed with binary variables, binary or unary operators, parenthesis, and equal sign. The result of the function can be either 0 or 1.
- Example:
- $$F = \bar{X}\bar{Y}Z + X\bar{Y}\bar{Z} + \bar{X}YZ + X\bar{Y}Z$$
- Complex binary operations can be expressed as Boolean functions which can later be used as basis to design the processing circuits of a CPU

EXAMPLE: SINGLE-BIT ADDER

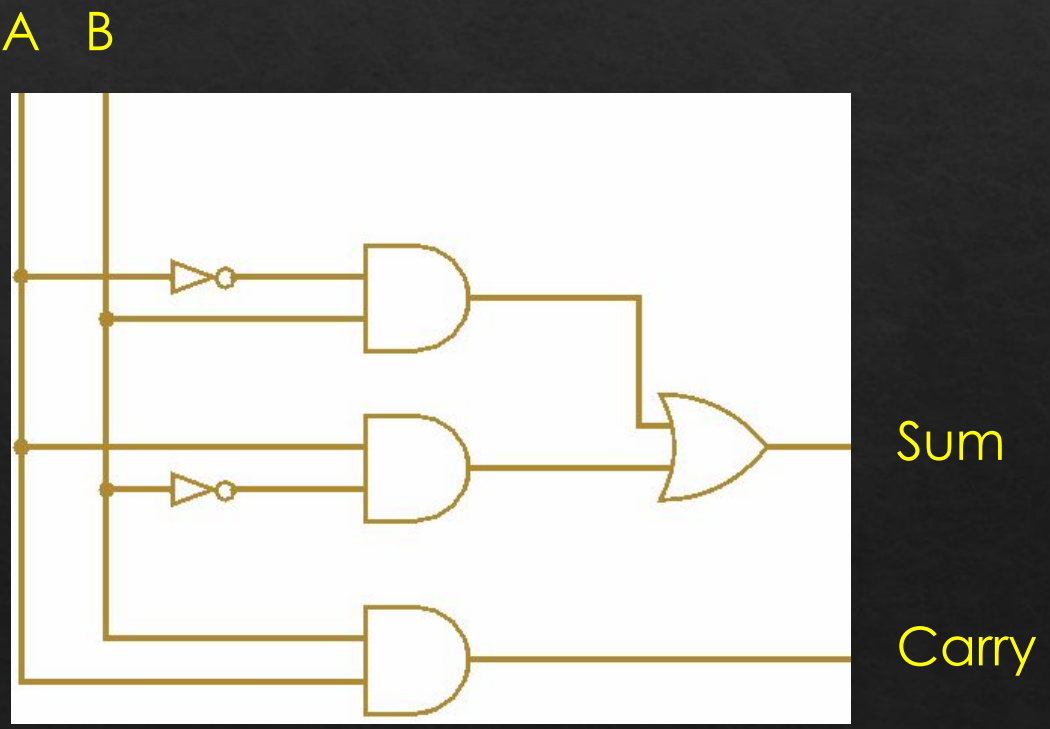
Recall: Binary addition

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Boolean functions

Sum = A'B + AB'
Carry = AB

Logic Circuit



SUMMARY

- Characters are represented using coding methods which assign a binary equivalent value or each character
- Strings are a complex data type composed of multiple characters
- Boolean data types have only two data values—true and false. On computers a zero value represents Boolean a ‘false’ and any non-zero value is considered a Boolean ‘true’.
- Boolean logic operations may be used to process Boolean data; and they form the basis for the design of CPU circuity that perform complex processing operations

KEY TAKEAWAYS

- All primitive data types are just binary values
- Different data types follow specific representation formats in order to be converted into a binary value
- What value a piece of data means is all just based on how it is interpreted following a specific representation format
- Primitive types can be put together to form complex data types