

mjqqt

# Caesar Cipher

- A simple way to encode a “message”
- CC-5 [  $D(x) = (x - n) \bmod 26$ , where  $n = 5$  ]

plain	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
cipher	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e

m	j	q	q	t

[illegible]

# Introduction to Encodings

- Today's Plan
  - Communication and Bits
  - Discussions on Functions and Mappings
  - Binary Strings and Fields: IPv4 and MIPS Instructions
  - Fixed-length Binary Encodings
    - 3 bits: Octal Encoding
    - 4 bits: Hexadecimal Encoding
    - 5 bits: MIPS Register Encoding
    - 6 bits: Base64 Encoding, MIPS Operations and Functions Encoding
    - 8 bits: ASCII (text) encoding
  - Variable Length Instructions: UTF-8

# Recall: OSI and TCP/IP

Layer	Name	Example Protocol	Naming	Transported	Hardware Device
7	Application	http	url	data	
6	Presentation	---			
5	Session	---			
4	Transport	TCP/IP	socket	segment	
3	Network / Internet	IPv4 IPv6	IP	packet	router
2	Data Link / Link	Ethernet	MAC	frame	switch
1	Physical	802.11g	Interface	symbols	hub, bridge

Host layers



Media layers

Comm

## Payload

## Layer 2

Layer 1

Preamble	Start frame delimiter	MAC destination	MAC source	802.1Q tag (optional)	Ethertype (Ethernet II) or length (IEEE 802.3)	Payload	Frame check sequence (32-bit CRC)	Interpacket gap
7 octets	1 octet	6 octets	6 octets	(4 octets)	2 octets	46-1500 octets	4 octets	12 octets

### Payload Header: Layer 3

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				IHL				DSCP				ECN				Total Length															
4	32	Identification																Flags				Fragment Offset											
8	64	Time To Live								Protocol								Header Checksum															
12	96	Source IP Address																															
16	128	Destination IP Address																															
20	160	Options (if IHL > 5)																															
:	:																																
60	480																																


## Decoding the Message (chunk the bits into fields)

[illegible]

- **Inter-Packet Gap, Preamble, and Start of Frame:**
  - 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
  - 10101010 10101010 10101010 10101010 10101010 10101010 10101010
  - 10101011
- **Mac Address: 3c:06:30:40:2d:8e**
  - 0011 1100 : 0000 0110 : 0011 0000 : 0100 0000 : 0010 1101 : 1000 1110
  - 0011 1100 : 0000 0110 : 0011 0000 : 0100 0000 : 0010 1101 : 1000 1110
- **Length: 0000 0000 0000 0000**
- **Payload:**
- **CRC: xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx**
- **Inter-Packet Gap**
  - 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
  - 00000000 00000000 00000000

# Types of Encodings

- String of bits:
  - Inter-Packet Gap, Preamble, Start of Frame
- Binary:
  - Flags: Reserved, Don't Fragment, More Fragments
- Integer:
  - Length, Version Number, TTL, etc.
- Index:
  - Protocol Lookup Table
- MAC Address: 3c:06:30:40:2d:8e
  - 0011 1100 : 0000 0100 : 0011 000 : 0100 0000 : 0010 1101 : 1000 1110
  - Hexadecimal -> Binary
- IP Address: www.csun.edu
  - Dotted Decimal Notation: 130.166.238.19
  - 1000 0010 . 1010 0110 . 1110 1110 . 0001 0011
  - Decimal -> Binary
- Data: (follows the IPv4 header)
  - text, images, video, audio, colors, etc.



Protocol Number	Protocol Name	Abbreviation
1	<a href="#">Internet Control Message Protocol</a>	ICMP
2	<a href="#">Internet Group Management Protocol</a>	IGMP
6	<a href="#">Transmission Control Protocol</a>	TCP
17	<a href="#">User Datagram Protocol</a>	UDP
41	<a href="#">IPv6 encapsulation</a>	ENCAP
89	<a href="#">Open Shortest Path First</a>	OSPF
132	<a href="#">Stream Control Transmission Protocol</a>	SCTP



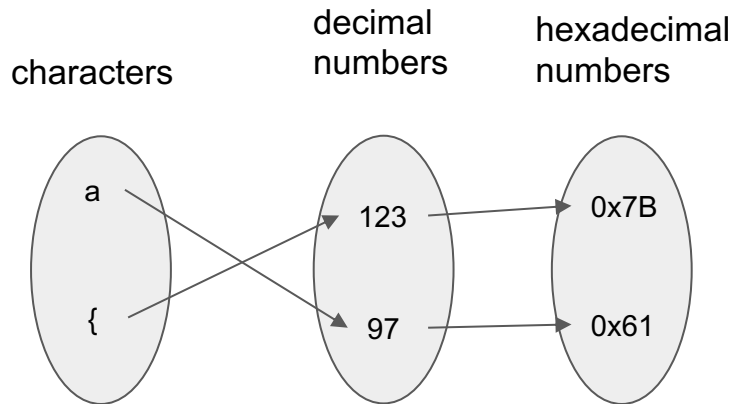
# Mappings and Functions

- Mapping: defines a relationship
- Function: a binary relation between two sets
  - Encode: input -> output
  - Decode: output -> input
- A table can represent a function

INPUT	OUTPUT 1	OUTPUT 2
5	8	53
2	5	50
4	7	52
9	?	?
h	? ■	104 ? ■■

0  
1  
2  
3  
4  
5  
6  
7  
8  
9

a  
b  
c  
d  
e  
f  
g  
h  
i  
j  
k  
l



# Encoding Examples:

- Fixed Length Lookup Tables:

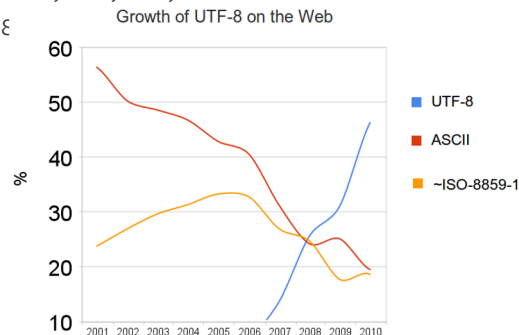
○ Octal (3 bit chunks):	e.g., 0100 0011 0110 0001 0111 0100	020660564	010 000 110 110 000 101 110 100
○ Binary Coded Decimal (4 bit chunks):		436,174	0100 0011 0110 0001 0111 0100
○ Hexadecimal (4 bit chunks) :		0x436174	0100 0011 0110 0001 0111 0100
○ Base64 (6 bit chunks):		Q2F0	010000 110110 000101 110100
○ ASCII (8 bit chunks):		Cat	01000011 01100001 01110100
○ MIPS Instruction (32 bit chunks):		add \$t0,\$t1,\$t2	0000 0001 0100 1011 0100 0000 0010 0000

- Various Lengths: (function used to perform the mapping)

○ short int (16 bit chunks):	-32,768 ... 32,767
○ int (32 bit chunks):	-2,147,483,648 ... 2,147,483,647
○ long int (64 bit chunks):	-9,223,372,036,854,775,807 ... 9,223,372,036,854,775,807

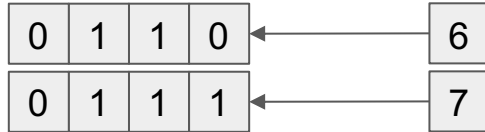
- Variable Length:

- UTF-8
  - Unicode Transformation Format
  - 1 byte to 4 bytes used to encode each character



# BCD: Binary Coded Decimal

- Encoding of: 6 & 7

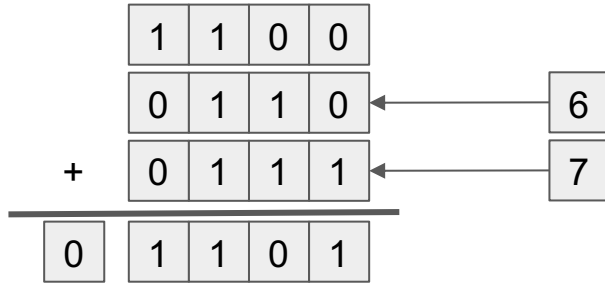


- An encoding for numbers, where precision is required
- Four bits are used to encode each digit
- Addition is performed on each 4-bit chunk (nibble)

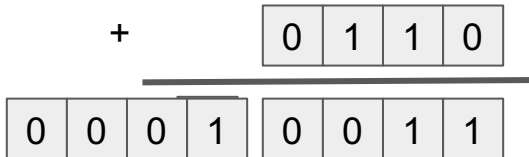
N	Code	N	Code
0	0000	8	1000
1	0001	9	1001
2	0010		1010
3	0011		1011
4	0100		1100
5	0101		1101
6	0110		1110
7	0111		1111

# BCD: Addition

- Addition performed on the nibble level: 6+7



if (overflow or invalid code ) then

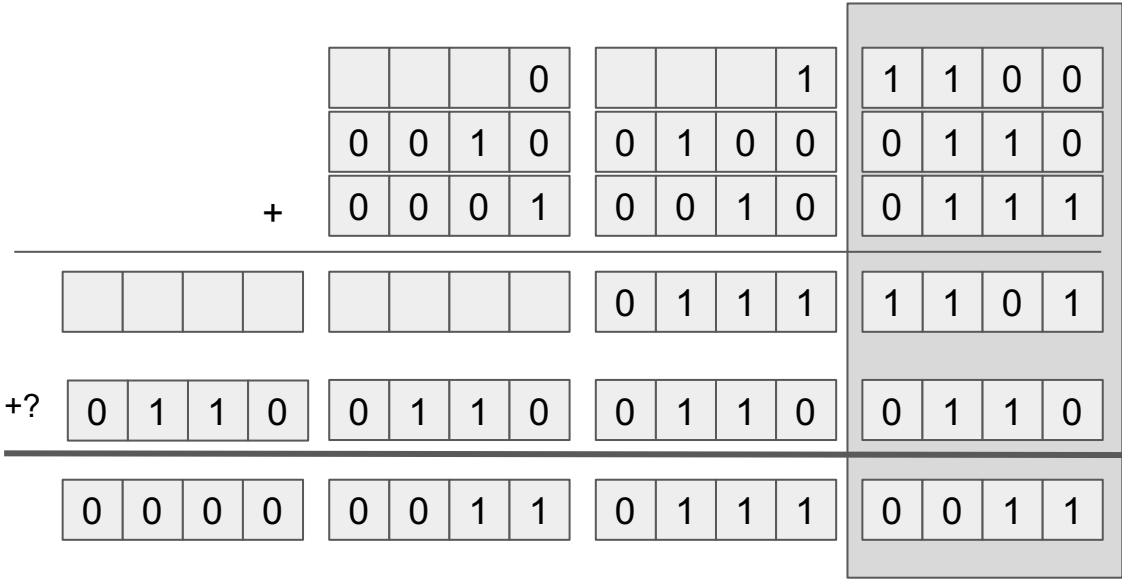


N	Code	N	Code
0	0000	8	1000
1	0001	9	1001
2	0010		1010
3	0011		1011
4	0100		1100
5	0101		1101
6	0110		1110
7	0111		1111

Diagram illustrating the BCD addition of 6 and 7, showing the result 13 (00100110) and the correction step (adding 0110) to produce the final result 00010011.

The correction step is labeled "jump 6" and is indicated by a curved arrow from the "6" row to the "0110" row.

# BCD: Addition Example: 246 + 127 = 373



N	Code	N	Code
0	0000	8	1000
1	0001	9	1001
2	0010		1010
3	0011		1011
4	0100		1100
5	0101		1101
6	0110		1110
7	0111		1111

# Instruction Encoding: MIPS

- (6 bits) The operation to be performed ([MIPS Encoding](#))

- It also indicates the encoding format to be used!
- There are three primary formats: R, I, and J.

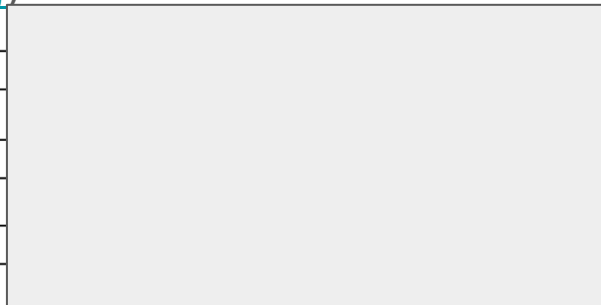
R      6 bits  
        op

I      6 bits  
        op

J      6 bits  
        op

- Other fields determine

- (5 bits) which registers are used ([Register Encoding](#))
  - rs: first source register
  - rt: second register
  - rd: destination register
- (5 bits) the amount a value is shifted (range: 0 .. 31)
- (6 bits) the mathematical function to be performed ([MIPS Encoding](#))
- (16 bits) the immediate value (range: -2048 .. 2047)
- (26 bits) the address / 4

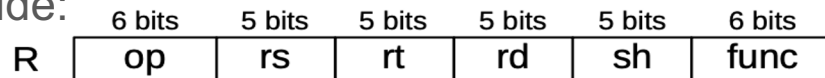


# Instruction Encoding: MIPS

- Three primary instruction encodings include:

- R-type (register)

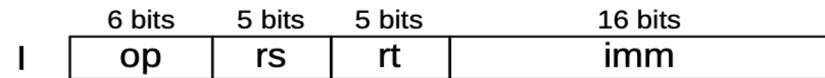
- for instructions using only registers
    - example: 0x014b4020
    - for: add \$t0, \$t1, \$t2



(2# 0000 0001 0100 1011 0100 0000 0010 0000)  
(\$t0 = \$t1 + \$t2)

- I-type (immediate)

- for instructions with immediate values:
    - example: 0x21280005
    - for: addi \$t0, \$t1, 5

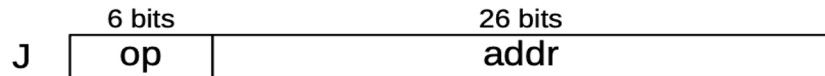


(2# )

(\$t0 = \$t1 + 5)

- J-type (jump)

- for instructions that perform unconditional jumps
    - example: 0x0810000
    - for: j label



(2# )

# goto label = 0x00400000

# An Encoding for the Keyboard

- Look at your keyboard.
  - a-z, A-Z, 0-9, !@#\$%^&\*()\_+-~`.,/<>?;:'"[]\}|
  - don't forget: space, tab, return, and delete key
  - plus we need other stuff:
  - All total, we we have 128 things to encode ( $2^n \leq 128$ , what is the value of  $n$ ? 7)
- We need to devise an encoding that maps everything to numbers
- How many bits do we need? How many things do we bits in a byte?
- An example of a fixed-width encoding!
- Let's build a table! [Keyboard Encodings](#)
- ASCII, abbreviated from American Standard Code for Information Interchange, is a character encoding standard for electronic communication.



# Parity Bit (or Check Bit)

- We are only using 7 of the 8 bits, what shall we do with it.

7 bits of data	(count of 1-bits)	8 bits including parity	
		even	odd
0000000	0	0000000 <b>0</b>	0000000 <b>1</b>
1010001	3	1010001 <b>1</b>	1010001 <b>0</b>
1101001	4	1101001 <b>0</b>	1101001 <b>1</b>
1111111	7	1111111 <b>1</b>	1111111 <b>0</b>

- Algorithm (odd)
  - a. count the number of 1's
  - b. add a 1 to make odd
  - c. transmit
  - d. receive
  - e. count the number of 1's
  - f. if even, ask for the data to be resent
- Checksum:
  - \* performs integrity checking at an aggregate level
  - \* reliability of networks have greatly improved since way back when!

A: 

S: |

# Extended ASCII and UTF-8 (unicode)

- We could use that bit to encode more stuff: 0..255
- But we have even more stuff. Let's use 16 bits to encode: 0..64K
- But now we have doubled what we need to send..
- Enter variable-length encoding.
  - Send only a byte for the most common symbols
  - Use the MSB to indicate a variable length encoding
- UTF-8: encodes  $>2,000,000$  ( $2^{21}$ ) values, using a maximum of 4 bytes
- Defines four type of bytes:
  - ASCII byte: begins with a 0 (1-byte indicator)
  - Continuation byte: begins with a 10
  - 2-byte Indicator: begins with a 110
  - 3-byte Indicator: begins with a 1110
  - 4-byte Indicator: begins with a 11110

# Extended ASCII and UTF-8

- The list of [UTF-8 characters](#):
- Layout of the bits:
- Example on how to lay it out:

**Layout of UTF-8 byte sequences**

Number of bytes	First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
1	U+0000	U+007F	0xxxxxxx			
2	U+0080	U+07FF	110xxxxx	10xxxxxx		
3	U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
4	U+10000	<sup>[nb 3]</sup> U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx