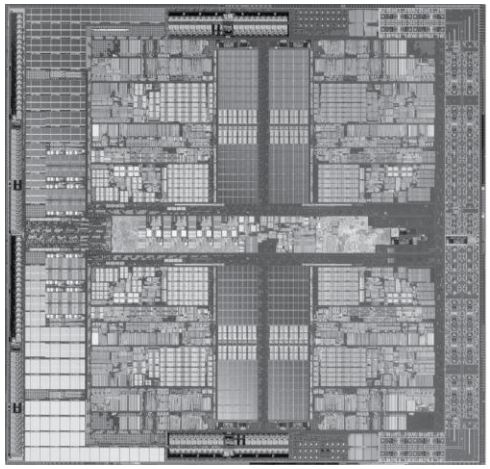
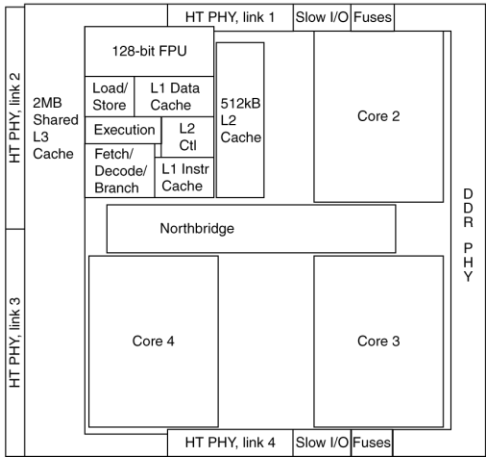
Chapter 1

Digital Systems and Number Systems

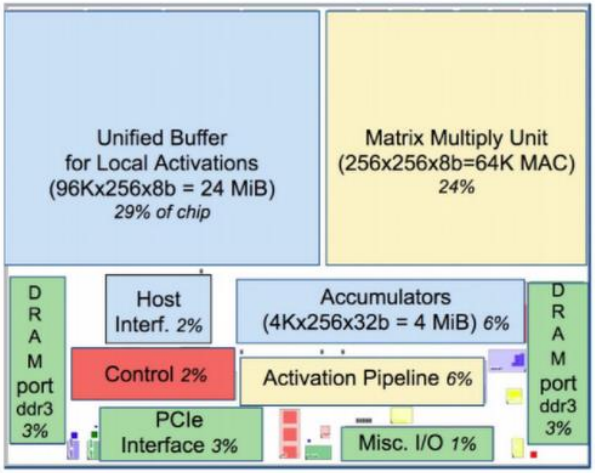
Why study digital logic design?

- Understand digital logic
- Learn how to design digital systems
- Understand computer architectures





AMD CPU (4 cores)
[2007]



Google's Tensor
Processor Unit (TPU)
[2017]

Introduction to Logic and Computer Design

This class introduces the fundamentals of digital logic design

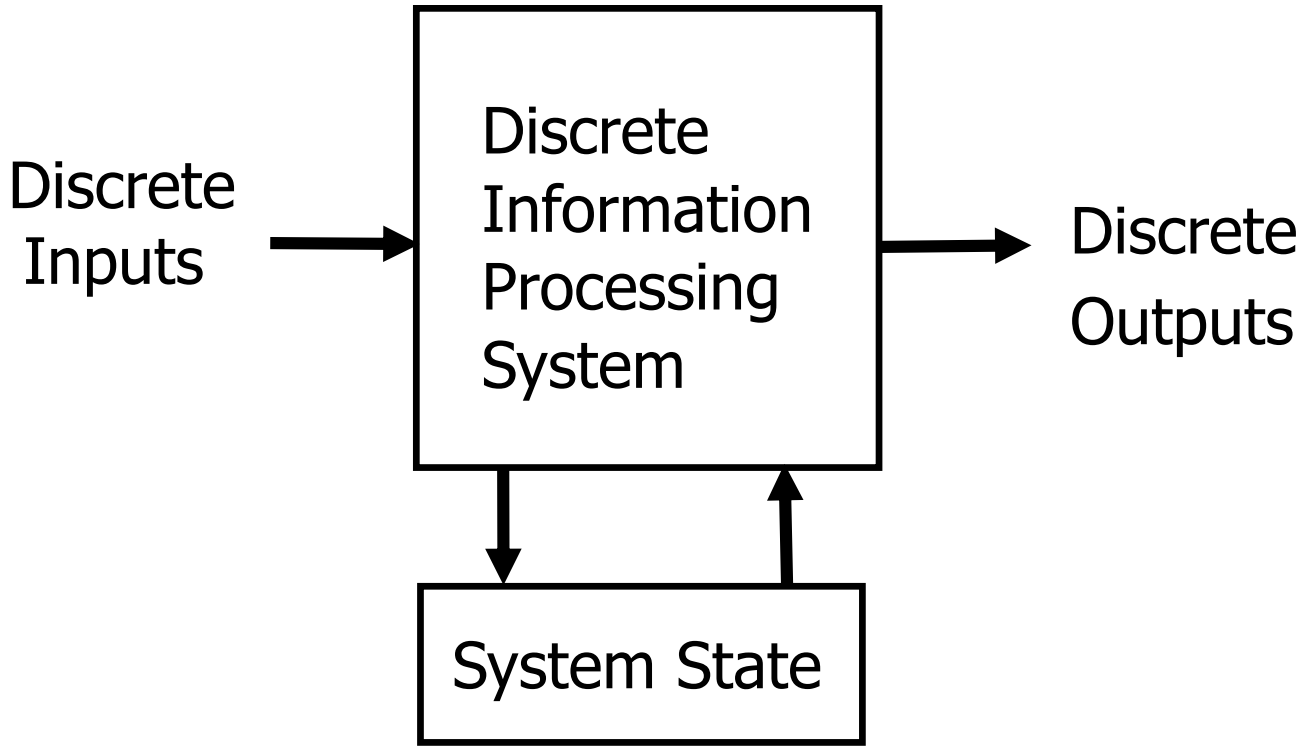
- Chapter 1 digital systems and number systems
- Chapter 2 combinational systems
- Chapter 3 the Karnaugh map
- Chapter 4 designing combinational systems
- Chapter 5 analysis of sequential systems
- Chapter 6 design of sequential systems
- Chapter 7 selected topics

Chapter 1 Digital systems and number systems

- Overview
  - Digital Systems, Computers, and Beyond
  - Information Representation
  - Number Systems [binary, octal and hexadecimal]
  - Arithmetic Operations
  - Base Conversion
  - Decimal Codes [BCD (binary coded decimal)]
  - Alphanumeric Codes
  - Parity Bit
  - Gray Codes

Digital systems

- Takes a set of discrete information inputs and discrete internal information (system state) and generates a set of discrete information outputs.



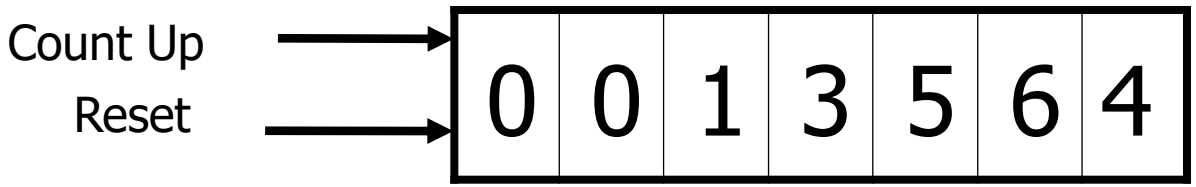
Types of digital systems

- No state present
  - Combinational Logic System
  - Output = Function(Input)

- State present
  - State updated at discrete times
    => Synchronous Sequential System
  - State updated at any time
    =>Asynchronous Sequential System
  - State = Function (State, Input)
  - Output = Function (State)
    or Function (State, Input)

Digital system example

A Digital Counter (e. g., odometer):



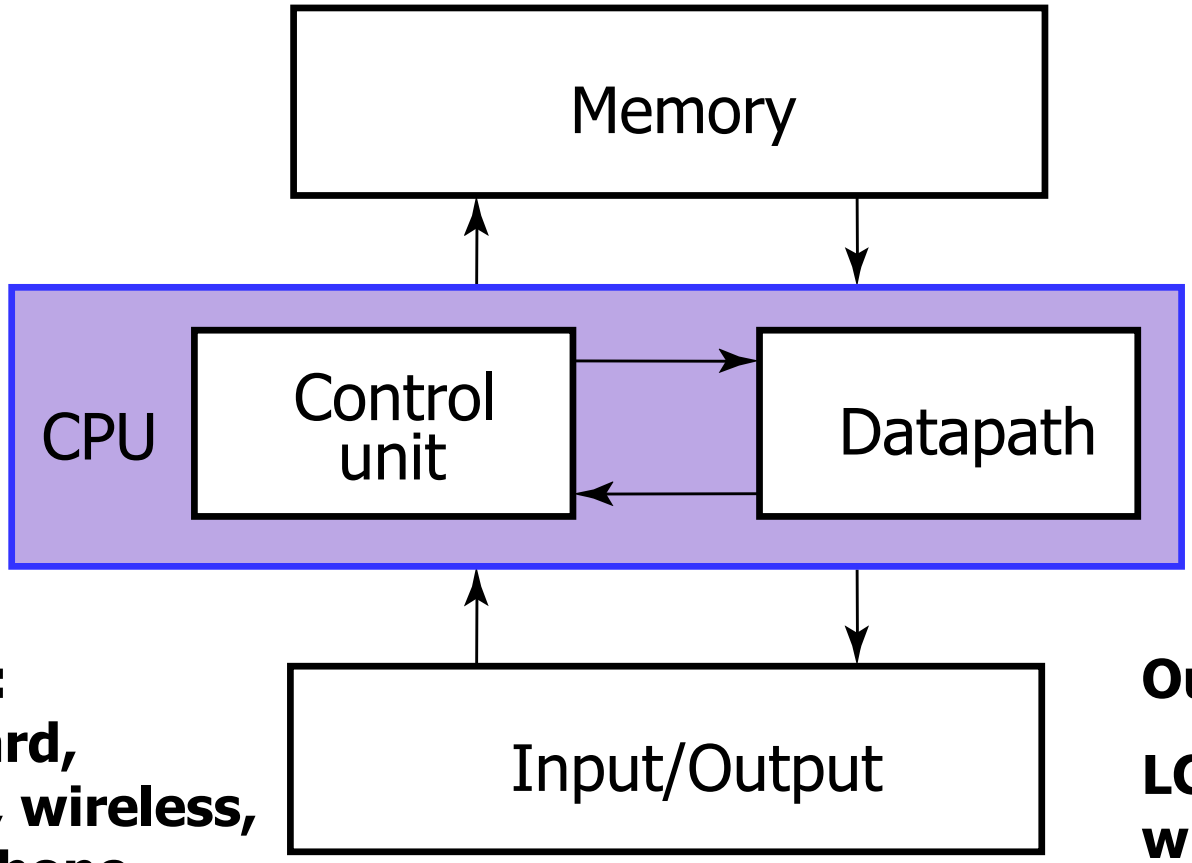
Inputs: **Count Up, Reset**

Outputs: **Visual Display**

State: **"Value" of stored digits**

Synchronous or Asynchronous?

Digital computer example



Inputs: keyboard, mouse, wireless, microphone

Outputs:

LCD screen, wireless, speakers

Synchronous or Asynchronous?

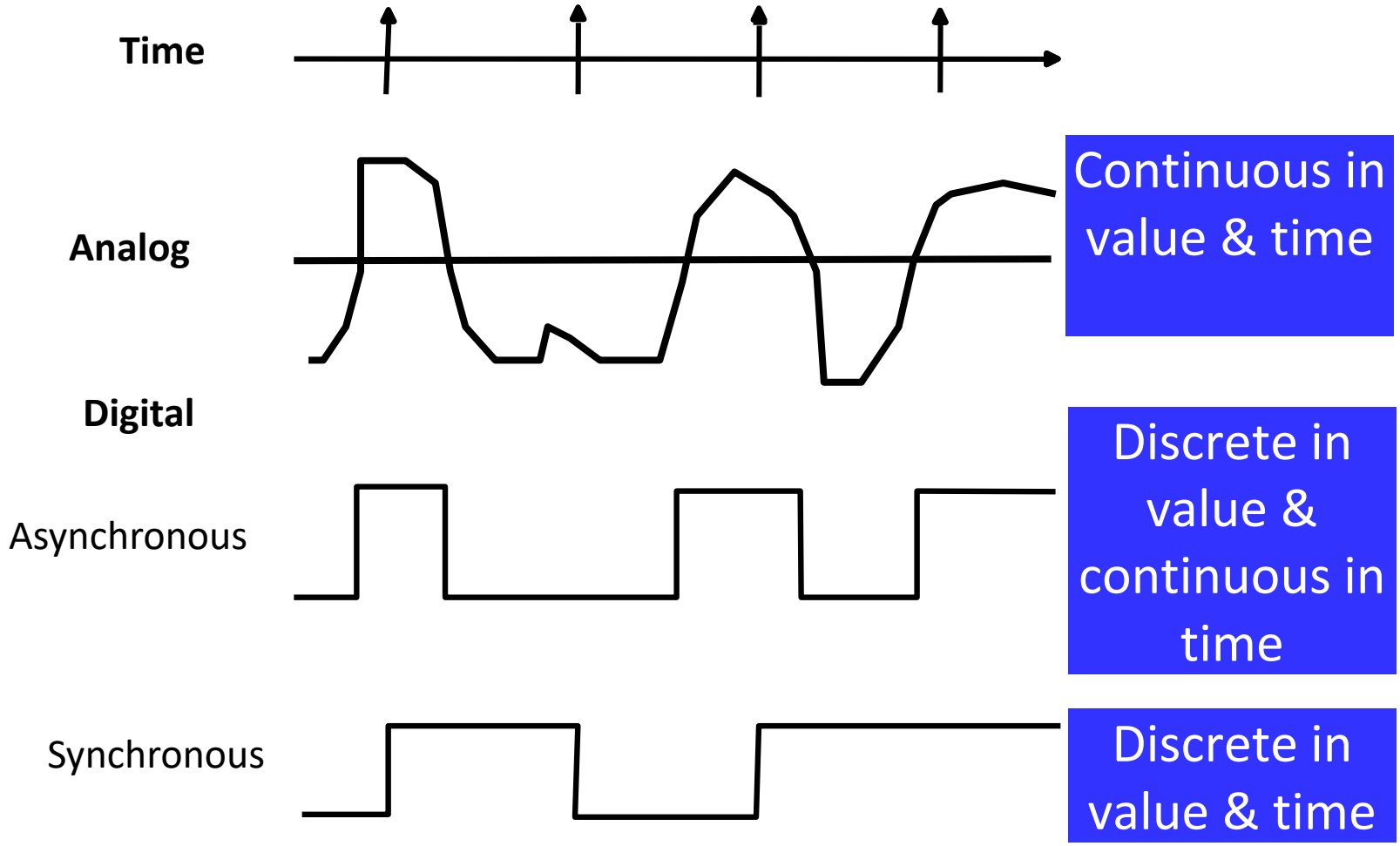
And embedded systems

- Computers as integral parts of other products
  - Cell phones
  - Automobiles
  - Video games
  - Global Positioning Systems (GPS)
  - Smart home electronics
  - Smart TV
  - Smart phone

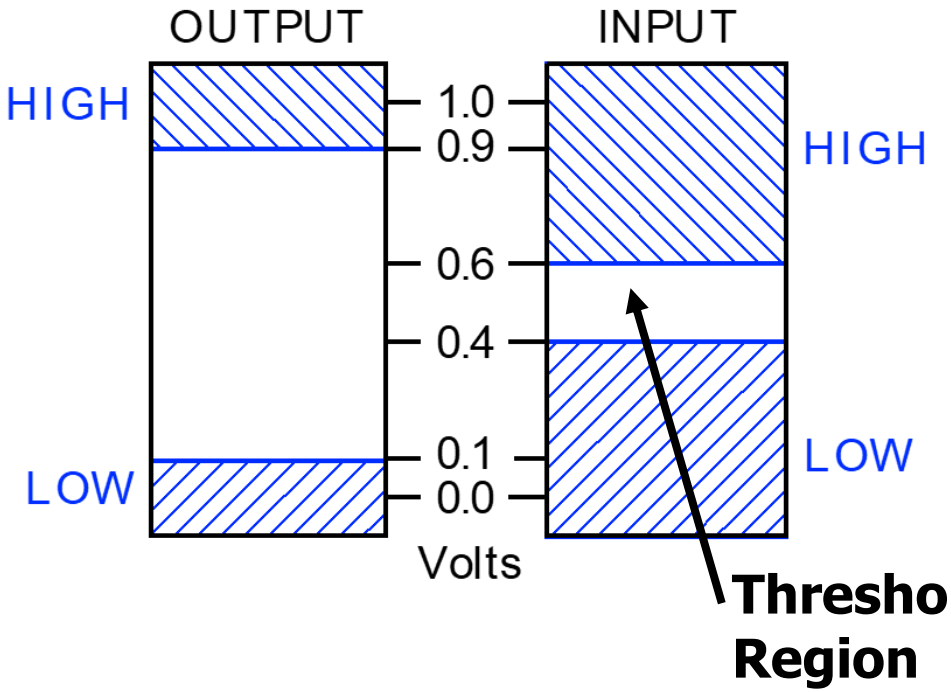
Information representation - Signals

- Information variables represented by physical quantities.
- For digital systems, the variables take on discrete values.
- Two level, or binary values are the most prevalent values in digital systems.
- Binary values are represented abstractly by:
- digits 0 and 1
- words (symbols) False (F) and True (T)
- words (symbols) Low (L) and High (H)
- and words On and Off.
- Binary values are represented by values or ranges of values of physical quantities

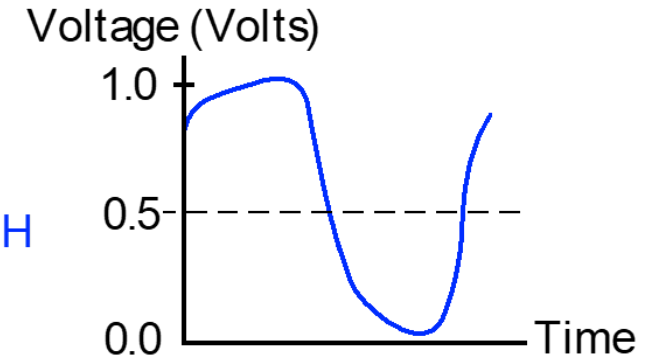
Signal examples over time



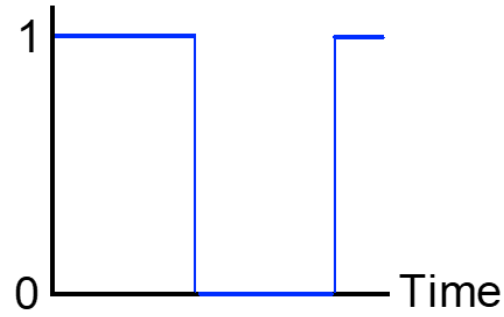
Signal Example – Physical Quantity: Voltage



(a) Example voltage ranges



(b) Time-dependent Voltage



(c) Binary model of time-dependent voltage

Number systems

$$N = a_{n-1}r^{n-1} + a_{n-2}r^{n-2} + \cdots + a_2 r^2 + a_1 r + a_0$$

n : number of digits
r : radix or base
a_i : coefficients

$$0 \le a_i < r$$

- Decimal number

$$7642_{10} = 7 \times 10^3 + 6 \times 10^2 + 4 \times 10^1 + 2$$

- Binary number

$$\begin{aligned} 101111_2 &= 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2 + 1 \\ &= 32 + 8 + 4 + 2 + 1 = 47_{10} \end{aligned}$$

Number systems example

		General	Decimal	Binary
Radix (Base)		**r**	**10**	**2**
Digits		**0 => r - 1**	**0 => 9**	**0 => 1**
	0	r^0	**1**	**1**
	1	r^1	**10**	**2**
	2	r^2	**100**	**4**
	3	r^3	**1000**	**8**
Powers of	**4**	r^4	**10,000**	**16**
Radix	**5**	r^5	**100,000**	**32**
	-1	r^{-1}	**0.1**	**0.5**
	-2	r^{-2}	**0.01**	**0.25**
	-3	r^{-3}	**0.001**	**0.125**
	-4	r^{-4}	**0.0001**	**0.0625**
	-5	r^{-5}	**0.00001**	**0.03125**

Number systems and Conversion

Decimal: $953.78_{10} = 9\times10^{2} + 5\times10^{1} + 3\times10^{0} + 7\times10^{-1} + 8\times10^{-2}$

Binary: $1011.11_{2} = 1\times2^{3} + 0\times2^{2} + 1\times2^{1} + 1\times2^{0} + 1\times2^{-1} + 1\times2^{-2}$

$$= 8 + 0 + 2 + 1 + \frac{1}{2} + \frac{1}{4} = 11\frac{3}{4} = 11.75_{10}$$

Radix(Base): $N = (a_4 a_3 a_2 a_1 a_0 . a_{-1} a_{-2} a_{-3})_R$

$$= a_4 \times R^4 + a_3 \times R^3 + a_2 \times R^2 + a_1 \times R^1 + a_0 \times R^0$$
$$+ a_{-1} \times R^{-1} + a_{-2} \times R^{-2} + a_{-3} \times R^{-3}$$

Example: $147.3_{8} = 1\times8^{2} + 4\times8^{1} + 7\times8^{0} + 3\times8^{-1} = 64 + 32 + 7 + \frac{3}{8}$

$$= 103.375_{10}$$

Hexa-Decimal: $A2F_{16} = 10\times16^{2} + 2\times16^{1} + 15\times16^{0} = 2560 + 32 + 15 = 2607_{10}$

Octal and Hexadecimal numbers

■ **Octal number system**

- 8 digits(0 ~ 7)
- 3-bit string ~ 1 octal digit

$100011001110_2 = 100\ \ 011\ \ 001\ \ 110 = 4316_8$

■ **Hexadecimal number system**

- 16 digits(0 ~ 9, A ~ F)
- 4-bit string ~ 1 hexadecimal digit

$100011001110_2 = 1000\ \ 1100\ \ 1110 = 8CE_{16}$

Number system conversions example

- **binary → octal** $10111011001_2 = 10\ \ 111\ \ 011\ \ 001 = 2731_8$
- **binary → hexadecimal** $10111011001_2 = 101\ 1101\ 1001 = 5D9_{16}$
- **binary → decimal** $10111011001_2 = 1 \cdot 1024 + 0 \cdot 512 + 1 \cdot 256 + 1 \cdot 128 + 1 \cdot 64 + 0 \cdot 32 + 1 \cdot 16 + 1 \cdot 8 + 0 \cdot 4 + 0 \cdot 2 + 1 \cdot 1 = 1497_{10}$

- **octal → binary** $1234_8 = 001\ \ 010\ \ 011\ \ 100_2$
- **octal → hexadecimal** $1234_8 = 001\ \ 010\ \ 011\ \ 100_2 = 0010\ \ 1001\ \ 1100_2 = 29C_{16}$
- **octal → decimal** $1234_8 = 1 \cdot 512 + 2 \cdot 64 + 3 \cdot 8 + 4 \cdot 1 = 668_{10}$

- **hexadecimal → binary** $C0DE_{16} = 1100\ \ 0000\ 1101\ \ 1110_2$
- **hexadecimal → octal** $C0DE_{16} = 1100\ \ 0000\ 1101\ \ 1110_2 = 1\ \ 100\ \ 000\ \ 011\ \ 011\ \ 110_2 = 140336_8$
- **hexadecimal → decimal** $C0DE_{16} = 12 \cdot 4096 + 0 \cdot 256 + 13 \cdot 16 + 14 \cdot 1 = 49374_{10}$

Representation of negative numbers : signed numbers

■ Sign-Magnitude Representation

- A number consists of tow parts: magnitude and sign
- The sign is represented by a single additional bit in binary numbers

  sign bit 0 : positive $\quad \mathbf{0}1010101_2 = +85_{10}$

  sign bit 1 : negative $\quad \mathbf{1}1010101_2 = -85_{10}$

- Signed-magnitude system has equal number of positive and negative integers
- Range of n-bits signed-magnitude integer : $-(2^{n-1}-1) \sim +(2^{n-1}-1)$
- Need to compare sign and magnitude when addition and subtraction

  $\rightarrow$ slower than complement number system

  sign bit 0 : positive $\quad \mathbf{0}101_2 = +5_{10}$

  sign bit 1 : negative $\quad \mathbf{1}101_2 = -5_{10}$

Representation of negative numbers : 2's complement

- 2's complement
  - Two's Complement

    Range of representable number $-(2^{n-1}) \sim +(2^{n-1}-1)$

    Positive number : stored in normal binary

    Negative number : 2^n – a in n-bit system

  - Converting steps for negative numbers in two's complement

    1. Find the binary equivalent of the magnitude
    2. Complement each bit (that is, change 0's to 1's and 1's to 0's)
    3. Add 1

	-5	-1	-0
1.	0101	0001	0000
2.	1010	1110	1111
3.	1	1	1
	1011	1111	0000

Table 1.6 Signed and unsigned 4-bit numbers.

Binary	Positive	Signed (two's complement)
0000	0	0
0001	1	+1
0010	2	+2
0011	3	+3
0100	4	+4
0101	5	+5
0110	6	+6
0111	7	+7
1000	8	−8
1001	9	−7
1010	10	−6
1011	11	−5
1100	12	−4
1101	13	−3
1110	14	−2
1111	15	−1

2's complement addition and subtraction

■ Addition Rules

- 4-bit two's complement additions

3	0011		-2		1110
+ 4	+ 0100		+ -6		+ 1010
7	0111		-8	ignored carry = **1**	1000

- Overflow
  Operation that produces a result that exceeds the range of the number number system

0100	(+4)
+ 0101	(+5)
1001	(-7)

- Addition overflow occurs whenever the sign of the sum is different from the sign of both addends

2's complement addition and subtraction

■ Subtraction Rules

- 4-bit two's complement subtractions:
  complementing the subtrahend and adding the minuend

4	0100		0100	-4	1100		1100
– 3	– 0011	⟶	+ 1101	– -8	– 1000	⟶	+ 1000
1			**1** 0001	4			**1** 0100

- Two's complement numbers are added and subtraction using same procedure(same logic circuit)

Binary Coded Decimal (BCD) Code

Decimal digit	8421 code	5421 code	2421 code	Excess 3 code	2 of 5 code
0	0000	0000	0000	0011	11000
1	0001	0001	0001	0100	10100
2	0010	0010	0010	0101	10010
3	0011	0011	0011	0110	10001
4	0100	0100	0100	0111	01100
5	0101	1000	1011	1000	01010
6	0110	1001	1100	1001	01001
7	0111	1010	1101	1010	00110
8	1000	1011	1110	1011	00101
9	1001	1100	1111	1100	00011
unused	1010	0101	0101	0000	any of
	1011	0110	0110	0001	the 22
	1100	0111	0111	0010	patterns
	1101	1101	1000	1101	with 0, 1,
	1110	1110	1001	1110	3, 4, or 5
	1111	1111	1010	1111	1's

Binary Coded Decimal

- The BCD code is the 8,4,2,1 code.
  - 8, 4, 2, and 1 are weights
  - BCD is a *weighted* code
  - This code is the simplest, most intuitive binary code for decimal digits and uses the same powers of 2 as a binary number, but only encodes the first ten values from 0 to 9.

- Example: 1001 (9) = 1000 (8) + 0001 (1)

- How many "invalid" code words are there?
- What are the "invalid" code words?

BCD addition

Given a BCD code, we use binary arithmetic to add the digits:

8	**1000**	**Eight**
+5	**+0101**	**Plus 5**
13	**1101**	**is 13 (> 9)**

Note that the result is MORE THAN 9, so must be represented by two digits!

To correct the digit, subtract 10 by adding 6 modulo 16.

8	**1000**	**Eight**
+5	**+0101**	**Plus 5**
13	**1101**	**is 13 (> 9)**
	+0110	**so add 6**
carry = 1	**0011**	**leaving 3 + cy**
0001	**0011**	**Final answer (two digits)**

If the digit sum is > 9, add one to the next significant digit

Excess 3 Code and 8, 4, –2, –1 Code

Decimal	Excess 3	8, 4, –2, –1
0	0011	0000
1	0100	0111
2	0101	0110
3	0110	0101
4	0111	0100
5	1000	1011
6	1001	1010
7	1010	1001
8	1011	1000
9	1100	1111

What interesting property is common to these two codes?

Warning: Conversion or Coding?

- Do NOT mix up conversion of a decimal number to a binary number with coding a decimal number with a BINARY CODE.

- **$13_{10} = 1101_2$ (This is conversion)**
- **13 $\Leftrightarrow$ 0001|0011 (This is coding)**

Gray code

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

- Consecutive numbers differ in only one bit
- Useful in coding the position of a continuous device

Hamming Code

- By Richard Hamming in 1950
- Single error correction code
- 4 data bits and 3 check bits
- The check bit is chosen so that the total number of 1's in the bits selected is even

	a_1	a_2	a_3	a_4	a_5	a_6	a_7
Bit 1	X		X		X		X
Bit 2		X	X			X	X
Bit 3				X	X	X	X

$$a_1 = a_3 \oplus a_5 \oplus a_7$$
$$a_2 = a_3 \oplus a_6 \oplus a_7$$
$$a_4 = a_5 \oplus a_6 \oplus a_7$$

Hamming code

Date/Bit	a_1	a_2	a_3	a_4	a_5	a_6	a_7
0000	0	0	0	0	0	0	0
0001	1	1	0	1	0	0	1
0010	0	1	0	1	0	1	0
0011	1	0	0	0	0	1	1
0100	1	0	0	1	1	0	0
0101	0	1	0	0	1	0	1
0110	1	1	0	0	1	1	0
0111	0	0	0	1	1	1	1
1000	1	1	1	0	0	0	0
1001	0	0	1	1	0	0	1
1010	1	0	1	1	0	1	0
1011	0	1	1	0	0	1	1
1100	0	1	1	1	1	0	0
1101	1	0	1	0	1	0	1
1110	0	0	1	0	1	1	0
1111	1	1	1	1	1	1	1

Hamming code example

- Received : 0010011
  - $e_1 = 0$ $e_2 = 1$ $e_4 = 0$
  - bit 2(check bit) error => $a_2 = 1$
  - Correct word : 0110011, Data : 1011

- Received : 1101101
  - $e_1 = 1$ $e_2 = 0$ $e_4 = 1$
  - bit 5 error => $a_5 = 0$
  - Correct word : 1101001, Data : 0001

Checking $4e_4 + 2e_2 + e_1$ for finding error bit

where e1= a3^a5^a7^a1 etc.

For n check bit, there can be $2^n - n - 1$ information bits