

CSE-103 Digital Computer Logic - Introduction

Athar Mahboob

WWW: <http://www.atharmahboob.com>

Email: athar@atharmahboob.com

Introduction to the Course and Instructor

- Course outline
- Textbook and reference book
- Class timings
- Policies
 - Assessment
 - Attendance
 - Ethics
- Instructor – how to contact, office hours
- Course website

What Do We Study in Digital Computer Logic?

- Computers are digital systems
- We study the issues in the hardware design of digital systems in Digital Computer Logic
- Digital systems have at least two parts:
 - Hardware
 - Software
- In addition, networking or communication capabilities of digital systems have become important nowadays.

What Does It Mean to be Digital?

- Digital vs. Analog
- Analog: Continuously variable in time and value. Examples: measurements of speed in a car, electrical representation of your voice.
- Digital: Only fixed levels of value are defined. Values can be updated at specified intervals of time only. Names of employees and their NIC numbers, list of temperature measurements taken at 5:00 PM every day with a 0.1 degree precision.

Why is Digital Better?

- Because computers are digital systems
- Computers can process digital information only
- All information can be represented in digital format
- Digital hardware is predictable
- Digital communication (error detection/correction, compression, encryption)
- Digital hardware is cheap – why?
- Digital storage – CD vs. tape

Why Study Digital Systems?

- Computer is not always a PC – implications?
- Many times computers are hidden within some thing else
- These are called **Embedded Systems**
- Embedded systems constitute majority of computers in the world

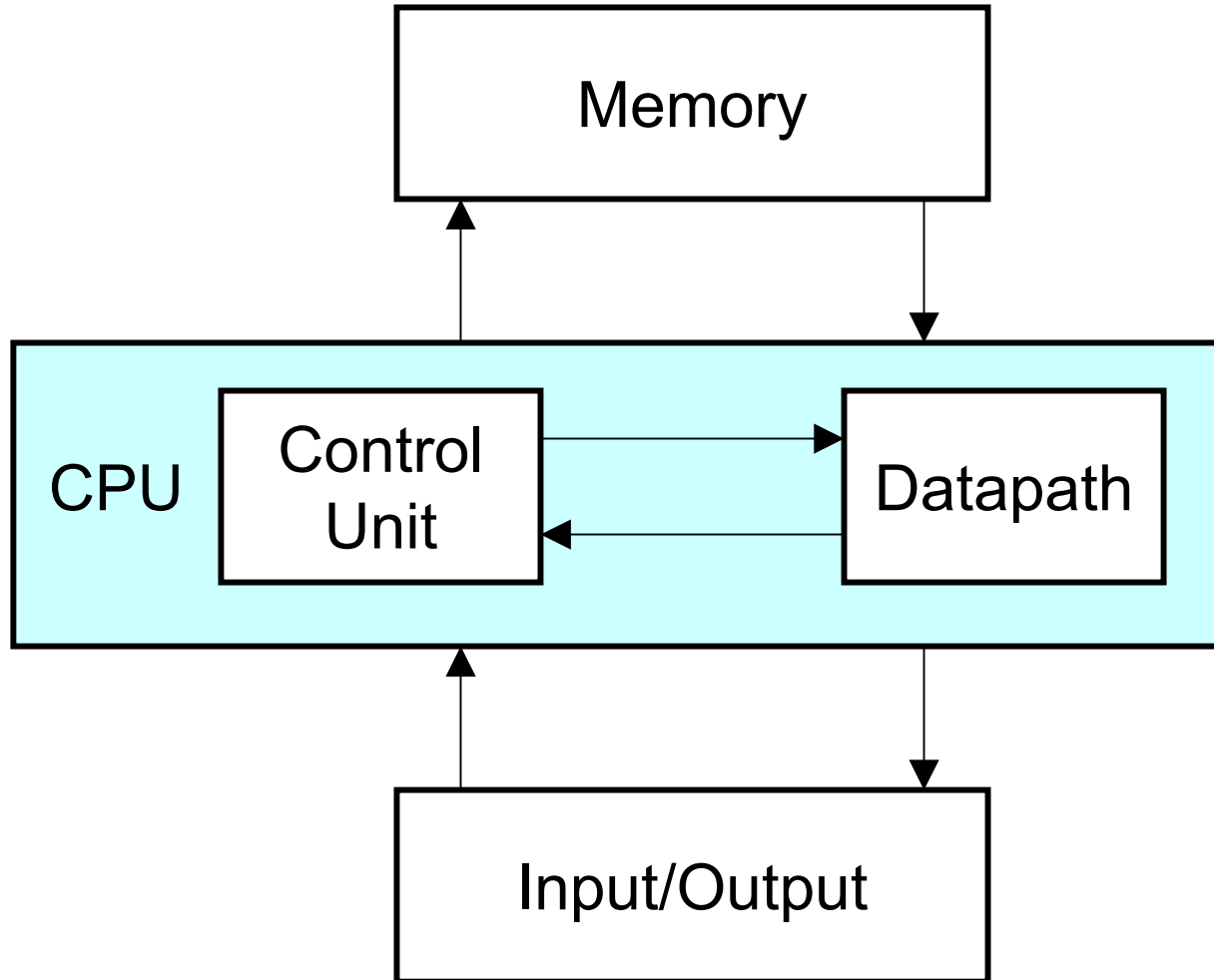
Why Should I Care About Hardware?

- Because it is real and it is there
- You really won't understand computers till you understand their hardware
- Many times product design involves hardware system design
- Many times solution to a problem can only be done with specialized hardware

What does Digital Hardware Look

- Computer hardware historically – mechanical, electromechanical, electronic
- Integrated circuits vs. Discrete components
- Levels of integration
- SSI, MSI, LSI, VLSI

Structure of a Computer



General Number System Concepts

- A base N number system has N unique digits and uses integer powers of the base to represent any number
- Base is also called radix
- Also know the radix point
- Any number can be represented in base r as follows:
 - $a_n \times r^n + a_{n-1} \times r^{n-1} + a_{n-2} \times r^{n-2} + \dots + a_1 \times r^1 + a_0 \times r^0 + a_{-1} \times r^{-1} + a_{-2} \times r^{-2} + \dots + a_{-m+1} \times r^{-m+1} + a_{-m} \times r^{-m}$

Number Systems

- The Decimal Number System uses base 10
- The Binary Number System uses base 2
- Other number systems:
 - Octal – base 8
 - Hexadecimal – base 16

Decimal Number System

- Natural for human beings
- Uses ten digits:
- 0 1 2 3 4 5 6 7 8 9
- ۰ ۱ ۲ ۳ ۴ ۵ ۶ ۷ ۸ ۹
- These are called Arabic Numerals but originally invented in India (that is why Arabs call them Hindu Numerals or “hindasa” - هندسه)

Example of Decimal Numbers

- $2561.782 = 2 \times 10^3 + 5 \times 10^2 + 6 \times 10^1 + 1 \times 10^0 + 7 \times 10^{-1} + 8 \times 10^{-2} + 2 \times 10^{-3}$
- The decimal point

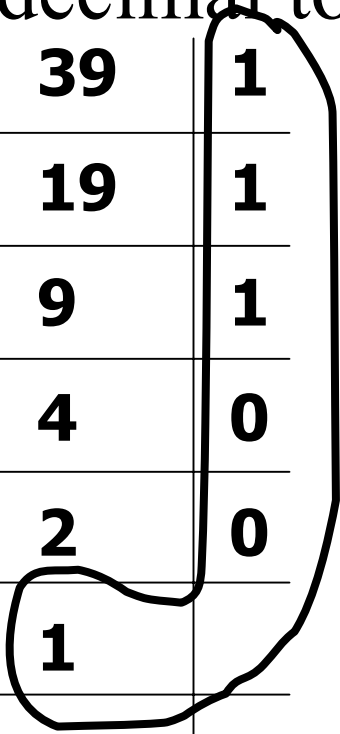
Binary Number System

- The Binary Number System uses base 2
- Why Computers use binary?
- Easy to represent just two things
- If computers used base 10 then they would need to represent 10 things.
- Computers use voltage to represent information.
- With just two levels there is more margin against the noise.
- With 10 levels noise can play havoc with the system.

Converting from Decimal to Binary

- Convert 39 from decimal to binary:

2	39	1
2	19	1
2	9	1
2	4	0
2	2	0
2	1	



$$(39)_{10} = (100111)_2$$

Converting from Binary to Decimal

- Convert $(10111101)_2$ to base 10.
- Convert $(1101110110)_2$ to base 10.

Powers of 2

n	2 ⁿ	n	2 ⁿ	n	2 ⁿ
0	1	17	131,072	-1	0.5000000000
1	2	18	262,144	-2	0.2500000000
2	4	19	524,288	-3	0.1250000000
3	8	20	1,048,576	-4	0.0625000000
4	16	21	2,097,152	-5	0.0312500000
5	32	22	4,194,304	-6	0.0156250000
6	64	23	8,388,608	-7	0.0078125000
7	128	24	16,777,216	-8	0.0039062500
8	256	25	33,554,432	-9	0.0019531250
9	512	26	67,108,864	-10	0.0009765625
10	1,024	27	134,217,728	-11	0.0004882813
11	2,048	28	268,435,456	-12	0.0002441406
12	4,096	29	536,870,912	-13	0.0001220703
13	8,192	30	1,073,741,824	-14	0.0000610352
14	16,384	31	2,147,483,648	-15	0.0000305176
15	32,768	32	4,294,967,296	-16	0.0000152588
16	65,536	33	8,589,934,592	-17	0.0000076294

Hexadecimal

- Base 16
- Gives a more compact notation.
- Uses 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F as digits
- Convert $(A45B)_{16}$ to base 10.
- Convert $(1256)_{10}$ to base 16.
- How is it a compact notation?
- Converting from hexadecimal to binary and vice versa.
- Hexadecimal is indicated by $0x$ prefix in computer literature.

Number Ranges

- An n -bit string can have 2^n unique values. For example a 3-bit number can have values 000, 001, 010, 011, 100, 101, 110, 111
- Hence with n -bits representing positive integers, integers from 0 to $2^n - 1$ can be represented
- With 8 bits 0-255
- With 16 bits 0 to 65535
- Similarly for fractions
- With 8 bits 0.0 to 0.99609375
- With 16 bits 0.0 to 0.9999847412109375

Arithmetic Operations

- Addition, Subtraction, Multiplication, Division
- Works the same way in base r as in base 10
- Just remember that only digits from 0 to $r-1$ can be used
- Let us do some examples.

Binary Addition

- Terms: Augend, Addend, Sum, Carry

001000	Carries
010101	Augend
+100110	Addend
<hr/>	<hr/>
111011	Sum

Binary Subtraction

- Minuend
- Subtrahend
- Difference
- Borrow

Binary Multiplication

- Multiplicand
- Multiplier
- Product

Binary Division

- Dividend
- Divisor
- Quotient
- Remainder

Decimal Codes

- Computers work in binary, people/humans work in decimal
- Computing must be in binary but input/output should be in decimal
- Binary Coded Decimal – BCD
- Uses 4 binary digits to represent each decimal digit as a separate entity
- Rules can even be defined to perform arithmetic on these BCD numbers directly

BCD Representation

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

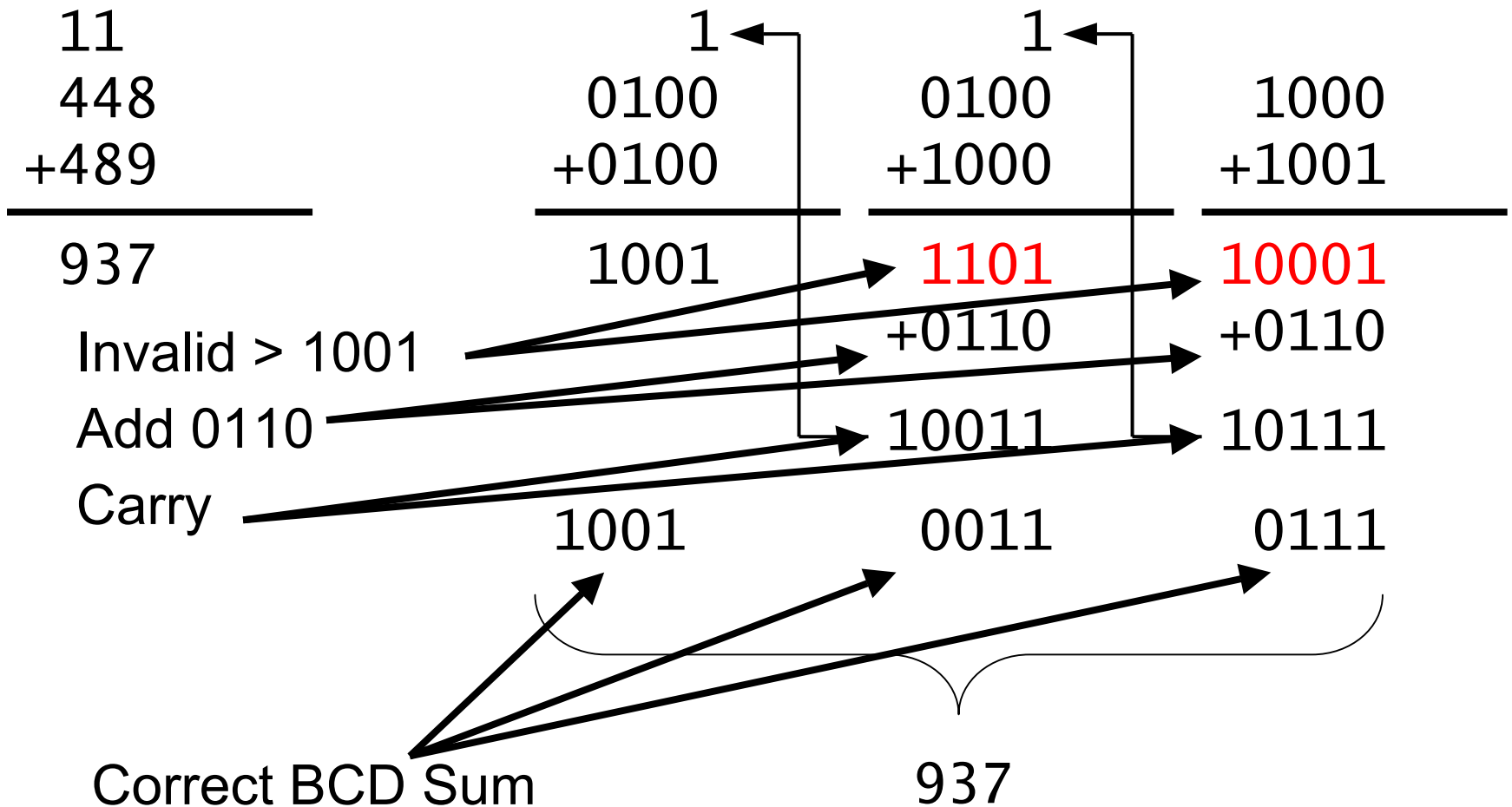
357 = 0011 0101 0111

Binary combinations 1010-1111 are not used in BCD and have no meaning

BCD Addition

- When adding the BCD representations of two decimal numbers add each BCD digit as per normal arithmetic
- If the result is between 0000 and 1001 (inclusive) the BCD digit is the correct sum of the decimal digits in BCD
- If the result is greater than 1010 then 0110 (i.e. decimal 6) needs to be added to the result to make the result a valid BCD digit and to produce a carry

BCD Addition Example



Alphanumeric Codes

- Why?
- What types of information has to be represented
 - Letters of the alphabet
 - Symbols such as %, &, \$, #, @, etc.
 - Control characters – what are these?
- How to estimate the number of bits the code would need?
- Evolution – EBCDIC, ASCII, Unicode

ASCII

- ASCII - American Standard Code for Information Interchange
- 7 bit code
- 128 valid codes ($2^7=128$)
- 94 printable characters, 34 non-printable characters
- Extended ASCII

American Standard Code for Information Interchange (ASCII)

B ₄ B ₃ B ₂ B ₁	B ₇ B ₆ B ₅							
	000	001	010	011	100	101	110	111
0000	NULL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	\$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(8	H	X	h	x
1001	HT	EM)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[k	{
1100	FF	FS	,	<	L	\	l	
1101	CR	GS	-	=	M]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

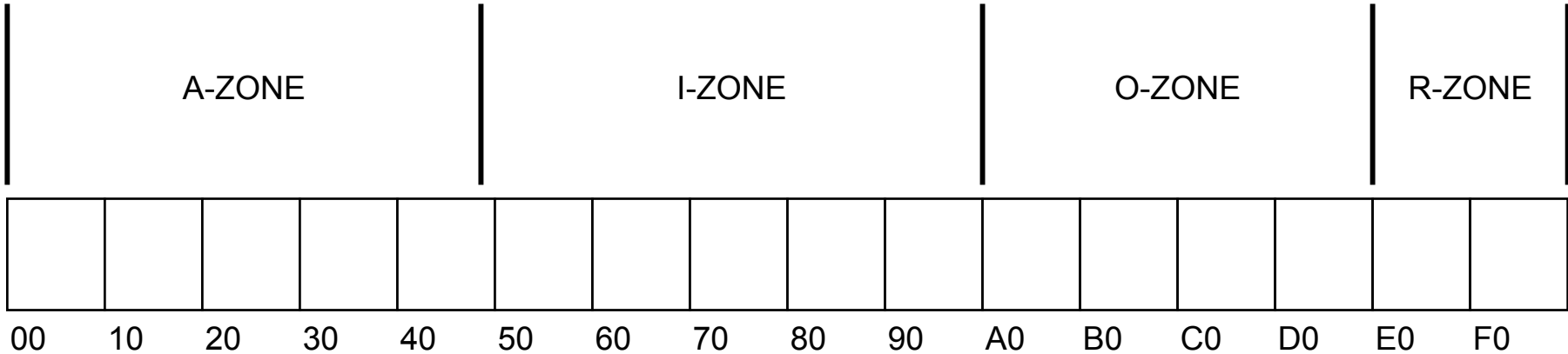
Parity Bit

- Error Detection vs. Error Correction
- Even Parity vs. Odd Parity
- How to implement parity?

Unicode

- 16 bit code – allows for 65536 unique codes
- ISO/IEC Standard
- Incorporates characters for most of the worlds languages
- Can be represented as four hexadecimal digits
- ASCII is included in Unicode – just append $(00)_{16}$ to the ASCII character – first 95 characters in Unicode are just ASCII characters
- Structure of the Unicode

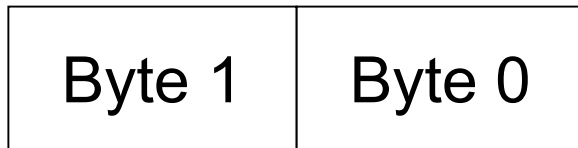
Structure of the Unicode



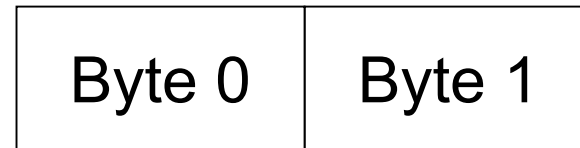
Major Zones for Unicode Assignment

Byte Ordering

- Byte ordering issues
 - Big-endian
 - Little-endian
- BYTE ORDER MARK – $(\text{FEFF})_{16}$ is placed before string of Unicode characters to indicate to the receiver the order of bytes received. $(\text{FFFE})_{16}$ is an invalid Unicode character so the receiver knows it has to swap the bytes.



Little Endian



Big Endian