CSE213

Microcontroller Programming

Introduction to the Microprocessor and Computer

-2-

Introduction

- How data is stored in memory
- Numeric data representations
- Alphanumeric representations

Chapter Objectives (*cont.*)

Upon completion of this chapter, you will be able to:

- Convert between binary, decimal, and hexadecimal numbers.
- Differentiate and represent numeric and alphabetic information as integers, floating-point, BCD, and ASCII data.

The Microprocessor

- Called the CPU (**central processing unit**).
- The controlling element in a computer system.
- Controls memory and I/O through connections called buses.
  - buses select an I/O or memory device, transfer data between I/O devices or memory and the microprocessor, control I/O and memory systems
- Memory and I/O controlled via instructions stored in memory, executed by the microprocessor.

- Microprocessor performs three main tasks:
  - data transfer between itself and the memory or I/O systems
  - simple arithmetic and logic operations
  - program flow via simple decisions
- Power of the microprocessor is capability to execute billions of millions of instructions per second from a program or software (**group of instructions**) stored in the memory system.
  - stored programs make the microprocessor and computer system very powerful devices

- Another powerful feature is the ability to make simple decisions based upon numerical facts.
  - a microprocessor can decide if a number is zero, positive, and so forth
- These decisions allow the microprocessor to modify the program flow, so programs appear to think through these simple decisions.

Simple arithmetic and logic operations

Operation	*Comment*
Addition	
Subtraction	
Multiplication	
Division	
AND	Logical multiplication
OR	Logic addition
NOT	Logical inversion
NEG	Arithmetic inversion
Shift	
Rotate	

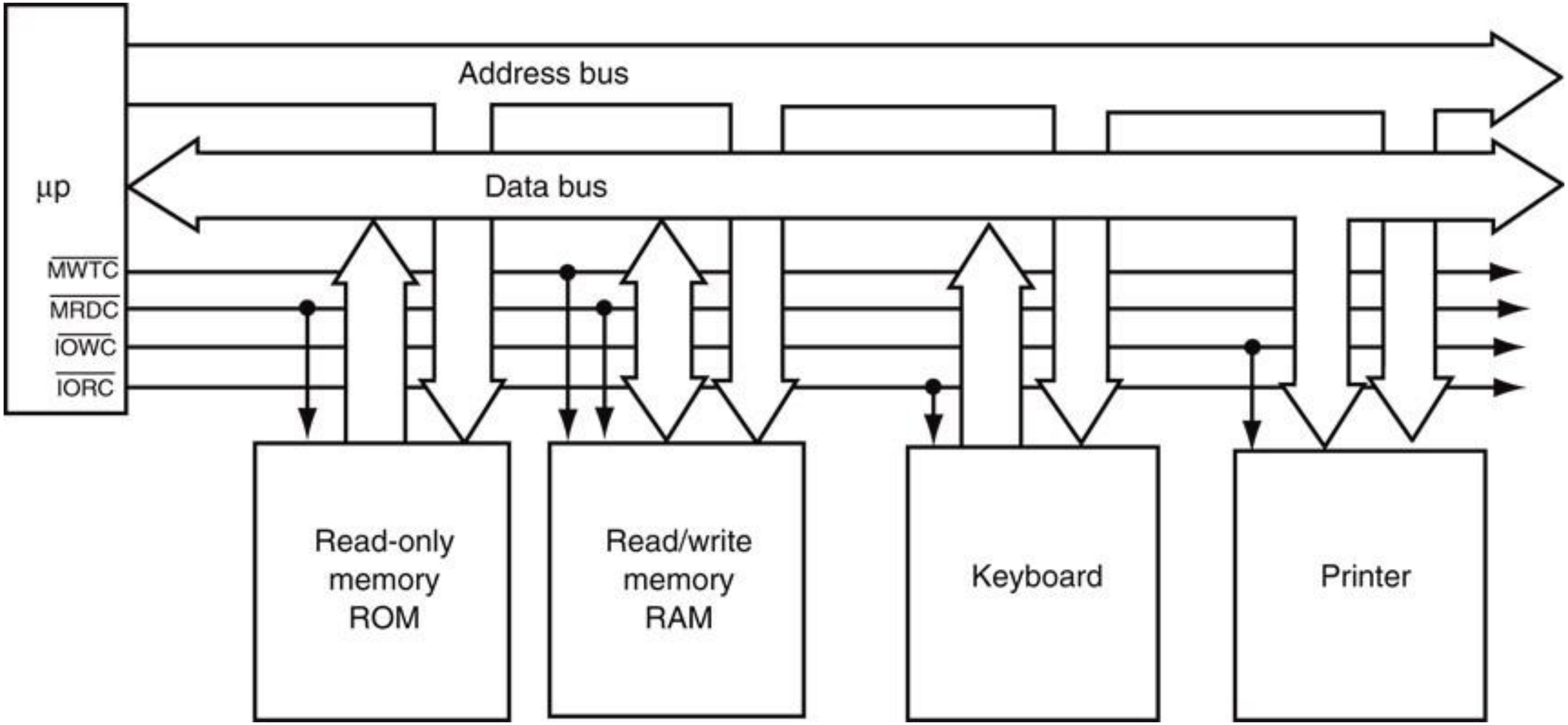
Decisions found in the 8086 through Core2 microprocessors

Decision	*Comment*
Zero	Test a number for zero or not-zero
Sign	Test a number for positive or negative
Carry	Test for a carry after addition or a borrow after subtraction
Parity	Test a number for an even or an odd number of ones
Overflow	Test for an overflow that indicates an invalid result after a signed addition or a signed subtraction

Buses

- A common group of wires that interconnect components in a computer system.
- Transfer address, data, & control information between microprocessor, memory and I/O.
- Three buses exist for this transfer of information: address, data, and control.
- Figure 1–12 shows how these buses interconnect various system components.

Figure 1–12 The block diagram of a computer system showing the address, data, and control bus structure.



- The address bus requests a memory location from the memory or an I/O location from the I/O devices.
  - if I/O is addressed, the address bus contains a 16-bit I/O address from 0000H through FFFFH.
  - if memory is addressed, the bus contains a memory address, varying in width by type of microprocessor (20-bits for 8086).
- 64-bit extensions to Pentium provide 40 address pins, allowing up to 1T byte of memory to be accessed.

- The data bus transfers information between the microprocessor and its memory and I/O address space.
- Data transfers vary in size, from 8 bits wide to 64 bits wide in various Intel microprocessors.
  - 8088 has an 8-bit data bus that transfers 8 bits of data at a time
  - 8086, 80286, 80386SL, 80386SX, and 80386EX transfer 16 bits of data
  - 80386DX, 80486SX, and 80486DX, 32 bits
  - Pentium through Core2 microprocessors transfer 64 bits of data

- Control bus lines select and cause memory or I/O to perform a read or write operation.
- In most computer systems, there are four control bus connections:
- $\overline{MRDC}$ (**memory read control**)
- $\overline{MWTC}$ (**memory write control**)
- $\overline{IORC}$ (**I/O read control**)
- $\overline{IOWC}$ (**I/O write control**).
- overbar indicates the control signal is active-low; (active when logic zero appears on control line)

- The microprocessor reads a memory location by sending the memory an address through the address bus.
- Next, it sends a memory read control signal to cause the memory to read data.
- Data read from memory are passed to the microprocessor through the data bus.
- Whenever a memory write, I/O write, or I/O read occurs, the same sequence ensues.

1–3 NUMBER SYSTEMS

- Use of a microprocessor requires working knowledge of numbering systems.
  - binary, decimal, and hexadecimal
- This section provides a background for these numbering systems.
- Conversions are described.
  - decimal and binary
  - decimal and hexadecimal
  - binary and hexadecimal

Digits

- Before converting numbers between bases, digits of a number system must be understood.
- First digit in any numbering system is always zero.
- A decimal (base 10) number is constructed with 10 digits: 0 through 9.
- A base 8 (**octal**) number; 8 digits: 0 through 7.
- A base 2 (**binary**) number; 2 digits: 0 and 1.

- If the base exceeds 10, additional digits use letters of the alphabet, beginning with an A.
  - a base 12 number contains 11 digits: 0 through 9, followed by A for 10 and B for 11
- Note that a base 10 number does not contain a *10* digit.
  - a base 8 number does not contain an *8* digit
- Common systems used with computers are decimal, binary, and hexadecimal (base 16).
  - many years ago octal numbers were popular

Positional Notation

- Once digits are understood, larger numbers are constructed using positional notation.
  - position to the left of the units position is the tens position
  - left of tens is the hundreds position, and so forth
- An example is decimal number 132.
  - this number has 1 hundred, 3 tens, and 2 units
- Exponential powers of positions are critical for understanding numbers in other systems.

- Exponential value of each position:
  - the units position has a weight of 10^0, or 1
  - tens position a weight of 10^1, or 10
  - hundreds position has a weight of 10^2, or 100
- Position to the left of the radix (**number base**) point is always the units position in system.
  - called a decimal point only in the decimal system
  - position to left of the binary point always 2^0, or 1
  - position left of the octal point is 8^0, or 1
- Any number raised to its zero power is always one (1), or the units position.

- Position to the left of the units position always the number base raised to the first power.
  - in a decimal system, this is 10^1, or 10
  - binary system, it is 2^1, or 2
  - 11 decimal has a different value from 11 binary
- 11 decimal has different value from 11 binary.
  - decimal number composed of 1 ten, plus 1 unit; a value of 11 units
  - binary number 11 is composed of 1 two, plus 1 unit: a value of 3 decimal units
  - 11 octal has a value of 9 decimal units

- In the decimal system, positions right of the decimal point have negative powers.
  - first digit to the right of the decimal point has a value of 10^{-1}, or 0.1.
- In the binary system, the first digit to the right of the binary point has a value of 2^{-1}, or 0.5.
- Principles applying to decimal numbers also generally apply to those in any other system.
- To convert a binary number to decimal, add weights of each digit to form its decimal equivalent.

Conversion to Decimal Examples

Power	2^2		2^1		2^0		2^{-1}		2^{-2}		2^{-3}		
Weight	4		2		1		.5		.25		.125		
Number	1		1		0	.	1		0		1		
Numeric Value	4	+	2	+	0	+	.5	+	0	+	.125	=	6.625

$$(110.101)_2 = (6.625)_{10}$$

Power	6^1	6^0	6^{-1}
Weight	6	1	.167
Number	2	5	.2
Numeric Value	12 +	5 +	.333 = 17.333

$$(25.2)_6 = (17.333)_{10}$$

Power	8^2	8^1	8^0	8^{-1}
Weight	64	8	1	.125
Number	1	2	5	.7
Numeric Value	64 +	16 +	5 +	.875 = 85.875

$$(125.7)_8 = (85.875)_{10}$$

Power	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
Weight	16	8	4	2	1	.5	.25	.125	.0625
Number	1	1	0	1	1 .	0	1	1	1
Numeric Value	16 +	8 +	0 +	2 +	1 +	0 +	.25 +	.125 +	.0625 = 27.4375

$$(11011.0111)_2 = (27.4375)_{10}$$

Power	16^1	16^0	16^{-1}
Weight	16	1	.0625
Number	6	A .	C
Number Value	96 +	10 +	.75 = 106.75

$$(6A.C)_{16} = (106.75)_{10}$$

Conversion to Decimal

- To convert from any number base to decimal, determine the weights or values of each position of the number.
- Sum the weights to form the decimal equivalent.

Conversion from Decimal

- Conversions from decimal to other number systems more difficult to accomplish.
- To convert the whole number portion of a number to decimal, divide by 1 radix.
- To convert the fractional portion, multiply by the radix.

Whole Number Conversion from Decimal

- To convert a decimal whole number to another number system, divide by the radix and save remainders as significant digits of the result.
- An algorithm for this conversion:
  - divide the decimal number by the radix (number base)
  - save the remainder (first remainder is the least significant digit)
  - repeat steps 1 and 2 until the quotient is zero

- To convert 10 decimal to binary, divide it by 2.
  - the result is 5, with a remainder of 0
- First remainder is units position of the result.
  - in this example, a 0
- Next, divide the 5 by 2; result is 2, with a remainder of 1.
  - the 1 is the value of the twos (2^1) position
- Continue division until the quotient is a zero.
- The result is written as 1010_2 from the bottom to the top.

- To convert 10 decimal to base 8, divide by 8.
  - a 10 decimal is a 12 octal.
- For decimal to hexadecimal, divide by 16.
  - remainders will range in value from 0 through 15
  - any remainder of 10 through 15 is converted to letters A through F for the hexadecimal number
  - decimal number 109 converts to a 6DH

Whole Number Conversion from Decimal Examples

```
2) 10          remainder = 0
 2) 5          remainder = 1
  2) 2         remainder = 0
   2) 1        remainder = 1          result = 1010
      0
```

```
8) 10          remainder = 2
 8) 1          remainder = 1          result = 12
    0
```

```
16) 109        remainder = 13 (D)
 16)   6       remainder = 6          result = 6D
       0
```

Converting from a Decimal Fraction

- Conversion is accomplished with multiplication by the radix.
- Whole number portion of result is saved as a significant digit of the result.
  - fractional remainder again multiplied by the radix
  - when the fraction remainder is zero, multiplication ends
- Some numbers are never-ending (repetend).
  - a zero is never a remainder

- Algorithm for conversion from a decimal fraction:
  - multiply the decimal fraction by the radix (number base).
  - save the whole number portion of the result (even if zero) as a digit; first result is written immediately to the right of the radix point
  - repeat steps 1 and 2, using the fractional part of step 2 until the fractional part of step 2 is zero
- Same technique converts a decimal fraction into any number base.

Converting from a Decimal Fraction Examples

```
    .125
x      2
--------
   0.25      digit is 0

     .25
x      2
--------
     0.5     digit is 0

      .5
x      2
--------
     1.0     digit is 1     result = 0.001₂
```

```
    .125
x      8
--------
     1.0     digit is 1     result = 0.1₈
```

```
    .046875
x        16
-----------
       0.75    digit is 0

     .75
x     16
--------
    12.0       digit is 12(C)   result = 0.0C₁₆
```

Binary-Coded Hexadecimal

- **Binary-coded hexadecimal** (BCH) is a hexadecimal number written each digit is represented by a 4-bit binary number.
- BCH code allows a binary version of a hexadecimal number to be written in a form easily converted between BCH and hexadecimal.
- Hexadecimal represented by converting digits to BCH code with a space between each digit.

BCH Table and Example

Hexadecimal Digit	*BCH Code*
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

2AC = 0010 1010 1100

1000 0011 1101 . 1110 = 83D.E

Complements

- At times, data are stored in complement form to represent negative numbers.
- Two systems used to represent negative data:
  - **radix**
  - **radix – 1** complement (earliest - 1's complement)

```
  1111 1111
- 0100 1100
  ---------
  1011 0011
```

radix–1 complement

```
  1111 1111
- 0100 1000
  ---------
  1011 0111   (one's complement)
+         1
  ---------
  1011 1000   (two's complement)
```

radix complement

1–4 COMPUTER DATA FORMATS

- Successful programming requires a precise understanding of data formats.
- Commonly, data appear as ASCII, Unicode, BCD, signed and unsigned integers, and floating-point numbers (real numbers).
- Other forms are available but are not commonly found.

ASCII and Unicode Data

- ASCII (**American Standard Code for Information Interchange**) data represent alphanumeric characters in computer memory.
- Standard ASCII code is a 7-bit code.
  - eighth and most significant bit used to hold parity
- If used with a printer, most significant bits are 0 for alphanumeric printing; 1 for graphics.

ASCII Codes

TABLE 1–8 ASCII code.

	Second															
	X0	X1	X2	X3	X4	X5	X6	X7	X8	X9	XA	XB	XC	XD	XE	XF
First																
0X	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1X	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EMS	SUB	ESC	FS	GS	RS	US
2X	SP	!	“	#	$	%	&	’	(	)	*	+	,	—	.	/
3X	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4X	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5X	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6X	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7X	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	:::

- In PC, an extended ASCII character set is selected by placing 1 in the leftmost bit.
- Extended ASCII characters store:
  - some foreign letters and punctuation
  - Greek & mathematical characters
  - box-drawing & other special characters
- ASCII control characters perform control functions in a computer system.
  - clear screen, backspace, line feed, etc.
- Enter control codes through the keyboard.
  - hold the Control key while typing a letter

Extended ASCII Codes

TABLE 1–9 Extended ASCII code, as printed by the IBM ProPrinter.

First					Second											
	X0	X1	X2	X3	X4	X5	X6	X7	X8	X9	XA	XB	XC	XD	XE	XF
0X		☺	☻	♥	♦	♣	♠	•	◘	○	◙	♂	♀	♪	♫	☼
1X	►	◄	↕	‼	¶	§	▬	↨	↑	↓	→	←	∟	↔	▲	▼
8X	Ç	ü	é	â	ä	à	å	ç	ê	ë	è	ï	î	ì	Ä	Å
9X	É	æ	Æ	ô	ö	ò	û	ù	ÿ	Ö	Ü	¢	£	¥	₧	ƒ
AX	á	í	ó	ú	ñ	Ñ	ª	º	¿	⌐	¬	½	¼	¡	«	»
BX	░	▒	▓	│	┤	╡	╢	╖	╕	╣	║	╗	╝	╜	╛	┐
CX	└	┴	┬	├	─	┼	╞	╟	╚	╔	╩	╦	╠	═	╬	╧
DX	╨	╤	╥	╙	╘	╒	╓	╫	╪	┘	┌	█	▄	▌	▐	▀
EX	α	β	Γ	π	Σ	σ	µ	γ	Φ	Θ	Ω	δ	∞	φ	∈	∩
FX	≡	±	≥	≤	⌠	⌡	÷	≈	°	∙	·	√	^{n}	^{2}	■	

- Many Windows-based applications use the **Unicode** system to store alphanumeric data.
  - stores each character as 16-bit data
- Codes 0000H–00FFH are the same as standard ASCII code.
- Remaining codes, 0100H–FFFFH, store all special characters from many character sets.
- Allows software for Windows to be used in many countries around the world.
- For complete information on Unicode, visit: *http://www.unicode.org*

BCD (Binary-Coded Decimal) Data

- The range of a BCD digit extends from 0000_2 to 1001_2, or 0–9 decimal, stored in two forms:
- Stored in packed form:
  - packed BCD data stored as two digits per byte;
  - used for BCD addition and subtraction in the instruction set of the microprocessor
- Stored in unpacked form:
  - unpacked BCD data stored as one digit per byte
  - returned from a keypad or keyboard

Packed and Unpacked BCD

TABLE 1–10 Packed and unpacked BCD data.

Decimal	*Packed*		*Unpacked*		
12	0001 0010		0000 0001	0000 0010	
623	0000 0110	0010 0011	0000 0110	0000 0010	0000 0011
910	0000 1001	0001 0000	0000 1001	0000 0001	0000 0000

- Applications requiring BCD data are point-of-sales terminals.
  - also devices that perform a minimal amount of simple arithmetic
- If a system requires complex arithmetic, BCD data are seldom used.
  nadiren
  - there is no simple and efficient method of performing complex BCD arithmetic

TABLE 1–10 Packed and unpacked BCD data.

Decimal	*Packed*		*Unpacked*		
12	0001 0010		0000 0001	0000 0010	
623	0000 0110	0010 0011	0000 0110	0000 0010	0000 0011
910	0000 1001	0001 0000	0000 1001	0000 0001	0000 0000

Byte-Sized Data

- Stored as *unsigned* and *signed* integers.
- Difference in these forms is the weight of the leftmost bit position.
  - value 128 for the unsigned integer
  - *minus* 128 for the signed integer
- In signed integer format, the leftmost bit represents the sign bit of the number.
  - also a weight of *minus* 128

Figure 1–14 The unsigned and signed bytes illustrating the weights of each binary-bit position.

128	64	32	16	8	4	2	1 ← Binary weights
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Unsigned byte

-128	64	32	16	8	4	2	1 ← Binary weights
-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Signed byte

- Unsigned integers range 00H to FFH (0–255)
- Signed integers from –128 to 0 to + 127.
- Negative signed numbers represented in this way are stored in the two's complement form.
- Evaluating a signed number by using weights of each bit position is much easier than the act of two's complementing a number to find its value.
  - especially true in the world of calculators designed for programmers

(+/-) Conversion practice

- Whenever a number is two's complemented, its sign changes from negative to positive or positive to negative.
- For example, the number 0000 1000 is a +8. Its negative value (-8) is found by two's complementing the +8.
  1. one's complement the number
     - ✓ invert each bit of a number from zero to one or from one to zero.
  2. add a one to the one's complement.

```
+ 8 = 00001000
      11110111   (one's complement)
+            1
      --------
- 8 = 11111000   (two's complement)
```

(+/-) Simpler conversion practice

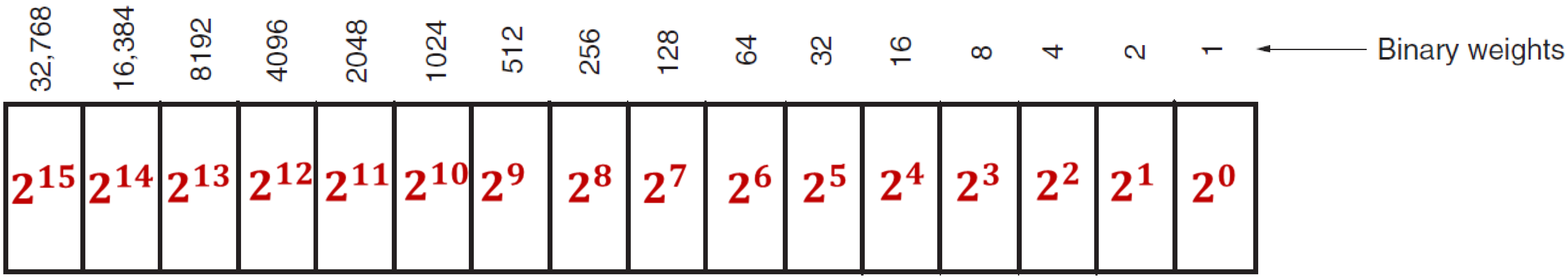
- Another, and probably simpler, technique for two's complementing a number starts with the rightmost digit.
  - Start by writing down the number from right to left.
  - Write the number exactly as it appears until the first one.
  - Write down the first one,
  - and then invert all bits to its left.

```
+8 = 00001000
         1000 (write number to first 1)
     1111     (invert the remaining bits)
-8 = 11111000
```

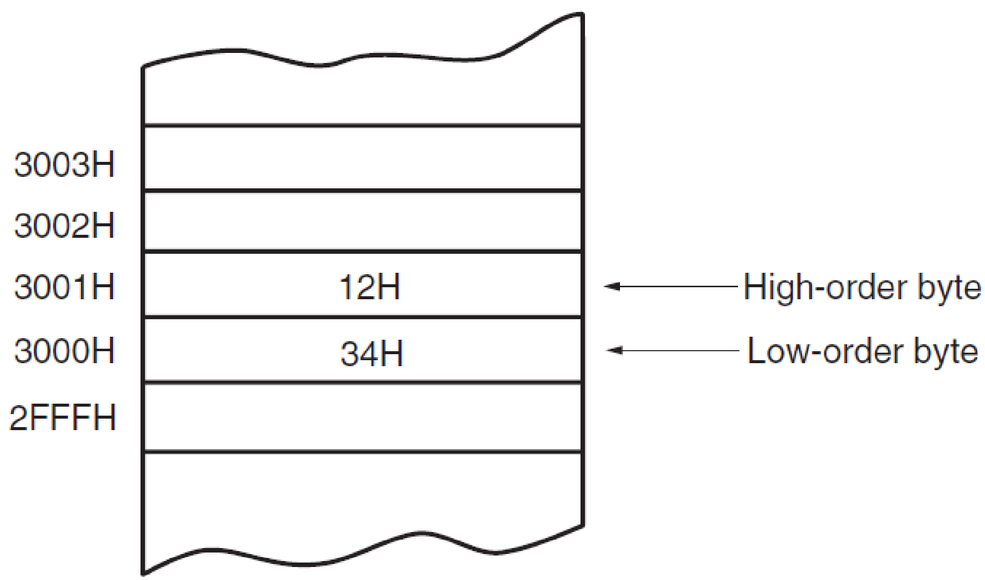
Word-Sized Data

- A word (16-bits) is formed with two bytes of data.
- The least significant byte always stored in the lowest-numbered memory location.
- Most significant byte is stored in the highest.
- This method of storing a number is called the **little-endian** format.

Figure 1–15 The storage format for a 16-bit word in (a) a register and (b) two bytes of memory.



(a) Unsigned word



(b) The contents of memory location 3000H and 3001H are the word 1234H.

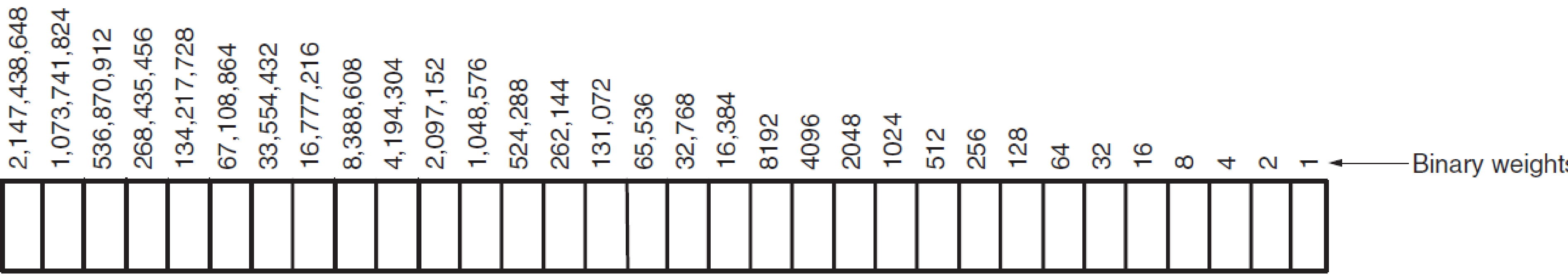
- Alternate method is called the **big-endian** format.
- Numbers are stored with the lowest location containing the most significant data.
- Not used with Intel microprocessors.
- The big-endian format is used with the Motorola family of microprocessors.

Doubleword-Sized Data

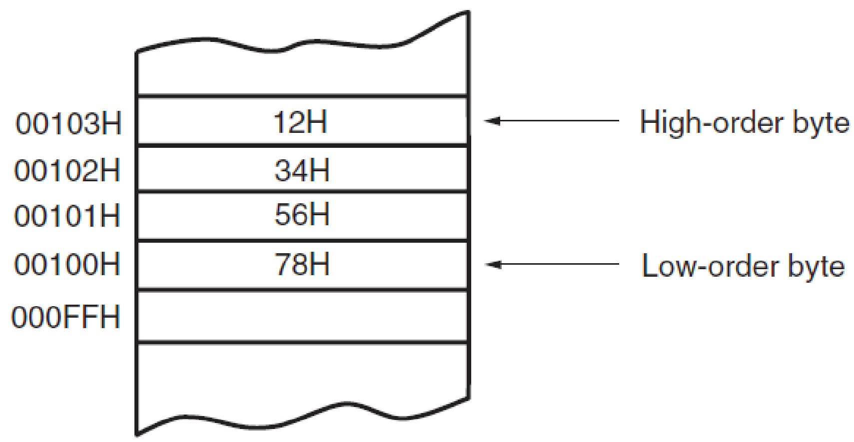
- **Doubleword-sized data** requires four bytes of memory because it is a 32-bit number.
  - appears as a product after a multiplication
  - also as a dividend before a division
- Define using the assembler directive **define doubleword(s)**, or **DD.**
  - also use the DWORD directive in place of DD

Doubleword-Sized Data

Figure 1–16 The storage format for a 32-bit word in (a) a register and (b) 4 bytes of memory.



(a) Unsigned doubleword

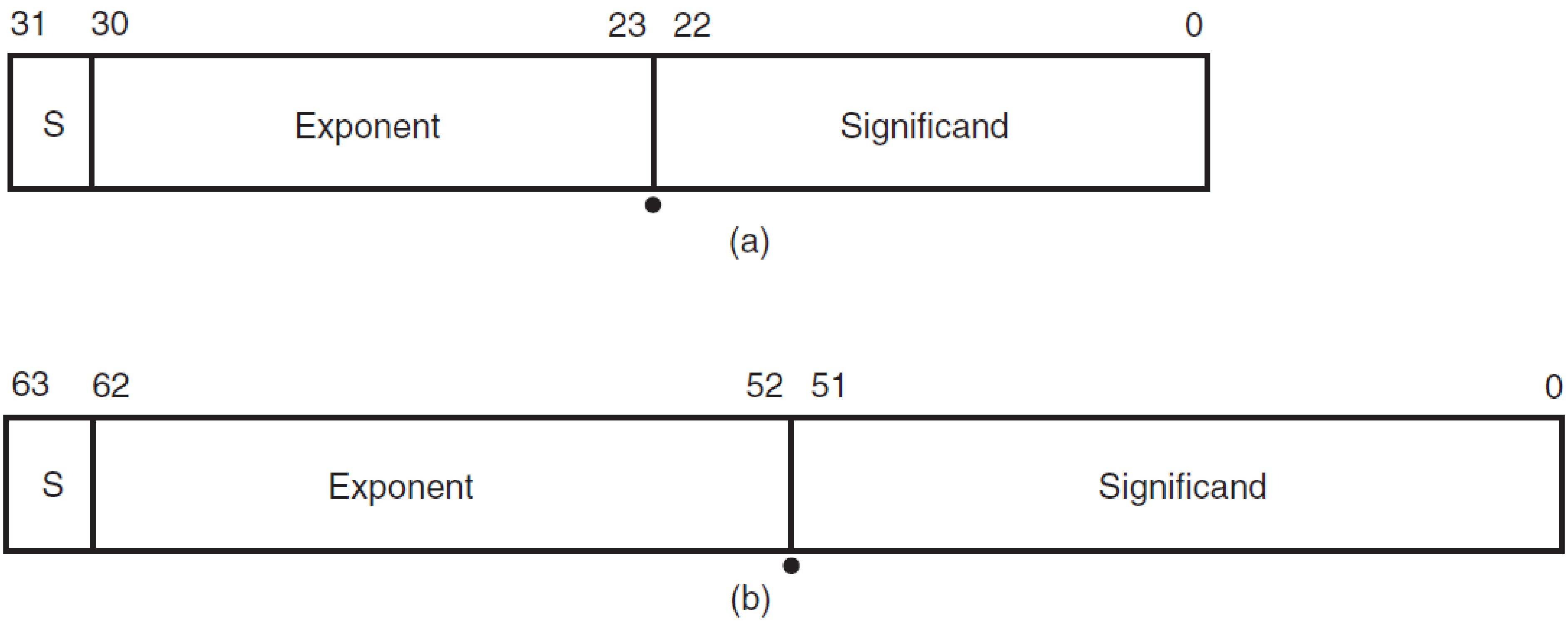


(b) The contents of memory location 00100H–00103H are the doubleword 12345678H.

Real Numbers

- Since many high-level languages use Intel microprocessors, real numbers are often encountered.
- A real, or a **floating-point number** contains two parts:
  – a mantissa, significand, or fraction
  – an exponent.
- A 4-byte number is called **single-precision.**
- The 8-byte form is called **double-precision.**

Figure 1–17 The floating-point numbers in (a) single-precision using a bias of 7FH and (b) double-precision using a bias of 3FFH.



- The assembler can be used to define real numbers in single- & double-precision forms:
  - use the DD directive for single-precision 32-bit numbers
  - use **define quadword(s)**, or DQ to define 64-bit double-precision real numbers
- Optional directives are REAL4, REAL8, and REAL10.
  - for defining single-, double-, and extended precision real numbers

Decimal	*Binary*	*Normalized*	*Sign*	*Biased Exponent*	*Mantissa*
+12	1100	1.1×2^3	0	10000010	10000000 00000000 00000000
−12	1100	1.1×2^3	1	10000010	10000000 00000000 00000000
+100	1100100	1.1001×2^6	0	10000101	10010000 00000000 00000000
−1.75	1.11	1.11×2^0	1	01111111	11000000 00000000 00000000
+0.25	0.01	1.0×2^{-2}	0	01111101	00000000 00000000 00000000
+0.0	0	0	0	00000000	00000000 00000000 00000000

```
                                   ;single-precision real numbers
                                   ;
0000  3F9DF3B6                     NUMB1    DD     1.234     ;define 1.234
0004  C1BB3333                     NUMB2    DD     -23.4     ;define -23.4
0008  43D20000                     NUMB3    REAL4  4.2E2     ;define 420
                                   ;
                                   ;double-precision real numbers
                                   ;
000C  405ED9999999999A             NUMB4    DQ     123.4     ;define 123.4
0014  C1BB333333333333             NUMB5    REAL8  -23.4     ;define -23.4
                                   ;
                                   ;Extended-precision real numbers
                                   ;
001C  4005F6CCCCCCCCCCCCCD         NUMB6    REAL10 123.4     ;define 123.4
```

```
//Single-precision real numbers
//
float Numb1 = 1.234;
float Numb2 = -23.4;
float Numb3 = 4.3e2;
//
//Double-precision real numbers
//
double Numb4 = 123.4;
double Numb5 = -23.4;
```

The Intel Microprocessors: 8086/8088, 80186/80188, 80286, 80386, 80486 Pentium, Pentium Pro Processor, Pentium II, Pentium, 4, and Core2 with 64-bit Extensions Architecture, Programming, and Interfacing, Eighth Edition

Barry B. Brey