Data Representation – Data Types

- Registers contain either data or control information
- Control information is a bit or group of bits used to specify the sequence of command signals needed for data manipulation
- Data are numbers and other binary-coded information that are operated on
- Possible data types in registers:
  - Numbers used in computations
  - Letters of the alphabet used in data processing
  - Other discrete symbols used for specific purposes
- All types of data, except binary numbers, are represented in binary-coded form

- A number system of *base*, or *radix, r* is a system that uses distinct symbols for *r* digits
- Numbers are represented by a string of digit symbols
- The string of digits 724.5 represents the quantity

$$7 \text{ x } 10^2 + 2 \text{ x } 10^1 + 4 \text{ x } 10^0 + 5 \text{ x } 10^{-1}$$

- The string of digits 101101 in the binary number system represents the quantity

$$1 \text{ x } 2^5 + 0 \text{ x } 2^4 + 1 \text{ x } 2^3 + 1 \text{ x } 2^2 + 0 \text{ x } 2^1 + 1 \text{ x } 2^0 = 45$$

- $(101101)_2 = (45)_{10}$
- We will also use the octal (radix 8) and hexidecimal (radix 16) number systems

$$(736.4)_8 = 7 \text{ x } 8^2 + 3 \text{ x } 8^1 + 6 \text{ x } 8^0 + 4 \text{ x } 8^{-1} = (478.5)_{10}$$

$$(F3)_{16} = F \text{ x } 16^1 + 3 \text{ x } 16^0 = (243)_{10}$$

- Conversion from decimal to radix *r* system is carried out by separating the number into its integer and fraction parts and converting each part separately
- Divide the integer successively by *r* and accumulate the remainders
- Multiply the fraction successively by *r* until the fraction becomes zero

Figure 3-1 Conversion of decimal 41.6875 into binary.

Integer = 41

41	
20	1
10	0
5	0
2	1
1	0
0	1

$(41)_{10} = (101001)_2$

Fraction = 0.6875

$$\begin{array}{r} 0.6875 \\ \underline{2} \\ 1.3750 \\ \underline{x\ \ 2} \\ 0.7500 \\ \underline{x\ \ 2} \\ 1.5000 \\ \underline{x\ \ 2} \\ 1.0000 \end{array}$$

$(0.6875)_{10} = (0.1011)_2$

$(41.6875)_{10} = (101001.1011)_2$

- Each octal digit corresponds to three binary digits
- Each hexadecimal digit corresponds to four binary digits
- Rather than specifying numbers in binary form, refer to them in octal or hexadecimal and reduce the number of digits by 1/3 or ¼, respectively

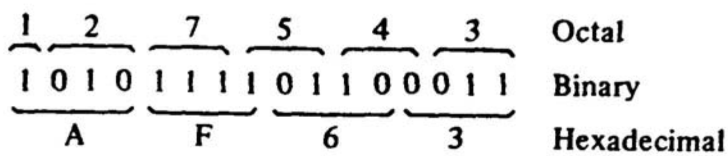

Figure 3-2 Binary, octal, and hexadecimal conversion.

TABLE 3-1 Binary-Coded Octal Numbers

Octal number	Binary-coded octal	Decimal equivalent	
0	000	0	↑
1	001	1	
2	010	2	Code
3	011	3	for one
4	100	4	octal
5	101	5	digit
6	110	6	
7	111	7	↓
10	001 000	8	
11	001 001	9	
12	001 010	10	
24	010 100	20	
62	110 010	50	
143	001 100 011	99	
370	011 111 000	248	

TABLE 3-2 Binary-Coded Hexadecimal Numbers

Hexadecimal number	Binary-coded hexadecimal	Decimal equivalent	
0	0000	0	
1	0001	1	
2	0010	2	
3	0011	3	
4	0100	4	
5	0101	5	
6	0110	6	Code
7	0111	7	for one
8	1000	8	hexadecimal
9	1001	9	digit
A	1010	10	
B	1011	11	
C	1100	12	
D	1101	13	
E	1110	14	
F	1111	15	
14	0001 0100	20	
32	0011 0010	50	
63	0110 0011	99	
F8	1111 1000	248	

- A binary code is a group of *n* bits that assume up to 2^n distinct combinations
- A four bit code is necessary to represent the ten decimal digits – 6 are unused
- The most popular decimal code is called *binary-coded decimal* (BCD)
- BCD is different from converting a decimal number to binary
- For example 99, when converted to binary, is 1100011
- 99 when represented in BCD is 1001 1001

TABLE 3-3 Binary-Coded Decimal (BCD) Numbers

Decimal number	Binary-coded decimal (BCD) number	
0	0000	↑
1	0001	
2	0010	
3	0011	Code
4	0100	for one
5	0101	decimal
6	0110	digit
7	0111	
8	1000	
9	1001	↓
10	0001 0000	
20	0010 0000	
50	0101 0000	
99	1001 1001	
248	0010 0100 1000	

- The standard alphanumeric binary code is ASCII
- This uses seven bits to code 128 characters
- Binary codes are required since registers can hold binary information only

TABLE 3-4 American Standard Code for Information Interchange (ASCII)

Character	Binary code	Character	Binary code
A	100 0001	0	011 0000
B	100 0010	1	011 0001
C	100 0011	2	011 0010
D	100 0100	3	011 0011
E	100 0101	4	011 0100
F	100 0110	5	011 0101
G	100 0111	6	011 0110
H	100 1000	7	011 0111
I	100 1001	8	011 1000
J	100 1010	9	011 1001
K	100 1011		
L	100 1100		
M	100 1101	space	010 0000
N	100 1110	.	010 1110
O	100 1111	(	010 1000
P	101 0000	+	010 1011
Q	101 0001	$	010 0100
R	101 0010	*	010 1010
S	101 0011	)	010 1001
T	101 0100	−	010 1101
U	101 0101	/	010 1111
V	101 0110	,	010 1100
W	101 0111	=	011 1101
X	101 1000		
Y	101 1001		
Z	101 1010		

– Complements

- Complements are used in digital computers for simplifying subtraction and logical manipulation
- Two types of complements for each base r system: r's complement and $(r – 1)$'s complement
- Given a number N in base r having n digits, the $(r – 1)$'s complement of N is defined as $(r^n – 1) – N$

- For decimal, the 9's complement of N is $(10^n – 1) – N$
- The 9's complement of 546700 is 999999 – 546700 = 453299

- The 9’s complement of 453299 is 999999 – 453299 = 546700
- For binary, the 1’s complement of N is $(2^n - 1) - N$
- The 1’s complement of 1011001 is 1111111 – 1011001 = 0100110
- The 1’s complement is the true complement of the number – just toggle all bits

- The r’s complement of an n-digit number N in base r is defined as $r^n - N$
- This is the same as adding 1 to the $(r - 1)$’s complement
- The 10’s complement of 2389 is 7610 + 1 = 7611
- The 2’s complement of 101100 is 010011 + 1 = 010100

- Subtraction of unsigned n-digit numbers: $M - N$
  - Add M to the r’s complement of N – this results in
    $M + (r^n - N) = M - N + r^n$
  - If $M \geq N$, the sum will produce an end carry r^n which is discarded
  - If $M < N$, the sum does not produce an end carry and is equal to $r^n - (N - M)$, which is the r’s complement of $(N - M)$. To obtain the answer in a familiar form, take the r’s complement of the sum and place a negative sign in front.

Example: 72532 – 13250 = 59282. The 10’s complement of 13250 is 86750.

M	= 72352
10’s comp. of N	= +86750
Sum	= 159282
Discard end carry	= -100000
Answer	= 59282

Example for M < N: 13250 – 72532 = -59282

M	= 13250
10’s comp. of N	= +27468
Sum	= 40718
No end carry	
Answer	= -59282 (10’s comp. of 40718)

Example for X = 1010100 and Y = 1000011

X	= 1010100
2’s comp. of Y	= +0111101
Sum	= 10010001
Discard end carry	= -10000000
Answer X – Y	= 0010001
Y	= 1000011
2’s comp. of X	= +0101100
Sum	= 1101111

No end carry
Answer = -0010001 (2's comp. of 1101111)

– Fixed-Point Representation

- Positive integers and zero can be represented by unsigned numbers
- Negative numbers must be represented by signed numbers since + and – signs are not available, only 1's and 0's are
- Signed numbers have msb as 0 for positive and 1 for negative – msb is the sign bit
- Two ways to designate binary point position in a register
    - Fixed point position
    - Floating-point representation
- Fixed point position usually uses one of the two following positions
    - A binary point in the extreme left of the register to make it a fraction
    - A binary point in the extreme right of the register to make it an integer
    - In both cases, a binary point is not actually present
- The floating-point representations uses a second register to designate the position of the binary point in the first register

- When an integer is positive, the msb, or sign bit, is 0 and the remaining bits represent the magnitude
- When an integer is negative, the msb, or sign bit, is 1, but the rest of the number can be represented in one of three ways
    - Signed-magnitude representation
    - Signed-1's complement representation
    - Signed-2's complement representation

- Consider an 8-bit register and the number +14
    - The only way to represent it is 00001110
- Consider an 8-bit register and the number –14
    - Signed magnitude: 1 0001110
    - Signed 1's complement: 1 1110001
    - Signed 2's complement: 1 1110010
- Typically use signed 2's complement

- Addition of two signed-magnitude numbers follow the normal rules
    - If same signs, add the two magnitudes and use the common sign
    - Differing signs, subtract the smaller from the larger and use the sign of the larger magnitude
    - Must compare the signs and magnitudes and then either add or subtract
- Addition of two signed 2's complement numbers does not require a comparison or subtraction – only addition and complementation
    - Add the two numbers, including their sign bits
    - Discard any carry out of the sign bit position
    - All negative numbers must be in the 2's complement form
    - If the sum obtained is negative, then it is in 2's complement form

+6	00000110	-6	11111010
+13	00001101	+13	00001101
+19	00010011	+7	00000111

+6	00000110	-6	11111010
-13	11110011	-13	11110011
-7	11111001	-19	11101101

- Subtraction of two signed 2's complement numbers is as follows
  - Take the 2's complement form of the subtrahend (including sign bit)
  - Add it to the minuend (including the sign bit)
  - A carry out of the sign bit position is discarded

- An *overflow* occurs when two numbers of *n* digits each are added and the sum occupies *n* + 1 digits
- Overflows are problems since the width of a register is finite
- Therefore, a flag is set if this occurs and can be checked by the user
- Detection of an overflow depends on if the numbers are signed or unsigned
- For unsigned numbers, an overflow is detected from the end carry out of the msb
- For addition of signed numbers, an overflow cannot occur if one is positive and one is negative – both have to have the same sign
- An overflow can be detected if the carry into the sign bit position and the carry out of the sign bit position are not equal

+70	0 1000110	-70	1 0111010
+80	0 1010000	-80	1 0110000
+150	1 0010110	-150	0 1101010

- The representation of decimal numbers in registers is a function of the binary code used to represent a decimal digit
- A 4-bit decimal code requires four flip-flops for each decimal digit
- This takes much more space than the equivalent binary representation and the circuits required to perform decimal arithmetic are more complex
- Representation of signed decimal numbers in BCD is similar to the representation of signed numbers in binary
- Either signed magnitude or signed complement systems
- The sign of a number is represented with four bits
  - 0000 for +
  - 1001 for –
- To obtain the 10's complement of a BCD number, first take the 9's complement and then add one to the least significant digit

- Example: (+375) + (-240) = +135

0 375	$(0000\ 0011\ 0111\ 1010)_{BCD}$
+9 760	$(1001\ 0111\ 0110\ 0000)_{BCD}$
0 135	$(0000\ 0001\ 0011\ 0101)_{BCD}$

– Floating-Point Representation

- The floating-point representation of a number has two parts
- The first part represents a signed, fixed-point number – the *mantissa*
- The second part designates the position of the binary point – the *exponent*
- The mantissa may be a fraction or an integer
- Example: the decimal number +6132.789 is
    - Fraction: +0.6123789
    - Exponent: +04
    - Equivalent to $+0.6132789 \times 10^{+4}$
- A floating-point number is always interpreted to represent $m \times r^e$
- Example: the binary number +1001.11 (with 8-bit fraction and 6-bit exponent)
    - Fraction: 01001110
    - Exponent: 000100
    - Equivalent to $+(.1001110)_2 \times 2^{+4}$
- A floating-point number is said to be *normalized* if the most significant digit of the mantissa is nonzero
- The decimal number 350 is normalized, 00350 is not
- The 8-bit number 00011010 is not normalized
- Normalize it by fraction = 11010000 and exponent = -3
- Normalized numbers provide the maximum possible precision for the floating-point number

– Other Binary Codes

- Digital systems can process data in discrete form only
- Continuous, or analog, information is converted into digital form by means of an analog-to-digital converter
- The reflected binary or *Gray code*, is sometimes used for the converted digital data
- The Gray code changes by only one bit as it sequences from one number to the next
- Gray code counters are sometimes used to provide the timing sequences that control the operations in a digital system

TABLE 3-5 4-Bit Gray Code

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

- Binary codes for decimal digits require a minimum of four bits
- Other codes besides BCD exist to represent decimal digits

TABLE 3-6 Four Different Binary Codes for the Decimal Digit

Decimal digit	BCD 8421	2421	Excess-3	Excess-3 gray
0	0000	0000	0011	0010
1	0001	0001	0100	0110
2	0010	0010	0101	0111
3	0011	0011	0110	0101
4	0100	0100	0111	0100
5	0101	1011	1000	1100
6	0110	1100	1001	1101
7	0111	1101	1010	1111
8	1000	1110	1011	1110
9	1001	1111	1100	1010
Unused bit combi- nations	1010	0101	0000	0000
	1011	0110	0001	0001
	1100	0111	0010	0011
	1101	1000	1101	1000
	1110	1001	1110	1001
	1111	1010	1111	1011

- The 2421 code and the excess-3 code are both *self-complementing*
- The 9's complement of each digit is obtained by complementing each bit in the code
- The 2421 code is a *weighted code*
- The bits are multiplied by indicated weights and the sum gives the decimal digit
- The excess-3 code is obtained from the corresponding BCD code added to 3

– Error Detection Codes

- Transmitted binary information is subject to noise that could change bits 1 to 0 and vice versa
- An *error detection code* is a binary code that detects digital errors during transmission
- The detected errors cannot be corrected, but can prompt the data to be retransmitted
- The most common error detection code used is the *parity bit*

- A parity bit is an extra bit included with a binary message to make the total number of 1's either odd or even

TABLE 3-7 Parity Bit Generation

Message *xyz*	*P*(odd)	*P*(even)
000	1	0
001	0	1
010	0	1
011	1	0
100	0	1
101	1	0
110	1	0
111	0	1

- The P(odd) bit is chosen to make the sum of 1's in all four bits odd
- The even-parity scheme has the disadvantage of having a bit combination of all 0's
- Procedure during transmission:
  - At the sending end, the message is applied to a *parity generator*
  - The message, including the parity bit, is transmitted
  - At the receiving end, all the incoming bits are applied to a *parity checker*
  - Any odd number of errors are detected

- Parity generators and checkers are constructed with XOR gates (odd function)
- An odd function generates 1 iff an odd number if input variables are 1

Figure 3-3 Error detection with odd parity bit.