There are 10 types of people in the world; Those who understand binary and those who don't.

02B Numbers Systems
CSC 230

Department of Computer Science
University of Victoria

Stl: Chapter 9; 10.1; 10.2; 10.3 (no multiplication / division), Appendix 12A (p. 447)

M&H: 2.1; 2.3; 2.4; 3.1.1; 3.1.2;

Integer Number Systems

Decimal
Base: 10
Digits: 0,1,2,3,4,5,6,7,8,9

Binary
Base: 2
Digits: 0,1

Octal
Base: 8
Digits: 0,1,2,3,4,5,6,7

Hexadecimal
Base: 16
Digits: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Small Trivial Example

7423 in decimal =

$$3 + 20 + 400 + 7000 =$$

$$\Rightarrow \quad 3 \times 1 + 2 \times 10 + 4 \times 100 + 7 \times 1000$$

$$\Rightarrow \quad 3 \times 10^0 + 2 \times 10^1 + 4 \times 10^2 + 7 \times 10^3$$

Integer Number Systems: Base 10 - Decimal

Positional Number Systems

$$\text{Integer} = \underline{D_{n-1}} \quad \underline{D_{n-2}} \quad \cdots \quad \underline{D_1} \quad \underline{D_0}$$

e.g. 7423 in decimal

Base 10:

$$\left(D_{n-1} \times 10^{n-1}\right) + \left(D_{n-2} \times 10^{n-2}\right) + \ldots + \left(D_1 \times 10^1\right) + \left(D_0 \times 10^0\right)$$

$$7423_{10} = \left(7 \times 10^3\right) + \left(4 \times 10^2\right) + \left(2 \times 10^1\right) + \left(3 \times 10^0\right)$$

Positional Number Systems

$$\text{Integer} = \underline{D_{n-1}} \quad \underline{D_{n-2}} \quad \cdots \quad \underline{D_1} \quad \underline{D_0}$$

Base 16:

e.g. 8254 in hexadecimal

$$\left(D_{n-1} \times 16^{n-1}\right) + \left(D_{n-2} \times 16^{n-2}\right) + \ldots + \left(D_1 \times 16^{1}\right) + \left(D_0 \times 16^{0}\right)$$

$$8254_{16} = \left(8 \times 16^3\right) + \left(2 \times 16^2\right) + \left(5 \times 16^1\right) + \left(4 \times 16^0\right)$$

$$= (8 \times 4096) + (2 \times 256) + (5 \times 16) + (4 \times 1) = 33{,}364_{10}$$

NOTE: we have converted from hex to decimal!

Integer Number Systems: Base 2 - Binary

Positional Number Systems

$$\text{Integer} = \underline{D_{n-1}} \quad \underline{D_{n-2}} \quad \cdots \quad \underline{D_1} \quad \underline{D_0}$$

e.g. 011011 in binary

Base 2:

$$\left(D_{n-1} \times 2^{n-1}\right) + \left(D_{n-2} \times 2^{n-2}\right) + \ldots + \left(D_1 \times 2^1\right) + \left(D_0 \times 2^0\right)$$

$$011011_2 =$$

$$= \left(0 \times 2^5\right) + \left(1 \times 2^4\right) + \left(1 \times 2^3\right) + \left(0 \times 2^2\right) + \left(1 \times 2^1\right) + \left(1 \times 2^0\right)$$

$$= (0 \times 32) + (1 \times 16) + (1 \times 8) + (0 \times 4) + (1 \times 2) + (1 \times 1) = 27_{10}$$

NOTE: we have converted from binary to decimal!

Weighted Positional Representation

BASE: **defines the range of values for digits (e.g. 0 – 9 for decimal; 0,1 for binary)**

GENERAL FORM AS AN n-BIT VECTOR:

B = BASE

$$\text{Integer Decimal Value} = \sum_{i=0}^{n-1} d_i \times B^i$$

d = DIGIT

i = position

$$\text{Decimal Value} = \sum_{i=-m}^{n-1} d_i \times B^i$$

Include fractions

Full example:

$$145.52_{10} = 1 \times 10^2 + 4 \times 10^1 + 5 \times 10^0 + 5 \times 10^{-1} + 2 \times 10^{-2}$$

$$= 100 + 40 + 5 + 0.5 + 0.02$$

Memorize This Table!

Binary	Decimal	Hexadecimal	
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	9	
1010	10	A	
1011	11	B	
1100	12	C	
1101	13	D	
1110	14	E	
1111	15	F	

Summary 1: Conversion from any Base "B" to Decimal (positive numbers)

➔ **Use the polynomial expansion in Base "B" as shown**

Base "B" gives the powers of the positional system

$$7423_{16} = \left(3 \times 16^{0}\right) + \left(2 \times 16^{1}\right) + \left(4 \times 16^{2}\right) + \left(7 \times 16^{3}\right)$$
$$= (3 \times 1) + (2 \times 16) + (4 \times 256) + (7 \times 4096)$$
$$= 29{,}731_{10}$$

$$11001011_{2} = \left(1 \times 2^{0}\right) + \left(1 \times 2^{1}\right) + \left(0 \times 2^{2}\right) + \left(1 \times 2^{3}\right) +$$
$$\left(0 \times 2^{4}\right) + \left(0 \times 2^{5}\right) + \left(1 \times 2^{6}\right) + \left(1 \times 2^{7}\right) = 203_{10}$$

Conversion from One Base to Another

Decimal to Base "B" for positive integers

1. **Repeated division by base *"B"***
2. **Collect remainders**
3. **Form result from right to left**

Example 1: from decimal to binary

$35_{10} = ???_2$

35/2	= 17	+ remainder	1
17/2	= 8	+ remainder	1
8/2	= 4	+ remainder	0
4/2	= 2	+ remainder	0
2/2	= 1	+ remainder	0
1/2	= 0	+ remainder	1

answer: 100011_2

Conversion from One Base to Another

Decimal to Base "B" for positive integers

1. **Repeated division by base *"B"***
2. **Collect remainders**
3. **Form result from right to left**

Example 2: from decimal to hexadecimal

$35_{10} = ???_{16}$

$35/16 = 2 + \text{remainder } 3$

$2/16 = 0 + \text{remainder } 2$

answer: 23_{16}

Conversion amongst binary, octal and hexadecimal is straightforward

- since $8 = 2^3$ it takes *3* bits to represent the 8 octal digits 0 .. 7
- Since $16 = 2^4$ it takes *4* bits to represent the 16 hex digits 0 .. F

from binary to hexadecimal : form groups of 4 bits from *right to left* and encode each group directly into a hexadecimal digit – *append leading zeroes if needed*

from binary to octal : form groups of 3 bits from *right to left* and encode each group directly into an octal digit – *append leading zeroes if needed*

$$0110\;0111_2 = 67_{16}$$

0 1 1 0	0 1 1 1
6	7

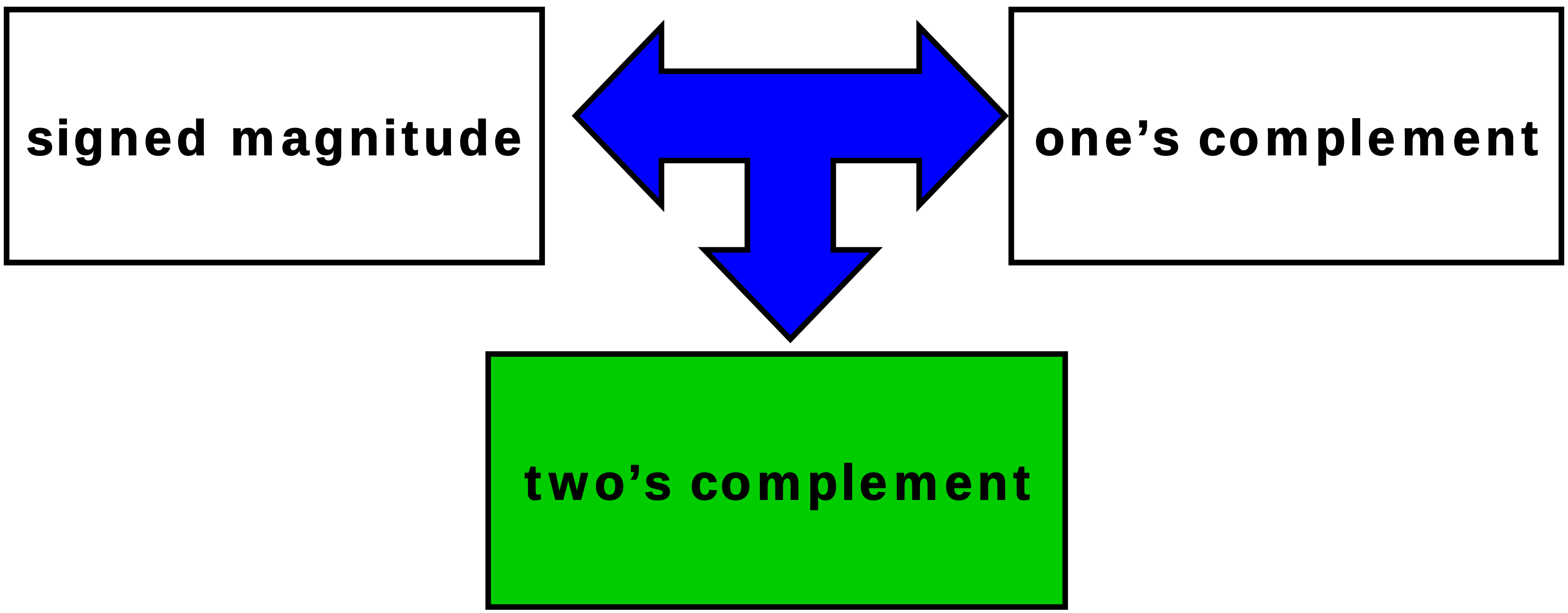
$(6\;7)_{16}$

$$0110\;0111_2 = 147_8$$

0 0 1	1 0 0	1 1 1
1	4	7

$(1\;4\;7)_8$

REPRESENTATION of Positive and Negative INTEGERS



SIGNED MAGNITUDE

- **Leading bit is the sign:** ***(not used in computation)***
  0 for +ve
  1 for –ve
- **other bits are the magnitude**

Ex. $0\,0\,0\,0\,1\,1\,0\,0_2 = +12_{10}$

$1\,0\,0\,0\,1\,1\,0\,0_2 = -12_{10}$

- **Requires separate add and subtract hardware**
- **There are 2 representations for "0": +ve and -ve**

TWO'S COMPLEMENT

Positive integers

✓ Positive integers are represented following regular conversion

✓ They all have the leading bit = 0

Example:

$+12_{10}$

using 8-bits in a 2's complement representation is =

$0000\ 1100_2$

➔ $0000\ 0000\ 0000\ 1100_2$ in 16 bits

Negative integers

✓ Negative integers are represented with the computation:

$$2^n – (\text{number-to-represent})$$

where *n* is the number of bits used

✓ They all have the leading bit = 1

Example:

-12_{10}

using 8-bits in a 2's complement representation is =

$2^8 – 12 = 244 = 1111\ 0100_2$

➔ in 16 bits = $1111\ 1111\ 1111\ 0100_2$

Converting from decimal to binary 2's complement – *Example 1 (negative integer)*

1. Convert the *absolute value* to binary
2. Complement (flip) all bits
3. Add 1 (that is, add 000.....001 in *n* bits binary)

Example: convert -12_{10} to 8 bits

1. Convert the absolute value $|12_{10}|$ to 8-bit binary as:

$0000\ 1100_2$

Note: this is the 2's complement representation of $+12_{10}$

2. Complement (flip) all bits as:

$0000\ 1100_2$ → $1111\ 0011_2$

3. Add +1 in binary as:

	1	1	1	1	0	0	1	1_2
+	0	0	0	0	0	0	0	1_2
=	1	1	1	1	0	1	0	0_2

→ this is the 2's complement for -12_{10}

Converting from decimal to binary 2's complement – Why does it work?

1. Convert the *absolute value* to binary
2. Complement (flip) all bits
3. Add 1 (that is, add 000…..001 in *n* bits binary)

Negative integers in 2's complement are represented with the general computation:

$$2^n - (\text{number-to-represented})$$

where *n* is the number of bits used

1's complement

Step 2:

Inverting all bits yields mathematically

➔ $[(2^n-1) - (\textit{number-to-represented})]$

Step 3:

Adding 1 yields

➔ $2^n - (\textit{number-to-represented})$

Summary 2: from decimal to binary

(1) Converting from decimal to binary 2's complement → positive integer	(2) Converting from decimal to binary 2's complement → negative integer
⇩	⬇
Regular conversion to base 2 by repeated division	1. Convert the *absolute value* to binary 2. Complement (flip) all bits 3. Add 1 (that is, add 000…..001 in *n* bits binary)

Summary 3: from binary to decimal

(3) Converting from binary 2's complement to decimal → positive integer

1. Convert using the regular expansion

(4) Converting from binary 2's complement to decimal (negative int)

1. Complement (flip) all bits
2. Add 1 (that is, add 000.....001 in *n* bits binary)
3. Convert to decimal → get the *absolute value*
4. Adjust sign

Avoid Confusion – Think it through!

In the two's complement representation of integers, a leading "1" bit denotes a negative value, but the remaining bits alone are *not* the magnitude

➔ the whole entity must be considered

➔ do not confuse with signed magnitude

Why do designers use 2's complement representation?

- it is a very efficient representation for +ve and -ve integers
- it avoids having 2 representations for "0"
- conversion between positive and negative is quite efficient and uniform
- only need one adder and no subtraction unit

Recap: Converting Decimal Integers to 2's Complement

positive decimal integer

convert to n-bit binary by division algorithm

negative decimal integer

get absolute value

convert to n-bit binary by division algorithm

flip all the bits and add 1

Example: convert $+9_{10}$ and -9_{10} to 8-bit binary using a Two's Complement Representation

9

0000 1001

-9

Start with |-9| = 9 and convert

0000 1001

flip all the bits = 1111 0110

add 1 = 1111 0111

Reversing the process - Converting from Binary 2's Complement to Decimal

$0000\ 0011_2$

if leftmost bit = 0,
then it must be a
positive integer,
➔ convert normally.

leftmost bit = 0
+ve number

$+3_{10}$

$1111\ 0011_2$

if leftmost bit = 1,
then it must be a
negative integer,
➔ do extra steps first

leftmost bit = 1, then -ve number

flip bits = 0000 1100

add 1 = 0000 1101

13

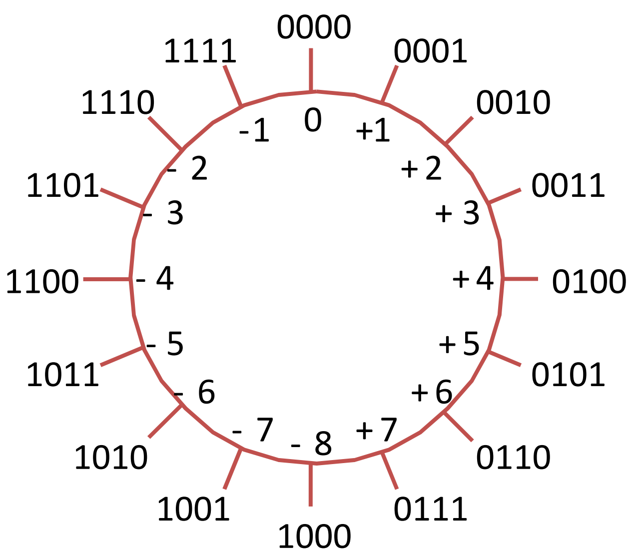
Original number must be -13

Table to memorize:

4 bits : { +7, -8} = range

In general the range for n bits is:

$$\left\{-2^{n-1},\ 2^{n-1}-1\right\}$$



0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	-8
1	0	0	1	-7
1	0	1	0	-6
1	0	1	1	-5
1	1	0	0	-4
1	1	0	1	-3
1	1	1	0	-2
1	1	1	1	-1

Quick Quiz

- What is 5 in 4-bit binary? 0101

- What is -5 as a 4-bit 2's complement?

  1011

DATA Representation – which CODE are we using

Code **- when a piece of information is represented by a particular pattern of symbols (bits in our case).**

There are many codes used in computing in addition to the number representations discussed earlier:

BCD - binary coded decimal as a direct representation of decimal digits

ASCII - American Standard Code for Information Interchange - used for character data

Parity - simple error detection using in serial data transmission

Binary Coded Decimal (BCD) – for the 10 decimal digits 0,1,…,9

A decimal digit is coded as 4 bits as follows:

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

Used in calculators and often in devices where display of decimal information is a primary function e.g. clock or VCR.

Also used extensively in business computing e.g. COBOL programs.

7-bit ASCII – The most commonly used code for representing character data

	0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1 XON	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3 XOFF	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	del

Parity – the extra bits for error detection

A parity bit can be added to a unit of information e.g. a character or a memory word

EVEN parity **➔ the parity bit is set to either 0 or 1 such that the total number of bits equal to 1 in the information unit, *including the parity bit,* is even**

ODD parity **➔ the parity bit is set to either 0 or 1 such that the total number of bits equal to 1 in the information unit, *including the parity bit,* is odd**

- ❑ **Single bit parity can detect any odd number of bits in error**
- ❑ **One can not tell which bits are in error ➔ no error correction**

Example

The 7-bit ASCII code for "D"	**=**	**100 0100**
with EVEN parity bit	**➔**	**0100 0100**
with ODD parity bit	**➔**	**1100 0100**

Let’s Review the Nomenclature

BIT : unit of information storage - value of 0 or 1

BYTE : collection of 8 bits - unit of "character" representation and small integers

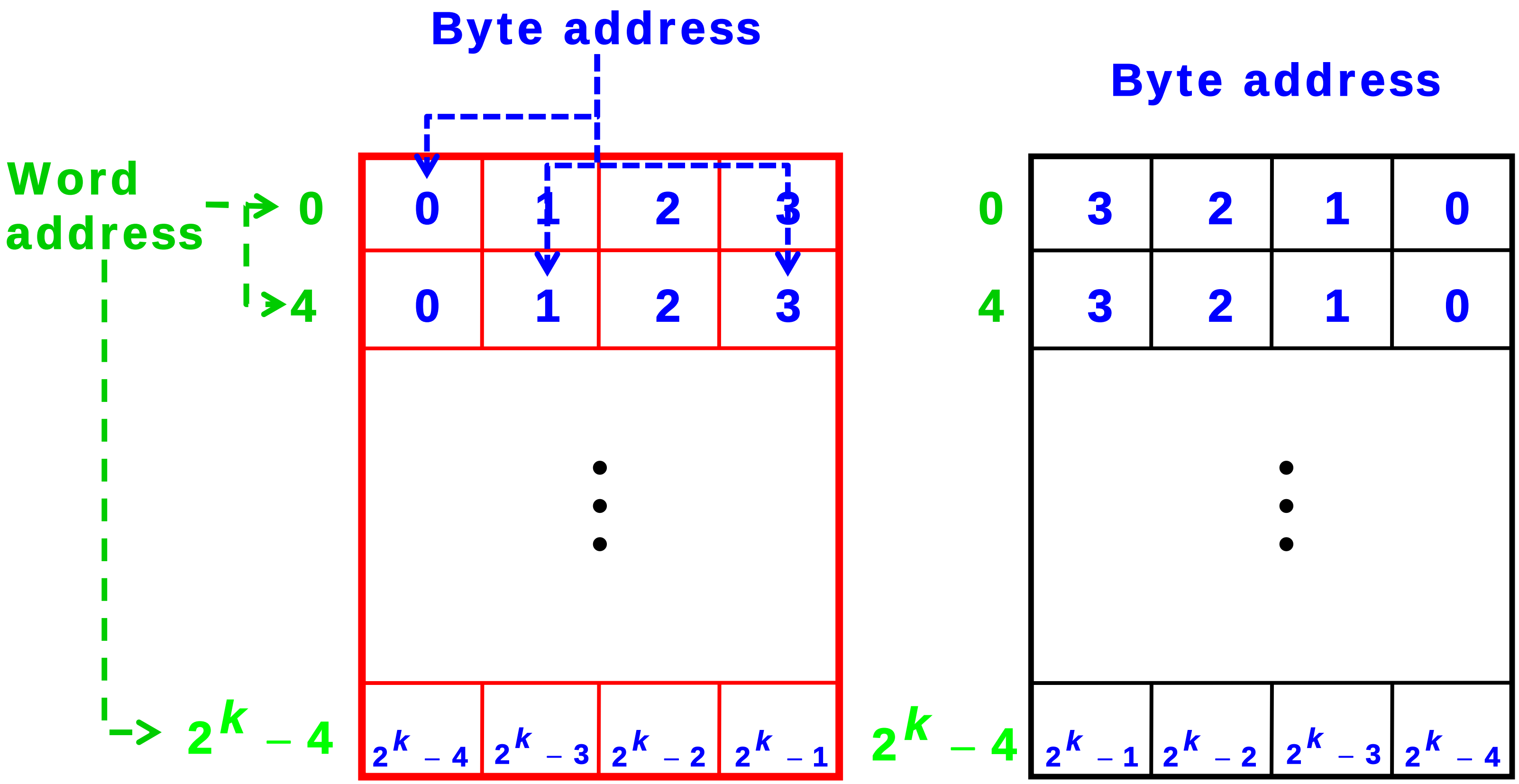
WORD : numbers and addresses - (size varies with processor)

- a byte or word can be viewed as :
  - (a) an unsigned number
  - (b) a signed magnitude number
  - (c) a 1's complement number
  - (d) a 2's complement number
  - (e) a character from some designated set (byte only)
  - (f) anything else you get agreement on

Assignment in Storage

Big-endian: lower byte addresses ➔ more significant bytes

Little-endian: lower byte addresses ➔ less significant bytes

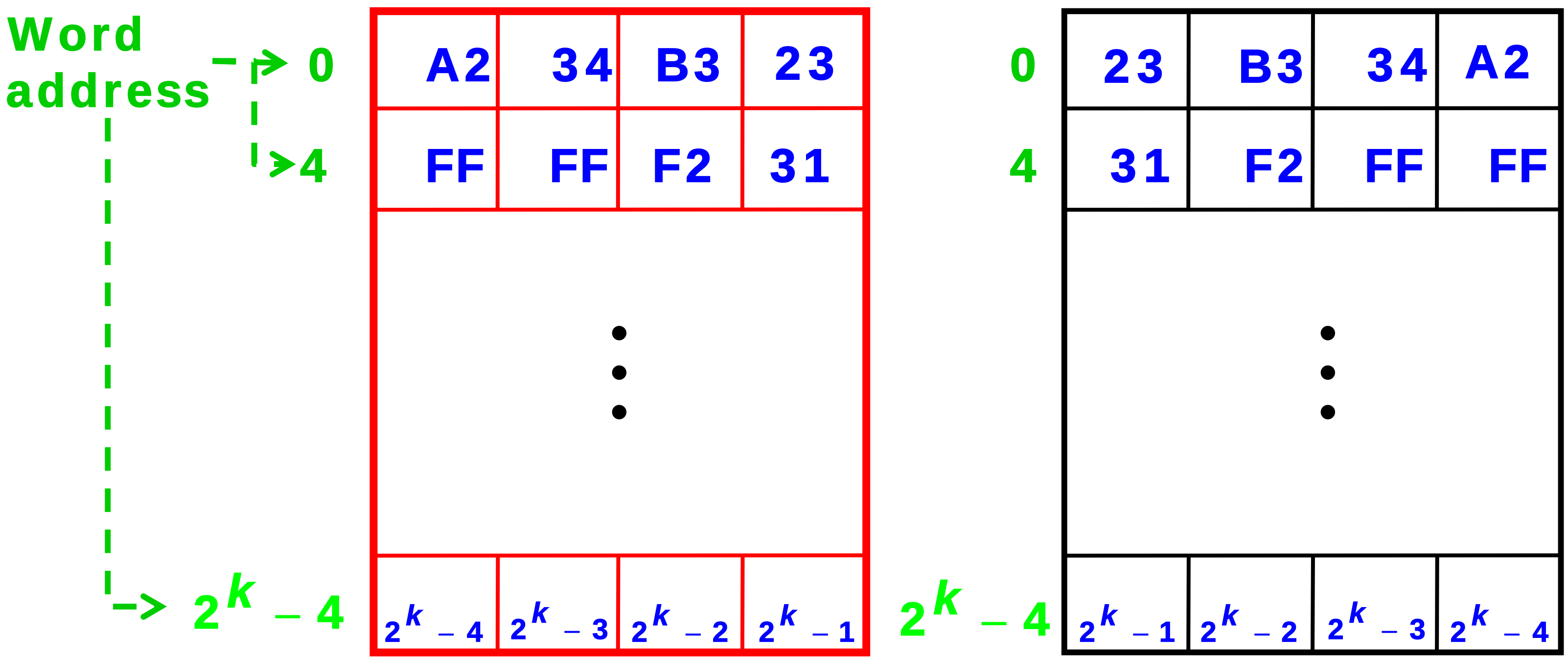


(a) Big-endian assignment (b) Little-endian assignment

Big-Endian / Little-Endian example

2 bytes to be stored: A2 34 B3 23 FF FF F2 31 $_{16}$

Reverse within each byte



(a) Big-endian assignment (b) Little-endian assignment

MC68xxx and ARM

byte addressable and little-endian

76543210	76543210	76543210	76543210
BYTE 3	BYTE 2	BYTE 1	BYTE 0
HALFWORD 1 (16 bits)		HALFWORD 0 (16 bits)	
WORD (32 bits)			

high order
(most significant)

low order
(least significant)

IBM 370

Byte addressable and big-endian

2 bytes = 1 halfword

4 bytes = 1 fullword

8 bytes = 1 doubleword

<table>
<tr><td>B0</td><td>B1</td><td>B2</td><td>B3</td><td>B4</td><td>B5</td><td>B6</td><td>B7</td></tr>
<tr><td colspan="2">half word 0</td><td colspan="2">half word 1</td><td colspan="2">half word 2</td><td colspan="2">half word 3</td></tr>
<tr><td colspan="4">full word 0</td><td colspan="4">full word 1</td></tr>
<tr><td colspan="8">double word 0</td></tr>
</table>

Addition, Subtraction, Overflow

- **Study from the textbook on your own**
- **They will be used later on**
- **They are not tested on Quiz #1**
- **They are tested in Midterm #1**

Competence quiz (Lab 1): what should you be able to do?

Given 8 binary digits, state their decimal equivalence in the cases of:

- **2’s complement**
- **1’s complement**
- **unsigned**
- **signed magnitude**

Given a decimal integer convert it to binary and to hex in the cases of:

- **2’s complement**
- **1’s complement**
- **unsigned**
- **signed magnitude**

READ textbook!

Example Questions (from an old test)

(1) Given the 4-bit hexadecimal numbers below, state the *decimal* equivalent according to the assumption of the representation listed in each heading:

Hexadecimal	E_{16}	5_{16}
Unsigned Integer		
Signed Integer in 2's complement		
Signed Integer in Signed Magnitude		

(2) State the range of decimal values that can be represented in 12 bits, assuming an unsigned integers representation

(3) State the range of decimal values that can be represented in 32 bits, assuming a 2's complement representation

(4) Convert the unsigned binary numbers to decimal and to hexadecimal:

00101101

11111111

(5) Convert the unsigned hexadecimal values to binary and to decimal:

2A

6E

(6) Convert the signed 2’s complement binary numbers to decimal:

00110110

11111111

(7) Convert the signed decimal values to 2’s complement 8-bit binary:

+21

-23

(8) Perform the following operations using 2's complement numbers of 5 bits each. As shown all operations are to be done as additions.

-7 + 8 (in decimal) ____________ +

____________ =

____

10 + 5 (in decimal) ____________ +

____________ =

____

Table available in QUIZ #1 for you

DEC	BIN 8	HEX	DEC	BIN 8	HEX
0	0000 0000	0	8	0000 1000	8
1	0000 0001	1	9	0000 1001	9
2	0000 0010	2	10	0000 1010	0A
3	0000 0011	3	11	0000 1011	0B
4	000 00100	4	12	0000 1100	0C
5	0000 0101	5	13	0000 1101	0D
6	0000 0110	6	14	0000 1110	0E
7	0000 0111	7	15	0000 1111	0F