

# ***Number Codes and Registers***

# Overview

- **2's complement numbers**
  - Addition and subtraction
- **Binary coded decimal**
- **Gray codes for binary numbers**
- **ASCII characters**
- **Moving towards hardware**
  - Storing data
  - Processing data



AB5F

## 2's Complement Subtraction

- Let's compute  $(13)_{10} - (5)_{10}$ .

- $(13)_{10} = +(1101)_2 = (01101)_2$

- $(-5)_{10} = -(0101)_2 = (11011)_2$

- Adding these two 5-bit codes...

				0	1	1	0	1
	+			1	1	0	1	1
				-----				
carry				1	0	1	0	0

- Discarding the carry bit, the sign bit is seen to be zero, indicating a correct result.
- Numbers in hexadecimal

## 2's Complement Subtraction

- Let's compute  $(5)_{10} - (12)_{10}$ .
  - $(-12)_{10} = -(1100)_2 = (10100)_2$
  - $(5)_{10} = +(0101)_2 = (00101)_2$
- Adding these two 5-bit codes...

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

- Here, there is no carry bit and the sign bit is 1. This indicates a negative result, which is what we expect.  $(11001)_2 = -(7)_{10}$ .
- Numbers in hexadecimal

# Binary Coded Decimal

Digit	BCD Code	Digit	BCD Code
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

- Binary coded decimal (BCD) represents each decimal digit with four bits
  - Ex.  $\underline{0011} \underline{0010} \underline{1001} = 329_{10}$   
3      2      9
- This is NOT the same as  $001100101001_2$
- **Why do this?** Because people think in decimal.

# Putting It All Together

Decimal	Binary	Octal	Hexadecimal	BCD
0	0	0	0	0000
1	1	1	1	0001
2	10	2	2	0010
3	11	3	3	0011
4	100	4	4	0100
5	101	5	5	0101
6	110	6	6	0110
7	111	7	7	0111
8	1000	10	8	1000
9	1001	11	9	1001
10	1010	12	A	0001 0000
11	1011	13	B	0001 0001
12	1100	14	C	0001 0010
13	1101	15	D	0001 0011
14	1110	16	E	0001 0100
15	1111	17	F	0001 0101

- **BCD not very efficient**
- **Used in early computers (40s, 50s)**
- **Used to encode numbers for seven-segment displays.**
- **Easier to read?**

# Gray Code

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

- Gray code is not a number system.
  - It is an alternate way to represent four bit data
- Only one bit changes from one decimal digit to the next
- Useful for reducing errors in communication.
- Can be scaled to larger numbers.

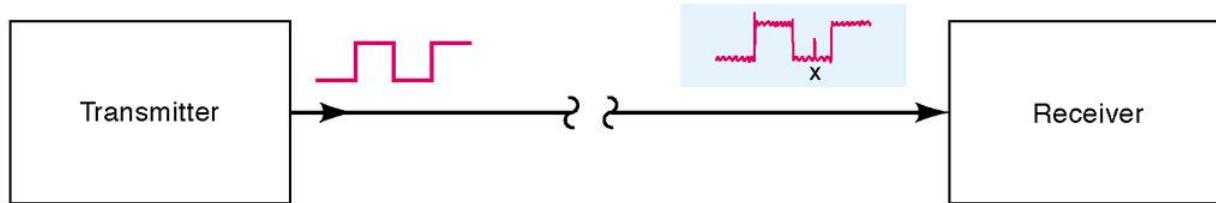
# ASCII Code

- **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange
- **ASCII** is a 7-bit code, frequently used with an 8<sup>th</sup> bit for error detection (more about that in a bit).

Character	ASCII (bin)	ASCII (hex)	Decimal	Octal
A	1000001	41	65	101
B	1000010	42	66	102
C	1000011	43	67	103
...				
Z				
a				
...				
1				
'				



# ASCII Codes and Data Transmission



- **ASCII Codes**
  - **A – Z (26 codes), a – z (26 codes)**
  - **0-9 (10 codes), others (@#\$%^&\*....)**
  - **Complete listing in Mano text**
- **Transmission susceptible to noise**
- **Typical transmission rates (1500 Kbps, 56.6 Kbps)**
  - **How to keep data transmission accurate?**

# Parity Codes

- Parity codes are formed by concatenating a *parity bit, P* to each code word of C.
- In an *odd-parity code*, the parity bit is specified so that the total number of ones is odd.
- In an *even-parity code*, the parity bit is specified so that the total number of ones is even.



1 1 0 0 0 0 1 1



Added even parity bit

0 1 0 0 0 0 1 1



Added odd parity bit

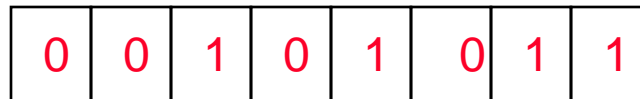
## Parity Code Example

- Concatenate a parity bit to the ASCII code for the characters 0, X, and = to produce both odd-parity and even-parity codes.

Character	ASCII	Odd-Parity ASCII	Even-Parity ASCII
0	0110000	10110000	00110000
X	1011000	01011000	11011000
=	0111100	10111100	00111100

# Binary Data Storage

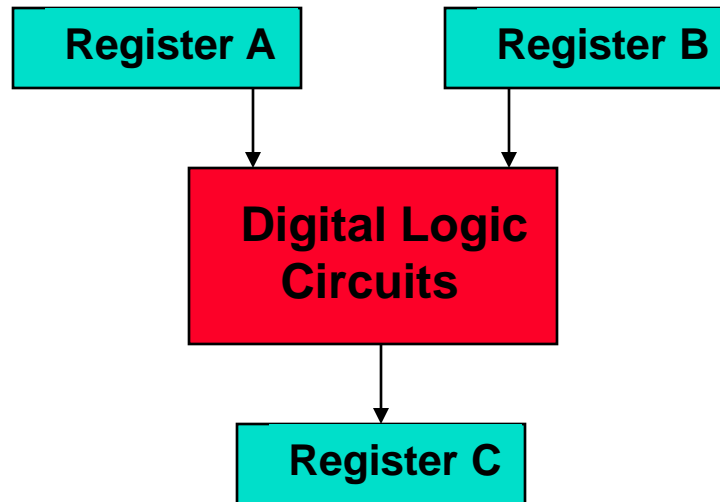
- **Binary cells** store individual bits of data
- Multiple cells form a **register**.
- Data in registers can indicate different values
  - Hex (decimal)
  - BCD
  - ASCII



Binary Cell

# Register Transfer

- **Data can move from register to register.**
- **Digital logic used to process data**
- **We will learn to design this logic**



# Transfer of Information

- Data input at keyboard
- Shifted into place
- Stored in memory

NOTE: Data input in ASCII

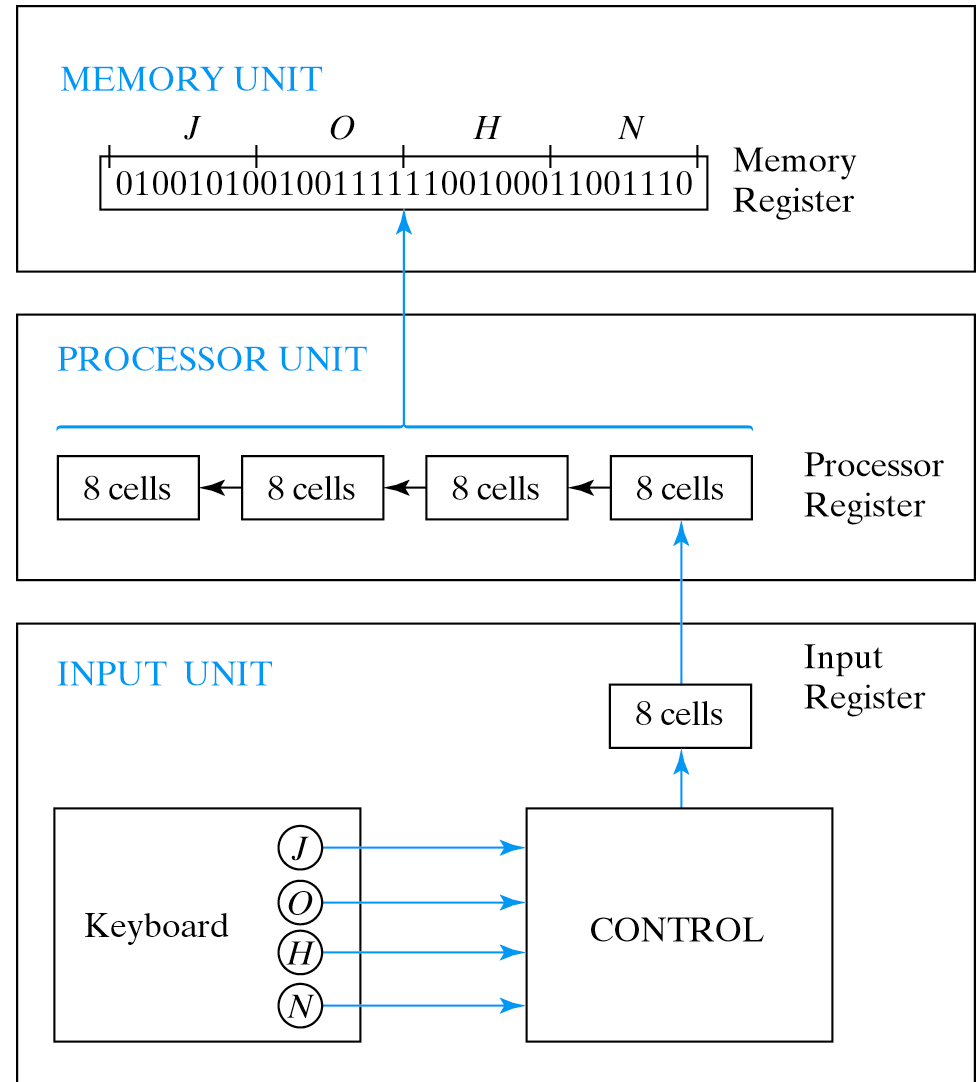


Fig. 1-1 Transfer of information with registers

# Building a Computer

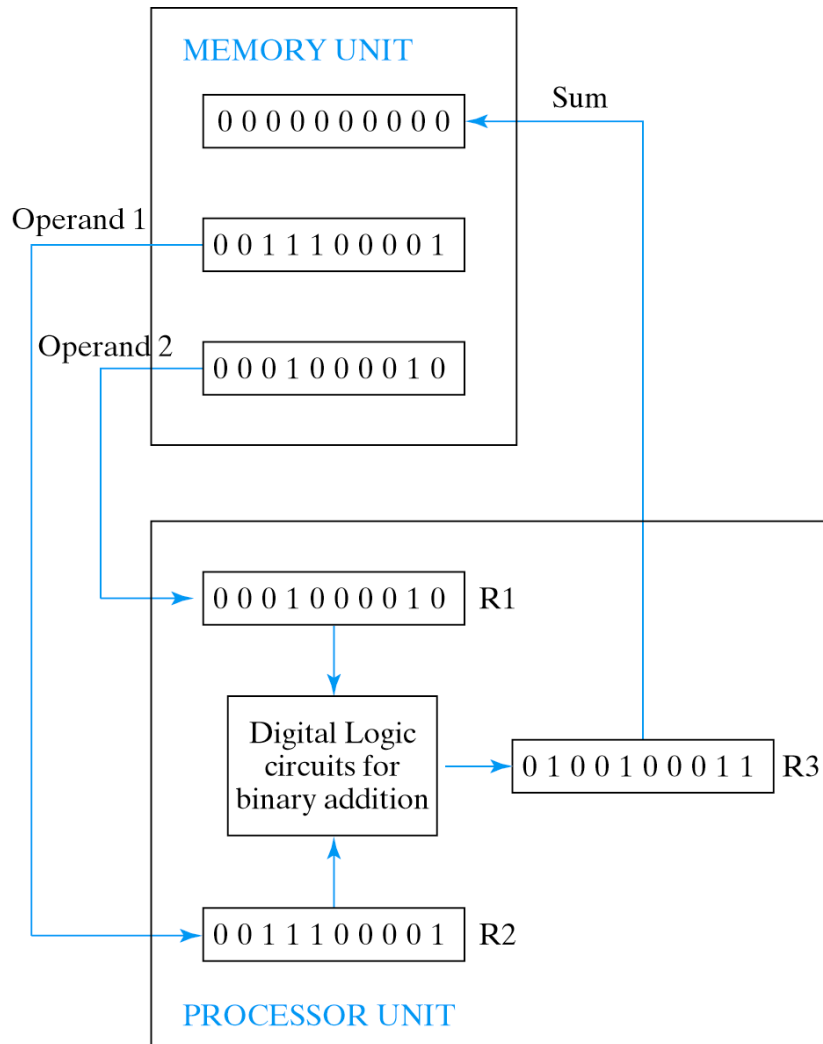


Fig. 1-2 Example of binary information processing

- **We need processing**
- **We need storage**
- **We need communication**
- **You will learn to use and design these components.**

# Summary

- **Although 2's complement most important, other number codes exist**
- **ASCII code used to represent characters (including those on the keyboard)**
- **Registers store binary data**
- **Next time: Building logic circuits!**

