Computer Systems Architecture

Data And Program Representation

Digital Logic

- Built on two-valued logic system
- Can be interpreted as
  - – *Five volts* and *zero volts*
  - – *High* and *low*
  - – *True* and *false*

Data Representation

- Builds on digital logic
- Applies familiar abstractions
- Interprets sets of Boolean values as
  - – Numbers
  - – Characters
  - – Addresses

Binary Digit (Bit)

- Direct representation of digital logic values
- Assigned mathematical interpretation
  - *0* and *1*
- Multiple bits used to represent complex data item

Byte

- Set of multiple bits
- Size depends on computer
- Examples of byte sizes
  - – CDC: 6-bit byte
  - – BBN: 10-bit byte
  - – IBM: 8-bit byte
- On many computers, smallest addressable unit of storage
- Note: following most modern computers, we will assume an 8-bit byte

Byte Size And Values

- Number of bits per byte determines range of values that can be stored
- Byte of *k* bits can store 2^k values
- Examples
  - – Six-bit byte can store 64 possible values
  - – Eight-bit byte can store 256 possible values

Binary Representation

0 0 0	0 0 1	0 1 0	0 1 1
1 0 0	1 0 1	1 1 0	1 1 1

- All possible combinations of three bits

Meaning Of Bits

- Bits themselves have no intrinsic meaning
- Byte merely stores string of 0's and 1's
- All interpretation determined by use

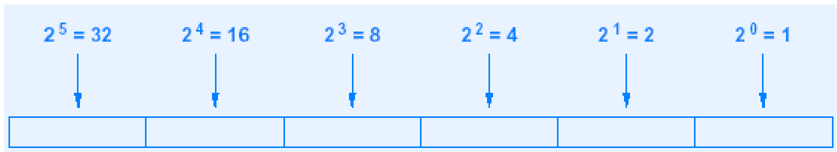
Example Of Interpretation

- Assume three bits used for status of peripheral devices
  - – First bit has the value 1 if a disk is connected
  - – Second bit has the value 1 if a printer is connected
  - – Third bit has the value 1 if a keyboard is connected

Arithmetic Values

- Combination of bits interpreted as an integer
- Positional representation uses base 2
- Note: interpretation must specify order of bits

Illustration Of Positional Interpretation



Example:

0 1 0 1 0 1

is interpreted as:

$$0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 21$$

The Range Of Values

A set of k bits can be interpreted to represent a binary integer.

When conventional positional notation is used, the values that can be represented with k bits range from

0 through $2^k - 1$.

Hexadecimal Notation

Convenient way to represent binary data

Uses base 16

Each hex digit encodes four bits

Hexadecimal Digits

Hex Digit	Binary Value	Decimal Equivalent
0	0 0 0 0	0
1	0 0 0 1	1
2	0 0 1 0	2
3	0 0 1 1	3
4	0 1 0 0	4
5	0 1 0 1	5
6	0 1 1 0	6
7	0 1 1 1	7

Hex Digit	Binary Value	Decimal Equivalent
8	1 0 0 0	8
9	1 0 0 1	9
A	1 0 1 0	10
B	1 0 1 1	11
C	1 1 0 0	12
D	1 1 0 1	13
E	1 1 1 0	14
F	1 1 1 1	15

Hexadecimal Constants

Supported in some programming languages

Typical syntax: constant begins with *0x*

Example:

0xDEC90949

Example Character Encodings

- EBCDIC
- ASCII
- Unicode

EBCDIC

- Extended Binary Coded Decimal Interchange Code
- Defined by IBM
- Popular in 1960s
- Still used on IBM mainframe computers
- Example encoding: lower case letter *a* assigned binary value

10000001

ASCII

- American Standard Code for Information Interchange
- Vendor independent: defined by American National Standards Institute (ANSI)
- Adopted by PC manufacturers
- Specifies 128 characters
- Example encoding: lower case letter *a* assigned binary value

01100001

Full ASCII Character Set

00	nul	01	soh	02	stx	03	etx	04	eot	05	enq	06	ack	07	bel
08	bs	09	ht	0A	lf	0B	vt	0C	np	0D	cr	0E	so	0F	si
10	dle	11	dc1	12	dc2	13	dc3	14	dc4	15	nak	16	syn	17	etb
18	can	19	em	1A	sub	1B	esc	1C	fs	1D	gs	1e	rs	1F	us
20	sp	21	!	22	"	23	#	24	$	25	%	26	&	27	’
28	(	29	)	2A	*	2B	+	2C	,	2D	–	2E	.	2F	/
30	0	31	1	32	2	33	3	34	4	35	5	36	6	37	7
38	8	39	9	3A	:	3B	;	3C	<	3D	=	3E	>	3F	?
40	@	41	A	42	B	43	C	44	D	45	E	46	F	47	G
48	H	49	I	4A	J	4B	K	4C	L	4D	M	4E	N	4F	O
50	P	51	Q	52	R	53	S	54	T	55	U	56	V	57	W
58	X	59	Y	5A	Z	5B	[	5C	\	5D	]	5E	^	5F	_
60	‘	61	a	62	b	63	c	64	d	65	e	66	f	67	g
68	h	69	i	6A	j	6B	k	6C	l	6D	m	6E	n	6F	o
70	p	71	q	72	r	73	s	74	t	75	u	76	v	77	w
78	x	79	y	7A	z	7B	{	7C	|	7D	}	7E	~	7F	del

Unicode

Each character is 16 bits long

Can represent larger set of characters

Motivation: accommodate languages such as Chinese

本課程是一門關於電腦體系結構的發展以及電腦系統軟硬體元件設計中的影響因素的學科。其主題包括：指令集設計、處理器微架構及流水線操作、高速緩衝記憶體(cache)及虛擬記憶體的組織結構、保護及共用、輸入/輸出(I/O)及中斷;順序及亂序超標量體系結構、超長指令字(VLIW)電腦、向量超級電腦、多線程體系結構、對稱多處理器;以及平行電腦。

6.823 is a study of the evolution of computer architecture and the factors influencing the design of hardware and software elements of computer systems. Topics may include: instruction set design; processor micro-architecture and pipelining; cache and virtual memory organizations; protection and sharing; I/O and interrupts; in-order and out-of-order superscalar architectures; VLIW machines; vector supercomputers; multithreaded architectures; symmetric multiprocessors; and parallel computers.

Integer Representation In Binary

Each binary integer represented in k bits

Computers have used k = 8, 16, 32, 60, and 64

Many computers support multiple integer sizes (e.g., 16 and 32 bit integers)

2^k possible bit combinations exist for k bits

Positional interpretation produces *unsigned integers*

Unsigned Integers

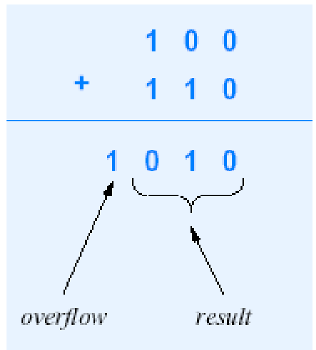
- Straightforward positional interpretation
- Each successive bit represents next power of 2
- No provision for negative values

- Arithmetic operations can produce *overflow* or *underflow*

(result cannot be represented in *k* bits)

- Handled with *wraparound* and *carry bit*

Illustration Of Overflow



- Values wrap around address space
- Hardware records overflow in separate carry indicator

Example Signed Integer Representations

- Sign magnitude
- One’s complement
- Two’s complement
- Note: each has interesting quirks

Sign Magnitude Representation

- Familiar to humans
- First bit represents sign
- Successive bits represent absolute value of integer
- Interesting quirk: can create negative zero

Sign-magnitude representation

Sign-magnitude representation: Most significant bit (*sign bit*) used to indicate the sign and the rest represent the magnitude. if

sign bit = 0 Positive number
sign bit = 1 Negative number

$$A = \begin{cases} A = \sum_{i=0}^{n-2} 2^i b_i \quad \text{if } a_n = 0 \\ A = -\sum_{i=0}^{n-2} 2^i b_i \quad \text{if } a_n = 1 \end{cases}$$

+18 = 00010010

-18 = 10010010

Sign-magnitude representation

Problems with sign-magnitude representation:

- Addition and subtraction:

  Require examination of both sign and magnitude

- Representing zero: +0 and -0

  +0 = 00000000

  -0 = 10000000

Two's complement Representation

- Positive number uses positional representation
- Negative number formed by subtracting 1 from positive value and inverting all bits of result
- Example: 4-bit representation

  0 0 1 0 represents 2
  1 1 1 0 represents –2

- High-order bit is set if number is negative
- Interesting quirk: one more negative values than positive values

Example Of Values In Unsigned And Two's Complement Representations

Binary Value	Unsigned Equivalent	Two's Complement Equivalent
11 1 1	15	-1
1 1 1 0	14	-2
1 1 0 1	13	-3
1 1 0 0	12	-4
1 0 1 1	11	-5
1 0 1 0	10	-6
1 0 0 1	9	-7
1 0 0 0	8	-8
0 1 1 1	7	7
0 1 1 0	6	6
0 1 0 1	5	5
0 1 0 0	4	4
0 0 1 1	3	3
0 0 1 0	2	2
0 0 0 1	1	1
0 0 0 0	0	0

Implementation Of Unsigned And Two's Complement

A computer can use a single piece of hardware to provide unsigned or two's complement integer arithmetic; software running on the computer can choose an interpretation for each integer.

Example (k = 4)

– Adding 1 to binary 1 0 0 1 produces 1 0 1 0
– Unsigned interpretation goes from 9 to 10
– Two's complement interpretation goes from -7 to -6

Sign Extension

- Needed when computer has multiple sizes of integers
- Works for unsigned and two's complement representations
- Extends high-order bit (known as *sign bit*)

- Assume a computer
  - – Supports 16-bit and 32-bit integers
  - – Uses two's complement representation
- When 16-bit integer assigned to 32-bit integer, correct numeric value requires upper sixteen bits to be filled with zeroes for positive number or ones for negative number

- In essence, sign bit from short integer must be extended

Example Of Sign Extension During Assignment

- The 8-bit version of integer -3 is:

1 1 1 1 1 1 0 1

- The 16-bit version of integer -3 is:

1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1

- During assignment to a larger integer, hardware copies all bits of smaller integer and then replicates the high-order (sign) bit in remaining bits

Twos complement representation

Conversion between different bit lengths

+18 = 00010010 (sign magnitude, 8-bit)
+18 = 0000000000010010 (sign magnitude, 16-bit)
-18 = 10010010 (sign magnitude, 8-bit)
-18 = 1000000000010010 (sign magnitude, 16-bit)

+18 = 00010010 (twos complement, 8-bit)
+18 = 0000000000010010 (twos complement, 16-bit)
-18 = 11101110 (twos complement, 8-bit)
-18 = 1111111111101110 (twos complement, 16-bit)

Fixed point Representation

Example Of Sign Extension During Shift

- Right shift of a negative value should produce a negative value
- Example
  - – Shifting -4 one bit should produce -2 (divide by 2)
  - – Using sixteen-bit representation, -4 is:

    1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
- After right shift of one bit, value is -2:

    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
- Solution: replicate high-order bit during right shift

Summary Of Sign Extension

Sign extension: in two's complement arithmetic, when an integer Q composed of K bits is copied to an integer of more than K bits, the additional high-order bits are made equal to the top bit of Q. Extending the sign bit means the numeric value remains the same.

A Consequence For Programmers

Because two's complement hardware performs sign extension, copying an unsigned integer to a larger unsigned integer changes the value; to prevent such errors from occurring, a programmer or a compiler must add code to mask off the extended sign bits.

Numbering Bits And Bytes

- Need to choose order for
  - – Storage in physical memory system
  - – Transmission over serial medium (e.g., a data network)
- Bit order
  - – Handled by hardware
  - – Usually hidden from programmer
- Byte order
- – Affects multi-byte data items such as integers
- – Visible and important to programmer

Possible Byte Order

- Least significant byte of integer in lowest memory location
  - – Known as *little endian*
- Most significant byte of integer in lowest memory location
  - – Known as *big endian*
- Other orderings
  - – Digital Equipment Corporation once used an ordering with sixteen-bit words in big endian order and bytes within the words in little endian order.
- Note: only big and little endian storage are popular

Illustration Of Big And Little Endian Byte Order

0	1	2	3
0x00	0x00	0x00	0x01

Big Endian

3	2	1	0
0x00	0x00	0x00	0x01

Little Endian

- Note: difference is especially important when transferring data between computers for which the byte ordering differs

Real Numbers

- **Numbers with fractions**
- **Could be done in pure binary**
  $1001.1010 = 2^4 + 2^0 + 2^{-1} + 2^{-3} = 9.625$
- **Where is the binary point?**
- **Fixed?**
  Very limited - cannot represent very large or very small numbers
- **Moving?**
  How do you show where it is?

Floating Point Representation

Principles

Scientific notation:

$$543{,}000{,}000{,}000{,}000 = 5.43 \times 10^{14}$$

Slide the decimal point to a convenient location
Keep track of the decimal point use the exponent of 10

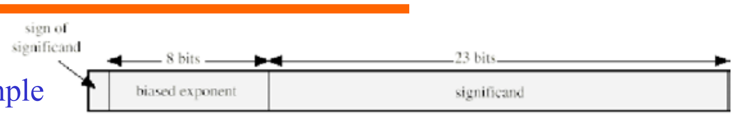
Do the same with binary number in the form of

$$\pm S \times B^{\pm E}$$

J Sign: + or -
J Significant: S
J Exponent: E

Floating Point Representation

Example



J 32-bit floating point format.
J Leftmost bit = sign bit (0 positive or 1 negative).
J Exponent in the next 8 bits. Use a biased representation.

A fixed value, called bias, is subtracted from the field to get the true exponent value. Typically, bias = $2^{k-1} - 1$, where k is the number of bits in the exponent field. Also known as excess-N format, where N = bias = $2^{k-1} - 1$. (The bias could take other values)

In this case: 8-bit exponent field, 0 - 255. Bias = 127. Exponent range -127 to +128

J Final portion of word (23 bits in this example) is the significant (sometimes called mantissa).

Floating Point Representation

Many ways to represent a floating point number, e.g.,

$$0.110 \times 2^5 \qquad 110 \times 2^2 \qquad 0.0110 \times 2^6$$

Normalization: Adjust the exponent such that the leading bit (MSB) of mantissa is always 1. In this example, a normalized nonzero number is in the form

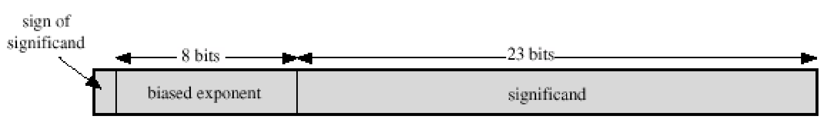
$$\pm 1.bbb...b \times 2^{\pm E}$$

J Left most bit always 1 - no need to store
J 23-bit field used to store 24-bit mantissa with a value between 1 to 2

The University of Nottingham School of Computer Science G51CSA

43

Floating Point Representation



(a) Format

$$\begin{aligned}
1.1010001 \times 2^{10100} &= 0\ 10010011\ 10100010000000000000000 = 1.638125 \times 2^{20} \\
-1.1010001 \times 2^{10100} &= 1\ 10010011\ 10100010000000000000000 = -1.638125 \times 2^{20} \\
1.1010001 \times 2^{-10100} &= 0\ 01101011\ 10100010000000000000000 = 1.638125 \times 2^{-20} \\
-1.1010001 \times 2^{-10100} &= 1\ 01101011\ 10100010000000000000000 = -1.638125 \times 2^{-20}
\end{aligned}$$

J Sign stored in the first bit
J Left most bit of the TRUE mantissa always 1 - no need to store
J The value of 127 is added to the TRUE exponent to be stored
J The base is 2

The University of Nottingham School of Computer Science G51CSA

44

Floating Point Representation

IEEE 754 Standard

Single Format and Double Format

- Single Precision format:
  - 32 bits, sign = 1 bit, Exponent = 8bits, Mantissa = 32 bits
  - Numbers are normalised to form: $\pm 1.bbb \ldots b \times 2^{\pm E}$; where b = 0 or 1
  - Exponent formatted using excess-127 notation with implied base of 2
  - Theoretical exponent range 2^{-127} to 2^{128}
  - Actuality, exponent values of 0 and 255 used for special values
  - Exponent range restricted to -126 to 127
  - 0.0 defined by a mantissa of 0 and the special exponent value of 0
  - Allows + - infinity defined by a mantissa value of 0 and exponent value 255

Range Of Values In IEEE Floating Point

- Single precision range is:

  2^{126} to 2^{127}
- Decimal equivalent is approximately:

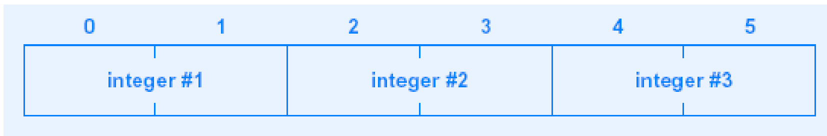
  10^{38} to 10^{38}
- Double precision range is:

  10^{308} to 10^{308}

Data Aggregates

1 Typically arranged in contiguous memory

1 Example: three integers



Integer Arithmetic

Negation

Sign-magnitude: Invert the sign bit

Twos complement:

J Invert each bit (including the sign bit).
J Treat the result as unsigned binary integer, and add 1

E.g.

Integer Arithmetic

Addition and Subtraction

```
 1001                    1100
+0101                   +0100
 1110 = -2              10000 = 0
(a) (-7) + (+5)         (b) (-4) + (+4)

 0011                    1100
+0100                   +1111
 0111 = 7               11011 = -5
(c) (+3) + (+4)         (d) (-4) + (-1)

 0101                    1001
+0100                   +1010
 1001 = Overflow        10011 = Overflow
(e) (+5) + (+4)         (f) (-7) + (-6)
```

Overflow

Result larger than can be held in the word size being used resulting in overflow.

If two numbers have the same sign are added, then overflow occurs iif (if and only if) the result has the opposite sign.

Carry bit ignored

School of Computer Science G51CSA

49

Integer Arithmetic

```
   0010                          0101
  +1001                         +1110
   1011 = -5                    10011 = 3

(a) M = 2 = 0010             (b) M = 5 = 0101
    S = 7 = 0111                 S = 2 = 0010
   -S =     1001                -S =     1110

   1011                          0101
  +1110                         +0010
  11001 = -7                     0111 = 7

(c) M =-5 = 1011             (d) M = 5 = 0101
    S = 2 = 0010                 S =-2 = 1110
   -S =     1110                -S =     0010

   0111                          1010
  +0111                         +1100
   1110 = Overflow              10110 = Overflow

(e) M = 7 = 0111             (f) M =-6 = 1010
    S =-7 = 1001                 S = 4 = 0100
   -S =     0111                -S =     1100
```

(M - S)

Subtraction

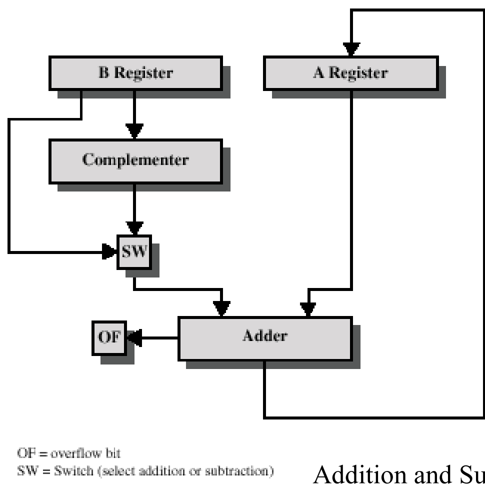
To subtract one number (subtrahend) from another number minuend), take the twos complement (negation) of the subtrahend and add it to the minuend.

Overflow rule applies here also

School of Computer Science G51CSA

50

Integer Arithmetic



Addition and Subtraction Hardware Block Diagram

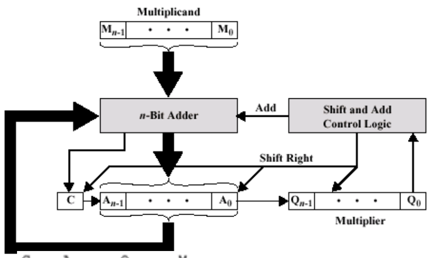
Integer Arithmetic

Multiplication: Unsigned binary integers

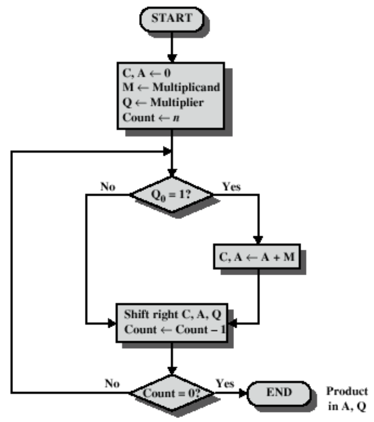
```
    1011      Multiplicand (11)
   x1101      Multiplier (13)
   -----
    1011  \
   0000    |
  1011     |  Partial products
 1011     /
--------
10001111      Product (143)
```

Integer Arithmetic

Multiplication



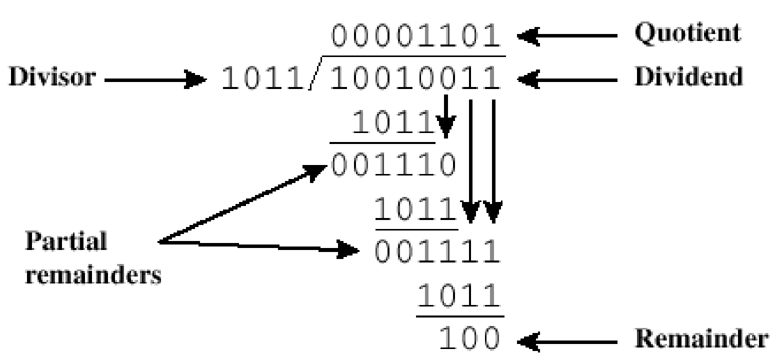
C	A	Q	M		
0	0000	1101	1011	Initial Values	
0	1011	1101	1011	Add	First Cycle
0	0101	1110	1011	Shift	
0	0010	1111	1011	Shift	Second Cycle
0	1101	1111	1011	Add	Third Cycle
0	0110	1111	1011	Shift	
1	0001	1111	1011	Add	Fourth Cycle
0	1000	1111	1011	Shift	



Flowchart for unsigned binary multiplication

Integer Arithmetic

Division: Unsigned binary integer



Floating Point Arithmetic

Addition and Subtraction

- Check for zero
- Align the significants
- Add or subtract the significants
- Normalise the result

E.g. $0.5566 \times 10^{3} + 0.7778 \times 10^{3}$

$0.5323 \times 10^{2} + 0.7268 \times 10^{-1}$

Summary

- Basic output from digital logic is a bit
- Bits grouped into sets to represent
  - – Integers
  - – Characters
  - – Floating point values
- Integers can be represented as
  - – Sign magnitude
  - – Two's complement

Summary

- One piece of hardware can be for both
  - – Two's complement arithmetic
  - – Unsigned arithmetic
- Bytes of integer can be numbered in
  - – Big-endian order
  - – Little-endian order
- Organizations such as ANSI and IEEE define standards for data representation

Summary

- Integer arithmetic
  - two's complement - subtraction and addition rules, overflow rule
- Floating point arithmetic