#1491: Introduction to Digital Systems

©*Yao-Wen Chang*
ywchang@cis.nctu.edu.tw
http://www.cis.nctu.edu.tw/~ywchang

Department of Computer & Information Science
National Chiao Tung University
Hsinchu 300, Taiwan

Spring 1998

Administrative Matters

- **Time/Location:** Mondays 3:40–4:30pm, Wednesdays 10:10am–12:00pm; ED 302; #1491.
- **Instructor:** Yao-Wen Chang.
- **E-mail:** ywchang@cis.
- **URL:** http://www.cis.nctu.edu.tw/~ywchang.
- **Office:** EC 125B. (Tel) 5712121 Ext. 56606; (Fax) 5721490.
- **Office Hours:** Mondays 4:40–5:40pm; other times by appointment.
- **Teaching Assistants:** Yue-Tsang Chang (gis86558@cis); Wen-Hao Shiue (gis86569@cis).
- **Prerequisites:** Introduction to Computer Sciences.
- **Goals:** To be able to design and analyze digital logic systems by understanding formal foundations and popular design techniques.

Administrative Matters (Cont'd)

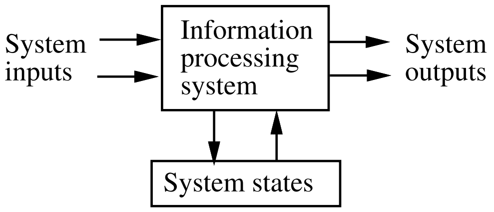
- **Required Text:** M. M. Mano and C. R. Kime, *Logic and Computer Design Fundamentals*, Prentice Hall, 1997. ISBN: 0-13-206780-3.
- **References:**
  - D. D. Gajski, *Principles of Digital Design*, Prentice Hall, 1997.
  - R. H. Katz, *Contemporary Logic Design*, Benjamin/Cummings, 1994.
  - C. H. Roth, Jr, *Fundamentals of Logic Design*, 4th Ed., PWS, 1995.
  - J. F. Wakerley, *Digital Design Principles and Practices*, Prentice Hall, 1990.
- **Course Contents:**
  - Digital computers and information
  - Boolean algebra, combinational (combinatorial) logic
  - Sequential (synchronous & asynchronous) logic
  - Memory and programmable logic devices
  - Register transfers and datapaths
  - Digital systems

Administrative Matters (Cont'd)

- **Grading:**
  - Bi-weekly homework assignments: 20%
  - Two 50-min quizzes: 20% (**March 16th; May 4th**)
  - Two 110-min midterms: 45% (**April 8th ; June 3rd**)
  - Final project & presentation: 15% (due **June 10**, tentative)
- **Homework:** Penalty for late submission: 50% per day.
- **Academic Honesty:** Avoid *cheating* at all cost.
- **WWW:** http://www.cis.nctu.edu.tw/~ywchang/Courses/Log98/log98.html.
  - Required web browser: **ghostview**
- **Personal Information:** E-mail me your name, student ID, **4–6 character Secret ID**.

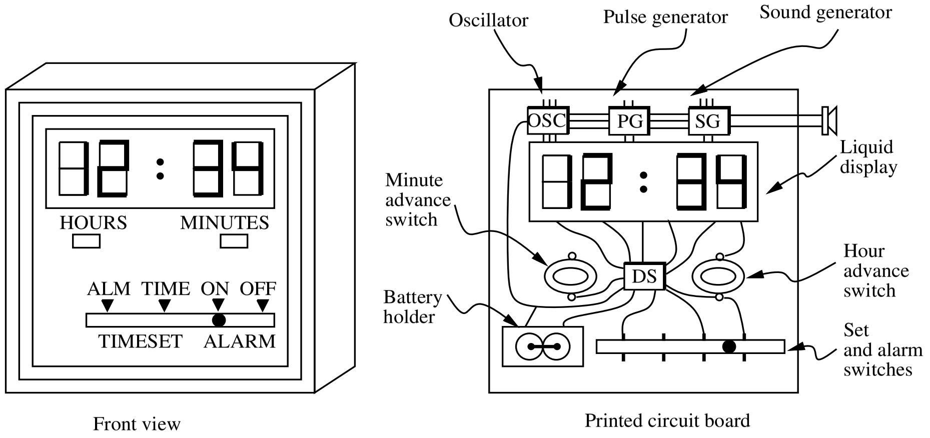
What is a "Digital/Analog System"

- **Digital system:** Takes a set of **discrete** information inputs and **discrete** internal information (system states) and generates a set of **discrete** information outputs.
- **Analog system:** Inputs, outputs, and states can take any value within a given range.
- **Analog-to-digital (A/D) converter:** Takes analog inputs and generates digital outputs.
- **Digital-to-analog (D/A) converter:** Takes digital inputs and generates analog outputs.
- **States:**
  - No state $\Longrightarrow$ Combinational (combinatorial) system: $Output := f(Inputs)$.
  - State present: State updated at **discrete** times $\Longrightarrow$ **Synchronous sequential** system; State updated at **any** time $\Longrightarrow$ **Asynchronous sequential** system.



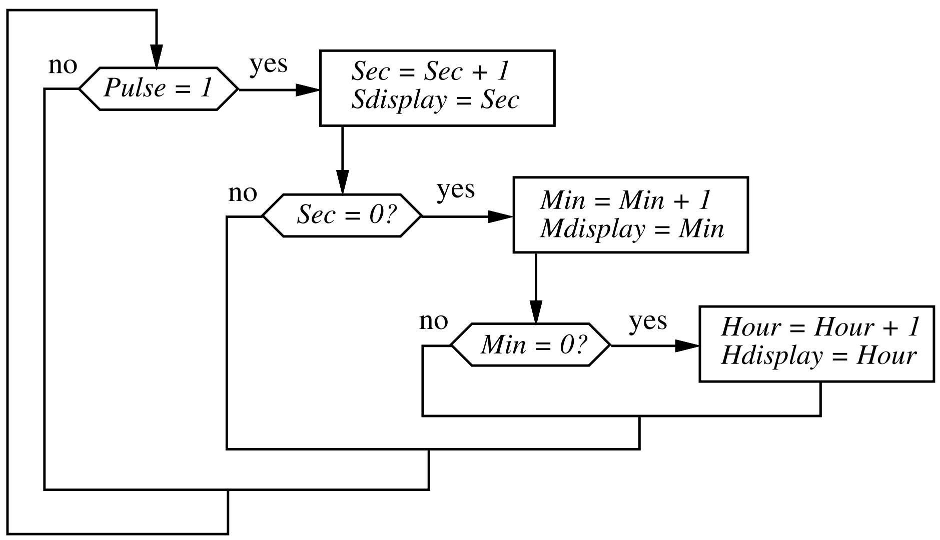
Mixed Digital/Analog System Example: Alarm Clock

- Outputs: visual display, sound; Inputs? States?
- Alarm clock **physical** representation:



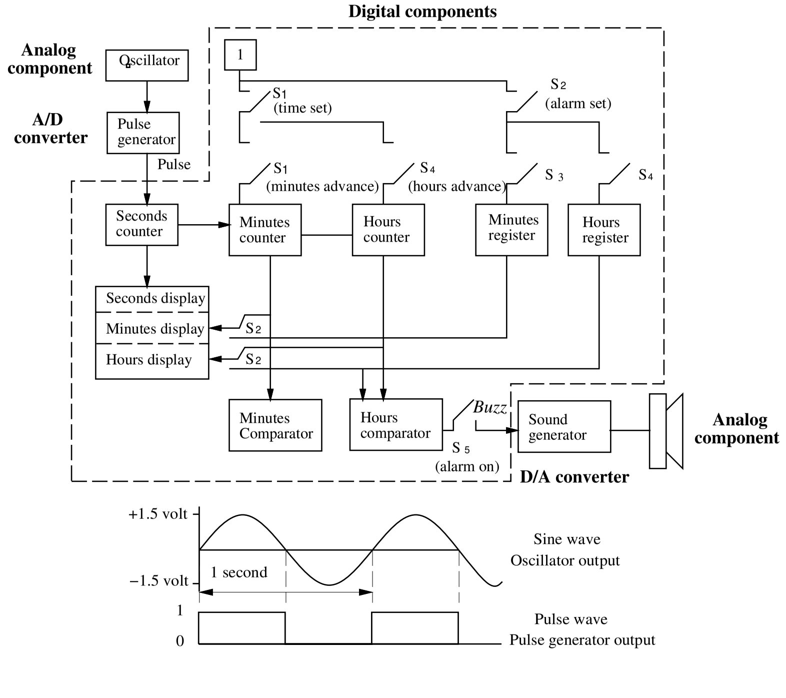
Mixed Digital/Analog System Example: Alarm Clock

- Alarm clock **behavioral** (**functional**) representation: clock process



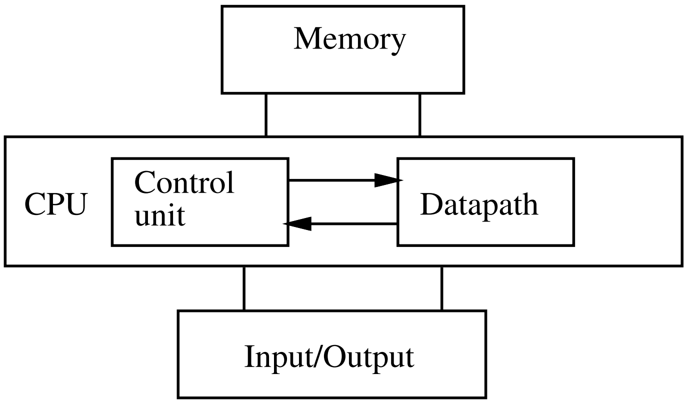
Mixed Digital/Analog System Example: Alarm Clock

- Alarm clock **structural** representation.

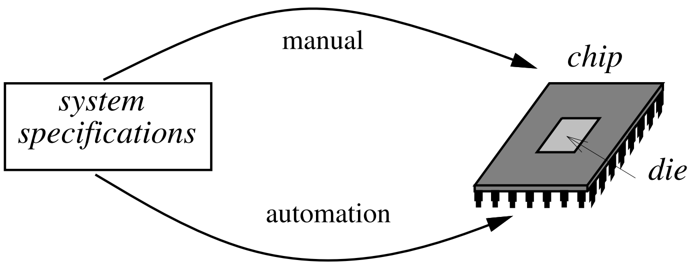


Digital Computer Example

- Inputs: Data from keyboard, disk drive, floppy, microphone, etc.
- Outputs: Data from screen, disk drive, floppy, sound card, etc.
- State: Data in memory, registers.
- Synchronous or asynchronous?



Design Considerations



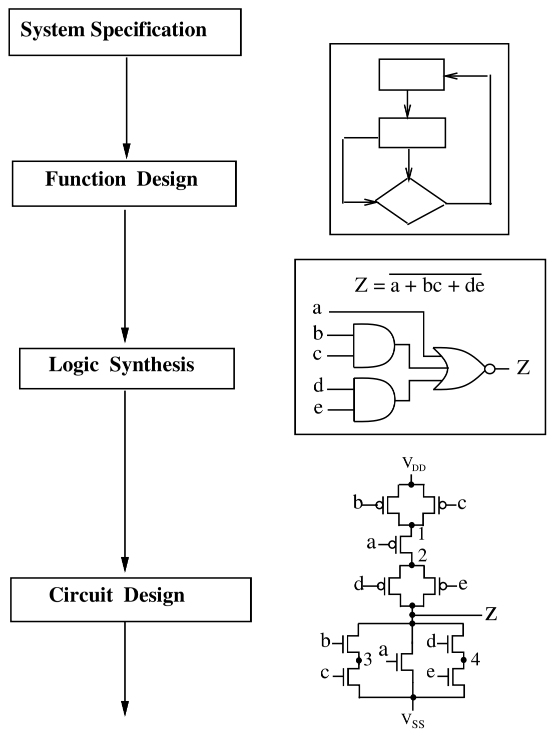
- Several conflicting considerations
  - **Design complexity:** large number of devices/transistors
  - **Performance:** optimization requirements for high performance
  - **Time-to-market:** about a 15% gain for early birds
  - **Cost:** die area, packaging, testing, etc

Traditional Design Cycle

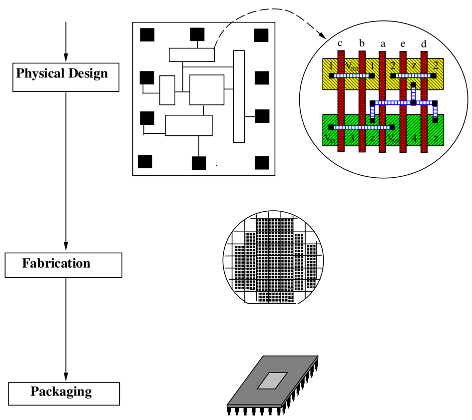
1. System specification
2. Functional design
3. **Logic design and synthesis**
4. Circuit design
5. Physical Design
6. Fabrication
7. Packaging

- Other tasks involved: Design verification, Testing, simulation

Traditional Design Flow

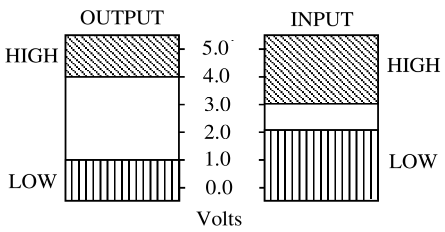


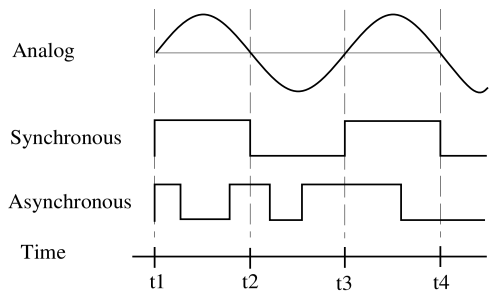
Traditional Design Flow (Cont'd)



Signals

- A collection of information variables mapped to some physical quantity.
- For digital systems, the quantities take on discrete values.
- Two level (**binary**) values are the most popular values in digital systems.





Number Systems

- Decimal (base-10) numbers: $(9876.5)_{10} = 9 \times 10^3 + 8 \times 10^2 + 7 \times 10^1 + 6 \times 10^0 + 5 \times 10^{-1}$.
- Binary (base-2) numbers: Only the digits 0 and 1 are needed; $(11010.1)_2 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} = (26.5)_{10}$.

n	2^n	n	2^n	n	2^n
0	1	8	256	16	65,536
1	2	9	512	17	131,072
2	4	10	1,024	18	262,144
3	8	11	2,048	19	524,288
4	16	12	4,096	20	1,048,576
5	32	13	8,192	21	2,097,152
6	64	14	16,384	22	4,194,304
7	128	15	32,768	23	8,388,608

- K (kilo): 2^{10}; M (mega): 2^{20}; G (giga): 2^{30}; T (tera): 2^{40}.

Binary-Decimal Conversion

- Binary → Decimal: Use decimal arithmetic to sum the weighted powers of two.
  - $(11010.1)_2 = 1\times 2^4 + 1\times 2^3 + 0\times 2^2 + 1\times 2^1 + 0\times 2^0 + 1\times 2^{-1} = 16 + 8 + 0 + 2 + 0 + 0.5 = (26.5)_{10}$.
- Decimal → Binary:
  - **S1:** Find the largest power of two that you can subtract and still be positive.
  - **S2:** Subtract it and note the power.
  - **S3:** Repeat until the remainder is zero.
  - $(625)_{10} = 2^9 + 2^6 + 2^5 + 2^4 + 2^0 = (1001110001)_2$.

$$\begin{array}{rrcrrl}
2^9 = & 512 & \Rightarrow & 625\text{-}512 = & 113 = & N_1 \\
2^6 = & 64 & \Rightarrow & 113\text{-}64 = & 49 = & N_2 \\
2^5 = & 32 & \Rightarrow & 49\text{-}32 = & 17 = & N_3 \\
2^4 = & 16 & \Rightarrow & 17\text{-}16 = & 1 = & N_4 \\
2^0 = & 1 & \Rightarrow & 1\text{-}1 = & 0 = & N_5
\end{array}$$

Common Bases

Decimal (base 10)	Binary (base 2)	Octal (base 8)	Hexadecimal (base 16)
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Base Conversion

- Given a number of **radix** r (base-r) with
  - n **integer** digits, $a_{n-1}a_{n-2}\ldots a_0$ and m **fraction** digits $a_{-1}a_{-2}\ldots a_{-m}$
  - Written so: $a_{n-1}a_{n-2}\ldots a_0.a_{-1}a_{-2}\ldots a_{-m}$
- $(N)_r = \underbrace{\sum_{i=0}^{n-1} a_i r^i}_{Integer\ portion} + \underbrace{\sum_{j=-m}^{-1} a_j r^j}_{Fraction\ portion}$
  - $(9876.54)_{10} \Longrightarrow a_3a_2a_1a_0.a_{-1}a_{-2} = 9 \times 10^3 + 8 \times 10^2 + 7 \times 10^1 + 6 \times 10^0 + 5 \times 10^{-1} + 4 \times 10^{-2}$.
  - $(11010.1)_2 \Longrightarrow a_4a_3a_2a_1a_0.a_{-1} = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1}$.
- Base conversion
  - **S1:** Convert the integer portion.
  - **S2:** Convert the fraction portion.
  - **S3:** Join the two with a radix point.

Base Conversion

- **S1:** Convert the integer portion.
  - Repeatedly **divide** the integer by the radix r you want to convert to, and save the **remainders**.
  - The new radix digits are the remainders in **reverse order** of computation.
  - Exp: $(41)_{10} = (101001)_2$.

41 ÷ 2 => 20	rem = 1	↑	Least significant digit
20 ÷ 2 => 10	rem = 0		
10 ÷ 2 => 5	rem = 0		
5 ÷ 2 => 2	rem = 1		
2 ÷ 2 => 1	rem = 0		
1 ÷ 2 => 0	rem = 1		Most significant digit

- **S2:** Convert the fraction portion.
  - Repeatedly **multiply** the fraction by the radix r, and save the **integer digits**.
  - The new radix fraction digits are the integer digits in **computed order**.
  - Exp: $(0.6875)_{10} = (.1011)_2$.

0.6875 * 2 = 1.3750	Integer = 1		Most significant digit
0.3750 * 2 = 0.7500	Integer = 0		
0.7500 * 2 = 1.5000	Integer = 1		
0.5000 * 2 = 1.0000	Integer = 1	↓	Least significant digit

- **S3:** Join the two with a radix point. (So, $(41.6875)_{10} = (101001.1011)_2$.)

Binary, Octal, and Hexadecimal Conversion

- Binary to Hexadecimal
  - Group the binary digits into **four-bit** groups, starting at the radix point and going both ways—integer and fraction—(padding with zeros as needed).
  - Convert each group of four bits to an hexadecimal digit.
  - Exp: $(1011100101.0111101101)_2 = (2E5.7B4)_{16}$.
- Hexadecimal to Binary
  - Restate the hexadecimal as **four** binary digits, starting at the radix point and going both ways (padding with zeros as needed).
  - Exp: $(2E5.7B4)_{16} = (1011100101.0111101101)_2$.

$$\begin{array}{cccccccc} 0010 & 1110 & 0101 & \bullet & 0111 & 1011 & 0100 \\ \updownarrow & \updownarrow & \updownarrow & & \updownarrow & \updownarrow & \updownarrow \\ 2 & E & 5 & \bullet & 7 & B & 4 \end{array}$$

Coding

- **Number of representable elements:** Given n digits in radix r, there are r^n distinct elements.
  - Exp: $r = 2, n = 3$ (3 binary digits) $\Longrightarrow$ 8 elements (000, 001, 010, 011, 100, 101, 110, 111).
- **Digital representation of numeric information:** Assign numeric meanings to the r^n distinct patterns of n radix-r digits.
  - Exp: binary digits $\leftrightarrow$ decimal (rainbow color): 000 $\leftrightarrow$ 0 (red); 001 $\leftrightarrow$ 1 (orange); ...; 110 $\leftrightarrow$ 6 (violet); 111 $\leftrightarrow$ 7 (not mapped).
  - Unsigned integers, signed integers, floating-point numbers?
- **Representation considerations:**
  - Expected **range** of data: 16-bit unsigned integer, etc?
  - **Simple computation** under common arithmetic operations.

Coding (cont'd)

- **Number of bits required for coding:** Given M elements to be represented by a **binary** code, the minimum # of bits, n, needed satisfies:

$$2^n \geq M \quad and \quad M > 2^{n-1}$$

  - Exp: How many bits to represent **decimal** digits with a binary code? ($M = 10 \Rightarrow n = 4$)
- **Conversion vs. Coding**: Don't mix up **conversion** of a decimal number to a binary number with **coding** a decimal number with a binary code.
  - Conversion: $12_{10} = 1100_2$
  - Coding: 12 $\Leftrightarrow$ 00010010

Binary Codes for Decimal Digits

- 4-binary digit code for decimal: $P(16, 10)$ ways!!
- **Binary Coded Decimal (BCD)** or **(8, 4, 2, 1) Code:**: Use the same weights as a binary number, but only encodes 0–9.
- **Excess-3 Code:**: Add 3 (0011) to the BCD code; self-complementing code.
- **Weighted codes:** BCD, (8, 4, -2, -1), etc.
  - Each bit has a **weight**; to compute the decimal value, add the weights where a 1 exists in the code word.

Decimal Symbol	BCD Digit (8, 4, 2, 1)	Excess–3 (self– complement)	(8, 4, –2, –1)
0	0000	0011	0000
1	0001	0100	0111
2	0010	0101	0110
3	0011	0110	0101
4	0100	0111	0100
5	0101	1000	1011
6	0110	1001	1010
7	0111	1010	1001
8	1000	1011	1000
9	1001	1100	1111

Binary Addition

- Given binary digits (X, Y), a carry in (Z), we get the sum (S) and carry (C):
  - $Z = 0$:

$$\begin{array}{rrrrr} Z & 0 & 0 & 0 & 0 \\ X & 0 & 0 & 1 & 1 \\ +Y & +0 & +1 & +0 & +1 \\ \hline \overline{C\ S} & \overline{0\ 0} & \overline{0\ 1} & \overline{0\ 1} & \overline{1\ 0} \end{array}$$

  - $Z = 1$:

$$\begin{array}{rrrrr} Z & 1 & 1 & 1 & 1 \\ X & 0 & 0 & 1 & 1 \\ +Y & +0 & +1 & +0 & +1 \\ \hline \overline{C\ S} & \overline{0\ 1} & \overline{1\ 0} & \overline{1\ 0} & \overline{1\ 1} \end{array}$$

- Multiple digits:

Carries	00000	101100
Augend	01100	10110
Addend	+10001	+10111
Sum	11101	101101

Multiplication

- Binary multiplication:

Multiplicand	1011
Multiplier	x 101
Partial Products	1011
	0000 -
	1011
Sum	110111

- Octal multiplication:

Multiplicand	762
Multiplier	x 45
Partial Products	4672
	3710 -
Sum	43772

BCD Arithmetic

- Given a BCD code, first use binary arithmetic to add the digits.
- If the digit sum > 9, then **add 6** (110) to the sum.

8	1000	Eight
+5	+0101	Plus 5
13	1101	13 (> 9)
	+0110	so add 6
	($C = 1$) 0011	leaving 3 and carry C
	$\underline{0001}$0011	Final answer

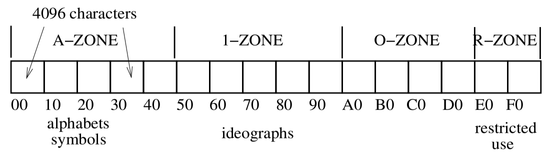
Error-Detection Codes

- **Redundancy** (e.g., extra information) can be added to a binary code word for the purpose of **detecting** and **correcting** errors.
- **Parity:** An extra bit is appended onto the code word to make the sum of bits odd or even.
  - **Even parity:** the sum of bits is even.
  - **Odd parity:** the sum of bits is odd.
- Parity can be used to **detect** all single-bit errors and some multiple-bit errors.
- Exp: Parity codes for 2 data bits

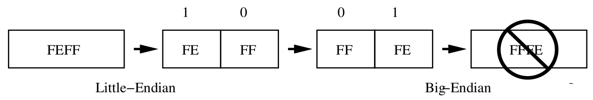
Even Parity	Odd Parity
Message - Parity	Message - Parity
00 - 0	00 - 1
01 - 1	01 - 0
10 - 1	10 - 0
11 - 0	11 - 1

ASCII: American Standard Code for Information Interchange

- Uses 7 bits to represent 94 graphic printing and 34 non-printing characters. (Refer to Table 1-4 in the text.)
  - Digits 0 to 9 are consecutive and span hexadecimal values 30_{16} to 39_{16}.
  - Upper case letters A–Z are consecutive and span 41_{16} to $5A_{16}$.
  - Lower case letters a–z are consecutive and span 61_{16} to $7A_{16}$.
  - Lower to upper case translation (and vice versa) occurs by complementing bit 6.
- **UNICODE** extends ASCII to 65536 (16-bit) universal characters codes.

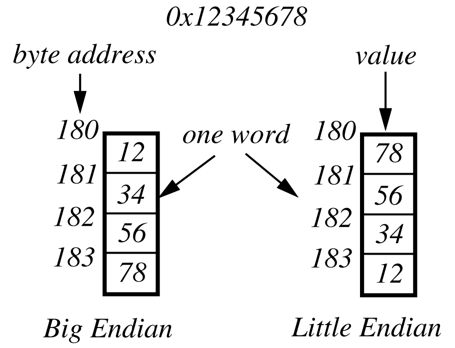


- **Byte-ordering problem?**



Byte Ordering

- **Big Endian:** Byte 0 is placed on the most significant end ($xx \ldots 00$ = Big end of word)
  - IBM 360/370, Motorola 68K, MIPS, SPARC, HP PA
  - Advantages: Consistent integers and character strings ordering, faster in character-string sorting, decimal/ASCII dumps.
- **Little Endian:** Byte 0 is placed on the least significant end ($00 \ldots xx$ = Little end of word)
  - Intel 80x86, Pentium, DEC VAX
  - Advantages: Integer address conversion, arithmetic.



Binary Logic

- **Binary variables:** one of two states.
  - True/False, On/Off, 1/0; variables are given names: $A, B, x, y, \ldots$
- **Logical operators:** operate on binary variables.
- **Logical operations:** AND, OR, NOT, etc.
  - **AND** ($\cdot$): $A \cdot B = Y$: "A AND B is equal to Y" (often omit $\cdot$)
  - **OR** ($+$): $x + y = z$: "x OR y is equal to z"
  - **NOT** ($'$ or overline): $A' = X$: "A NOT is equal to X"
  - Note: $1+1=2$: "one **plus** one equals two"; $1+1=1$: "one **or** one equals one";
- **Truth table:** Truth tables list the **output** of a function for **all possible inputs**.

AND

X	Y	Z=X*Y
0	0	0
0	1	0
1	0	0
1	1	1

OR

X	Y	Z=X+Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT

X	$Z=\overline{X}$
0	1
1	0