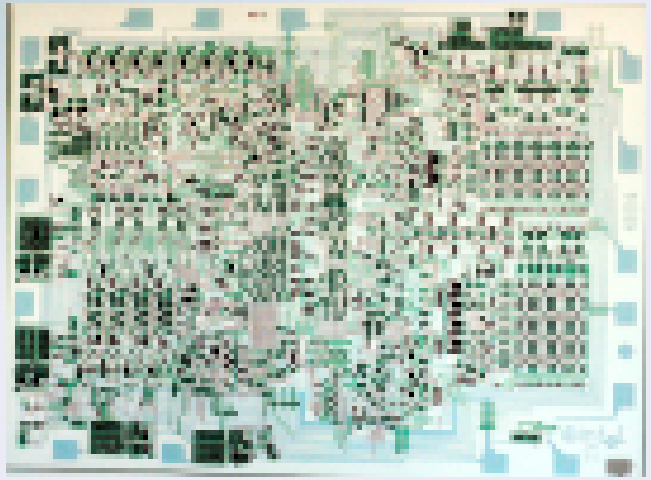

Lecture 2a: Bits, Data Types and Operations

CSCI11: **Computer Architecture and Organization**

Chapter 2 in Patt and Patel: Introduction to Computing Systems...

How do we represent data in a computer?

At the lowest level, a computer has electronic “plumbing”

Operates by controlling the flow of electrons

Easy to recognize two conditions:

1. Presence of a voltage – we’ll call this state “1”
2. Absence of a voltage – we’ll call this state “0”

On/Off light switch versus dimmer switch

Computer is a binary digital system

Basic unit of information: the binary digit, or bit

- 2 states require 1 bit, state 0 or state 1
- 3 or more state values require multiple bits
- A collection of two bits has four possible states: 00, 01, 10, 11
- A collection of three bits has eight possible states: 000, 001, 010, 011, 100, 101, 110, 111
- A collection of n bits has 2^n possible states

Binary Digital system: Finite number of 2 state symbols

What kinds of data do we need to represent?

Numbers – signed, unsigned, integers, real, floating point, complex, rational, irrational

Text – characters, strings

Images – pixels, colors, shapes

Sound - music, noise

Logical – true, false

Instructions - programming instructions

Data type:

Representation and operations within the computer

We'll start with *numbers…*

Unsigned Integers

Non-positional notation

- Could represent a number (“5”) with a string of ones (“11111”) **Why not?**

Weighted positional notation

- Decimal numbers: “329”
- “3” is worth 300, because of its position, while “9” is only worth 9
- most significant->least significant, base-10 (decimal)
  3x100 + 2x10 + 9x1 = 329

Base 3 Number System

Decimal	Base 3
0	0
1	1
2	2
3	10
4	11
5	12
6	20
7	21
8	22
9	100

Binary Number System (2^3 values)

N = 3	Decimal
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Unsigned Integers

An n-bit unsigned integer represents 2^n values

From 0 to 2^n-1

Bits	Expression	Total Value	Range
3	2^3	8	0 - 7
8	2^8	256	0 - 255
16	2^{16}	65,536	0 - 65,535
32	2^{32}	4,294,967,296	0 - 4,294,967,295

Powers of 2

- Notice the sum of all values are 1 less than next power
- With 8 bits, you can add to 255, 1 less than 2**8
- With 16 bits, you can add to 65,535

8 bits

7	6	5	4	3	2	1	0
128	64	32	16	8	4	2	1

16 bits

15	14	13	12	11	10	9	8
32,768	16,384	8192	4096	2048	1024	512	256

Table of Unsigned Integers

N = 3	Value	Expression
000	0	$0*2^2 + 0*2^1 + 0*2^0$
001	1	$0*2^2 + 0*2^1 + 1*2^0$
010	2	$0*2^2 + 1*2^1 + 0*2^0$
011	3	$0*2^2 + 1*2^1 + 1*2^0$
100	4	$1*2^2 + 0*2^1 + 0*2^0$
101	5	$1*2^2 + 1*2^1 + 0*2^0$
110	6	$1*2^2 + 1*2^1 + 0*2^0$
111	7	$1*2^2 + 1*2^1 + 1*2^0$

Unsigned Binary Arithmetic

Base-2 addition – just like base-10

Add from right to left, propagating carry

Signed Integers

- With n bits, we have 2^n distinct values
- Assign “half” to positive integers (1 through ~2^n-1) and “half” to negative (~-2^n-1 through -1)
- That leaves two values: one for 0, and one extra

Positive integers

- Just like unsigned with zero in most significant bit

Negative integers

- Sign-magnitude: set high-order bit to show negative, other bits are the same as unsigned

Negative Numbers

Idea

Find representation to make arithmetic simple and consistent

Specifics

For each positive number (X), assign value to its negative (-X), such that X + (-X) = 0 with “normal” addition, **ignoring carry out**

Positive and Negative Numbers in Binary (Signed Magnitude)

Positive Numbers	Binary Representation	Negative Numbers	Binary Representation
0	0000	-0	1000
1	0001	-1	1001
2	0010	-2	1010
3	0011	-3	1011
4	0100	-4	1100
5	0101	-5	1101
6	0110	-6	1110
7	0111	-7	1111

0010 + 1010 = 1100 -> 2 + -2 = 4? ❌

1's Complement - flip every bit to represent negative

Positive	Binary	Negative	Binary
0	0000	-0	1111
1	0001	-1	1110
2	0010	-2	1101
3	0011	-3	1100
4	0100	-4	1011
5	0101	-5	1010
6	0110	-6	1001
7	0111	-7	1000

0010 + 1101 = 1111 -> 2 + -2 = -0 ✅

0100 + 1100 = 0000 -> 4 + -3 = 0 ❌

Two’s Complement

If number is positive or zero

- Normal binary representation, zeroes in upper bit(s)

If number is negative

- Start with positive number
- Flip every bit (i.e., take the one’s complement)
- Then add one

Two’s Complement Shortcut

To take the two’s complement of a number:

1. Copy bits from right to left until (**and including**) the first “1”
2. Flip remaining bits to the left

Two’s Complement Signed Integers

- MS bit is sign bit: it has weight –2^n-1
- Range of an n-bit number: -2^n-1 through 2^n-1 – 1

  Note: most negative number (-2^n-1) has no positive counterpart

Positive/Negative Numbers in 2's Complement

Positive	Binary	Negative	Binary
0	0000	-0	0000
1	0001	-1	1111
2	0010	-2	1110
3	0011	-3	1101
4	0100	-4	1100
5	0101	-5	1011
6	0110	-6	1010
7	0111	-7	1001

0001 + 1110 = 0 ✅

0100 + 1101 = 0001 ✅

Converting Binary (2’s C) to Decimal

1. If leading bit is one, take two’s complement to get a positive number
2. Add powers of 2 that have “1” in the corresponding bit positions
3. If original number was negative, add a minus sign

Converting Decimal to Two's Complement Binary

First Method: Division

1. Change to positive decimal number
2. Divide by two – remainder is least significant bit
3. Keep dividing by two until answer is zero, recording remainders from right to left
4. Append a zero as the MS bit; if original number negative, take two’s complement

Converting Decimal to Two's Complement Binary

Second Method: Subtract Powers of Two

1. Change to positive decimal number
2. Subtract largest power of two less than or equal to number
3. Put a one in the corresponding bit position
4. Keep subtracting until result is zero
5. Append a zero as MS bit; if original was negative, take two’s complement

Operations: Logic and Arithmetic

A data type will include both representation and operations

Logic operations are common

- *AND*
- *OR*
- *NOT*

Operations for signed integers

- Addition
- Subtraction
- Sign Extension

Logic Operations

- Operations on logical TRUE or FALSE
- Two states: TRUE=1, FALSE=0
- View n-bit number as a collection of n logical values
- Operation applied to each bit independently

AND Logic

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

OR Logic

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

Examples of Logical Operations

AND

- Useful for clearing bits
- *AND* with zero = 0, *AND* with one = no change

OR

- Useful for setting bits
- *OR* with zero = no change *OR* with one = 1

NOT

- Unary operation, flips every bit

Bit Vector or Bit *Mask*

- Track specific aspects of a system
- In an 8-bit register, track the state of 8 conditions, *on/off*
- **Extremely common**, both internal to microprocessor (or *microcontroller*) and external to other devices
- Used extensively with logic operations, typically called a **mask**

2's Complement Addition

- 2’s comp. addition is just binary addition
- Assume all integers have the same number of bits
- **Ignore carry out**
- For now, assume that sum fits in n-bit 2’s comp. representation

2's Complement Subtraction

- **Negate 2nd operand and add**
- Assume all integers have the same number of bits
- Ignore carry out
- For now, assume that difference fits in n-bit 2’s comp. representation

Sign Extension

To add:

- Must represent numbers with same number of bits
- What if we just pad with zeroes on the left?
- No, let’s replicate the MSB (the sign bit)

Overflow

What if operands are too big?

- Sum cannot be represented as n-bit 2’s comp number
- We have overflow if:
  - Signs of both operands are the same, **and**
  - Sign of sum is different
- Another test (easy for hardware)
  - Carry into most significant bit does not equal carry out

Hexadecimal or Base 16 Numbers

Dec	Bin	Hex	Dec	Bin	Hex
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

Hexadecimal Notation (also called "hex")

- It is often convenient to write binary (base-2) numbers as hexadecimal (base-16) numbers instead
- Fewer digits: four bits per hex digit
- Less error prone: easy to misread long string of 1’s and 0’s
- Written as 0x*hexnumber* as in 0xf3, 0x19, 0x00

Binary	Grouped	Hex
10101111	1010 1111	xAF
1010111110101111	1010 1111 1101 1111	xAFDF
10101111001011111000 1111	1010 1111 0010 1111 1000 1111	xAF2F8E

Converting from Binary to Hexadecimal

1. Every group of four bits is a hex digit
2. Start grouping from right-hand side

Examples:

```
b0110 1000 => x68
b1010 1001 => xA9
```

Fractions: Fixed-Point

- How can we represent fractions?
- Use a “binary point” to separate positive from negative powers of two (just like “decimal point”)
- 2’s comp addition and subtraction still work!

Very Large and Very Small: Floating-Point

Problem

- Large values: 6.023 x 10^23 -- requires 79 bits
- Small values: 6.626 x 10^-34 -- requires >110 bits
- Use equivalent of “scientific notation”: F x 2E
- Need to represent F (fraction), E (exponent), and sign
- IEEE 754 Floating-Point Standard (32-bits):
- 1 bit for sign, 8 bits for exponent, 23 bits for fraction

Floating Point Example

- Single-precision IEEE floating point number - 32 bits
- Sign is 1: number is negative
- Exponent field is 01111110 = 126 (decimal)
- Fraction is 0.100000000000… = 1/10 = 0.5 (decimal)
- Value = -1.5 x 2(126-127) = -1.5 x 2-1 = -0.75

Floating Point Specials

- If exponent bits are 0, “denormalized” numbers
- Gradual underflow (also used for representing zero)
- Other specials
- Two zeros (-0, 0)
- Two Infinities (-infinity, infinity)
- Not a number (negative and positive)!
- When does this occur?
- Lots of corner cases (difficult to implement correctly)
- Example: rounding modes

Text: ASCII Characters

- ASCII: Maps 128 characters to 7-bit code.
- Both printable and non-printable (ESC, DEL, …) characters

ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	*[NULL]*
1	1	1	1	*[START OF HEADING]*
2	2	10	2	*[START OF TEXT]*
3	3	11	3	*[END OF TEXT]*
4	4	100	4	*[END OF TRANSMISSION]*
5	5	101	5	*[ENQUIRY]*
6	6	110	6	*[ACKNOWLEDGE]*
7	7	111	7	*[BELL]*
8	8	1000	10	*[BACKSPACE]*
9	9	1001	11	*[HORIZONTAL TAB]*
10	A	1010	12	*[LINE FEED]*
11	B	1011	13	*[VERTICAL TAB]*
12	C	1100	14	*[FORM FEED]*
13	D	1101	15	*[CARRIAGE RETURN]*
14	E	1110	16	*[SHIFT OUT]*
15	F	1111	17	*[SHIFT IN]*
16	10	10000	20	*[DATA LINK ESCAPE]*
17	11	10001	21	*[DEVICE CONTROL 1]*
18	12	10010	22	*[DEVICE CONTROL 2]*
19	13	10011	23	*[DEVICE CONTROL 3]*
20	14	10100	24	*[DEVICE CONTROL 4]*
21	15	10101	25	*[NEGATIVE ACKNOWLEDGE]*
22	16	10110	26	*[SYNCHRONOUS IDLE]*
23	17	10111	27	*[END OF TRANS. BLOCK]*
24	18	11000	30	*[CANCEL]*
25	19	11001	31	*[END OF MEDIUM]*
26	1A	11010	32	*[SUBSTITUTE]*
27	1B	11011	33	*[ESCAPE]*
28	1C	11100	34	*[FILE SEPARATOR]*
29	1D	11101	35	*[GROUP SEPARATOR]*
30	1E	11110	36	*[RECORD SEPARATOR]*
31	1F	11111	37	*[UNIT SEPARATOR]*
32	20	100000	40	*[SPACE]*
33	21	100001	41	**!**
34	22	100010	42	**"**
35	23	100011	43	**#**
36	24	100100	44	**$**
37	25	100101	45	**%**
38	26	100110	46	**&**
39	27	100111	47	**'**
40	28	101000	50	**(**
41	29	101001	51	**)**
42	2A	101010	52	*****
43	2B	101011	53	**+**
44	2C	101100	54	**,**
45	2D	101101	55	**-**
46	2E	101110	56	**.**
47	2F	101111	57	**/**
48	30	110000	60	**0**
49	31	110001	61	**1**
50	32	110010	62	**2**
51	33	110011	63	**3**
52	34	110100	64	**4**
53	35	110101	65	**5**
54	36	110110	66	**6**
55	37	110111	67	**7**
56	38	111000	70	**8**
57	39	111001	71	**9**
58	3A	111010	72	**:**
59	3B	111011	73	**;**
60	3C	111100	74	**<**
61	3D	111101	75	**=**
62	3E	111110	76	**>**
63	3F	111111	77	**?**
64	40	1000000	100	**@**
65	41	1000001	101	**A**
66	42	1000010	102	**B**
67	43	1000011	103	**C**
68	44	1000100	104	**D**
69	45	1000101	105	**E**
70	46	1000110	106	**F**
71	47	1000111	107	**G**
72	48	1001000	110	**H**
73	49	1001001	111	**I**
74	4A	1001010	112	**J**
75	4B	1001011	113	**K**
76	4C	1001100	114	**L**
77	4D	1001101	115	**M**
78	4E	1001110	116	**N**
79	4F	1001111	117	**O**
80	50	1010000	120	**P**
81	51	1010001	121	**Q**
82	52	1010010	122	**R**
83	53	1010011	123	**S**
84	54	1010100	124	**T**
85	55	1010101	125	**U**
86	56	1010110	126	**V**
87	57	1010111	127	**W**
88	58	1011000	130	**X**
89	59	1011001	131	**Y**
90	5A	1011010	132	**Z**
91	5B	1011011	133	**[**
92	5C	1011100	134	****
93	5D	1011101	135	**]**
94	5E	1011110	136	**^**
95	5F	1011111	137	**_**
96	60	1100000	140	**`**
97	61	1100001	141	**a**
98	62	1100010	142	**b**
99	63	1100011	143	**c**
100	64	1100100	144	**d**
101	65	1100101	145	**e**
102	66	1100110	146	**f**
103	67	1100111	147	**g**
104	68	1101000	150	**h**
105	69	1101001	151	**i**
106	6A	1101010	152	**j**
107	6B	1101011	153	**k**
108	6C	1101100	154	**l**
109	6D	1101101	155	**m**
110	6E	1101110	156	**n**
111	6F	1101111	157	**o**
112	70	1110000	160	**p**
113	71	1110001	161	**q**
114	72	1110010	162	**r**
115	73	1110011	163	**s**
116	74	1110100	164	**t**
117	75	1110101	165	**u**
118	76	1110110	166	**v**
119	77	1110111	167	**w**
120	78	1111000	170	**x**
121	79	1111001	171	**y**
122	7A	1111010	172	**z**
123	7B	1111011	173	**{**
124	7C	1111100	174	**|**
125	7D	1111101	175	**}**
126	7E	1111110	176	**~**
127	7F	1111111	177	*[DEL]*

Interesting Properties of ASCII Code

- What is relationship between a decimal digit ('0', '1', …) and its ASCII code?
- What is the difference between an upper-case letter ('A', 'B', …) and its lower-case equivalent ('a', 'b', …)?
- Given two ASCII characters, how do we tell which comes first in alphabetical order?
- (http://www.unicode.org/)
- No new operations -- integer arithmetic and logic.
- Table attribution - *ZZT32, Public domain, via Wikimedia Commons

Other Data Types

Text strings

- Sequence of characters, terminated with NULL (0)
- Typically, no hardware support

Image

- Array of pixels!
- Monochrome: one bit (0/1 = black/white)!
- Color: RGB components (e.g., 8-24 bits each), transparency
- Hardware support, not in general-purpose processors

Sound

- Sequence of fixed-point numbers