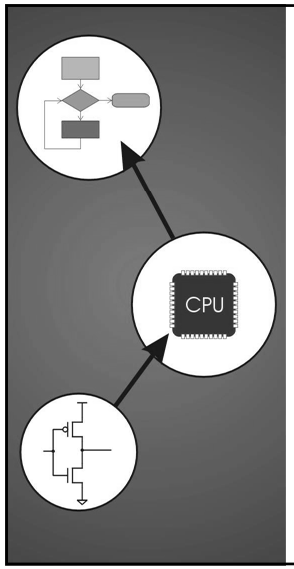


Chapter 1
Welcome Aboard

How do we solve a problem using a computer?

Using a computer to solve a problem involves works carried out at several levels.

- Problem Statement
- Algorithm
- Program
- Machine Architecture
- Micro-architecture
- Logic Circuits
- Devices

Descriptions of Each Level

Problem Statement
- stated using "natural language"
- may be ambiguous, imprecise

Algorithm
- step-by-step procedure, guaranteed to finish

Program
- express the algorithm using a computer language
- the language cannot be directly understood by the computer

Instruction Set Architecture (ISA)
- specifies the set of instructions the computer can perform
- data types, addressing mode

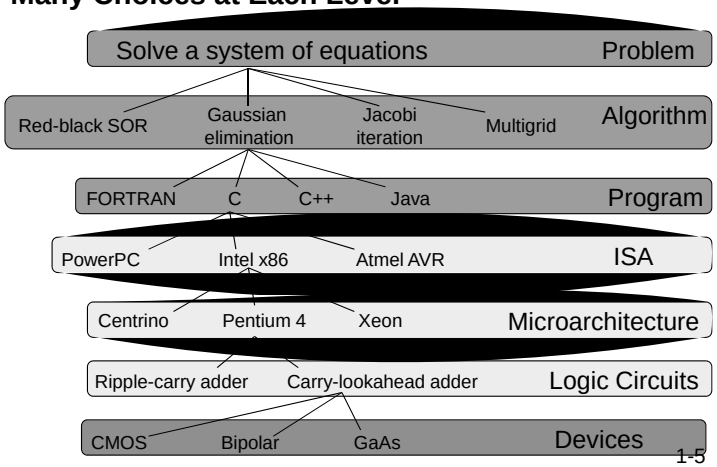
Descriptions of Each Level (cont.)

Microarchitecture
- detailed organization of a processor implementation
- different implementations of a single ISA

Logic Circuits
- combine basic operations to realize microarchitecture
- e.g., addition, comparison

Devices
- properties of materials, manufacturability

Many Choices at Each Level



Outline of My Part of the Course

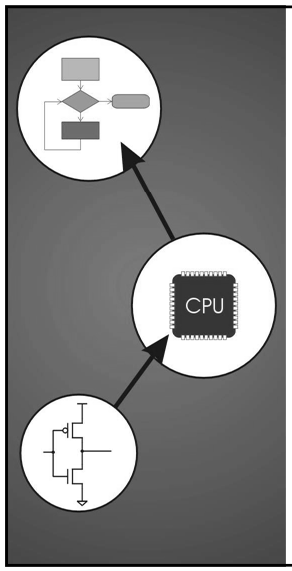
Number Representation
- **How do we represent information in computer**

Processor and Instruction Set
- **What does a processor consist of and what do these components do?**
- **What operations (instructions) will we implement?**

Assembly Language Programming
- **How do we use processor instructions to implement algorithms?**
- **How do we write modular, reusable code? (subroutines)**

I/O and Traps
- **How does processor communicate with outside world?**

Digital Logic
- **How do we build circuits to process information?**



Chapter 2
Bits, Data Types, and Operations

Agenda

1. **Why does computer use binary numbers**
2. **Conversion between unsigned decimal and binary numbers**
3. **Representing signed numbers**
4. **Arithmetic Operations**
5. **Logical Operations**
6. **Shifting**
7. **Hexadecimal & Octal Notation**
8. **ASCII characters**

How do we represent data in a computer?

At the lowest level, a computer is an electronic machine.

- **works by controlling the flow of electrons**

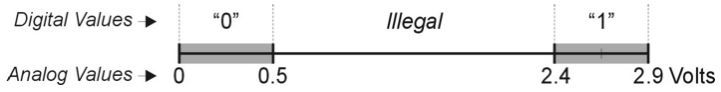
Easy to recognize two conditions:

1. **presence of a voltage – we'll call this state "1"**
2. **absence of a voltage – we'll call this state "0"**

This feature determines that the best number system for a computer is the base-2 (i.e. binary) number

Computer is a binary digital system.

Binary (base two) system has two states: 0 and 1



Basic unit of information is the *binary digit*, or *bit*.

Agenda

1. **Why does computer use binary numbers**
2. **Conversion between unsigned decimal and binary numbers**
3. **Representing signed numbers**
4. **Arithmetic Operations**
5. **Logical Operations**
6. **Shifting**
7. **Hexadecimal & Octal Notation**
8. **ASCII characters**

What kinds of data do we need to represent?

- **Numbers – signed, unsigned, integers, floating point, …**
- **Text – characters, strings, …**
- **Images – pixels, colors, shapes, …**
- **Sound**
- **Logical – true, false**
- **Instructions**
- **…**

We'll start with numbers…

Decimal Numbers

"decimal" means base-10 number.

We have ten digits to use in our representation (the symbols 0 through 9)

What is 3,546?

- **it is *three* thousands plus *five* hundreds plus *four* tens plus *six* ones.**
- **i.e. 3,546 = $3*10^3 + 5*10^2 + 4*10^1 + 6*10^0$**

How about negative numbers?

- **we use two more symbols to distinguish positive and negative: + and –**

13

In general, a base-X number "abc" can be represented as
$a*X^2 + b*X^1 + c*X^0$

Unsigned Binary (base-2) Integers

Y = "abc" = $a.2^2 + b.2^1 + c.2^0$

(where the digits a, b, c can each take on the values of 0 or 1 only)

14

Converting Unsigned Decimal to Binary

Repeated *Division*

1. **Divide by two – the remainder is the least significant bit and the quotient is used in the next round of division.**
2. **Keep dividing by two until quotient is zero, writing remainders from right to left.**

$X = 104_{ten}$

Convert the number to an 8-bit binary number

$X = 01101000_{two}$

104/2	=	52 r0	*bit 0*
52/2	=	26 r0	*bit 1*
26/2	=	13 r0	*bit 2*
13/2	=	6 r1	*bit 3*
6/2	=	3 r0	*bit 4*
3/2	=	1 r1	*bit 5*
1/2	=	0 r1	*bit 6*

14_{10}=

15

Converting Unsigned Binary to Decimal

1. **For any unsigned binary number, say abc, represent the number as $a*2^2 + b*2^1 + c*2^0$**
2. **Treat $a*2^2 + b*2^1 + c*2^0$ as an expression for decimal number and calculate the value of the expression**

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

Convert an 8-bit unsigned binary number:

$X = 01101000_{two}$
$= 0*2^7+1*2^6+1*2^5+0*2^4+1*2^3+0*2^2+0*2^1+0*2^0$
$= 2^6+2^5+2^3$
$= 64+32+8$
$= 104_{ten}$

1011_2 =

16

Unsigned Binary Arithmetic

Base-2 addition – just like base-10!

- **add from right to left, propagating carry**

carry

```
  10010        10010        1111
+  1001      +  1011      +    1
  11011        11101       10000

               10111
             +   111
```

Subtraction, multiplication, division can be done in the same way as base-10 number

What is the range of an N-bit unsigned binary number?

N-bit unsigned binary can be represented as

- $a_{N-1}*2^{N-1} + a_{N-2}*2^{N-2} + \ldots + a_1*2^1 + a_0*2^0$
- **Smallest = 0 (i.e. all $a_i = 0$)**
- **Largest = 1…1 = $2^N - 1$**

N-bit unsigned binary number
Range is:
$0 \le i \le 2^N - 1$

Agenda

1. **Why does computer use binary numbers**
2. **Conversion between unsigned decimal and binary numbers**
3. **Representing signed numbers**
4. **Arithmetic Operations**
5. **Logical Operations**
6. **Shifting**
7. **Hexadecimal & Octal Notation**
8. **ASCII characters**

Signed Magnitude

Leading bit is the <u>sign</u> bit

Y = "abc" = $(-1)^a (b.2^1 + c.2^0)$

Range is:
$-2^{N-1} + 1 \le i \le 2^{N-1} - 1$

-3	111
-2	110
-1	101
-0	100
+0	000
+1	001
+2	010
+3	011

Convert -7_{10} to a signed magnitude binary.
Convert 1001_2 to a decimal number.

<u>Problems:</u>

- How do we do addition/subtraction?
- We have two numbers for zero (+/-)!

One's Complement

Invert all bits of negative number

If msb (most significant bit) is 1 then the number is negative (same as signed magnitude)

Range is:

$-2^{N-1} + 1 \leq i \leq 2^{N-1} - 1$

-3	100
-2	101
-1	110
-0	111
+0	000
+1	001
+2	010
+3	011

Convert -7_{10} to an one's complement binary.
Convert 0001_2 to a decimal number.

Problems:

•Same as for signed magnitude

Two's Complement

Problems with sign-magnitude and 1's complement

- **two representations of zero (+0 and –0)**
- **arithmetic circuits are complex**
  - **The sign bit has to be treated differently**

Two's complement **representation is developed to make circuits easy for arithmetic.**

- **The sign bit can be processed in the same way as other bits**

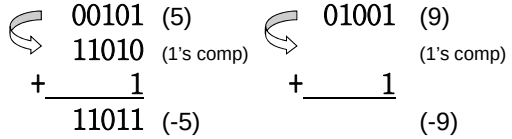
Two's Complement Representation

If number is positive or zero,

- **normal binary representation, zeroes in upper bit(s)**

If number is negative,

- **start with positive number (i.e. ignore the sign)**
- **flip every bit (i.e., take the one's complement)**
- **then add one**



What is the 5-bit 2's complement representation of 14?

What is the 5-bit 2's complement representation of -10?

Given a 2's complement number X, how to get the 2's complement of –X?

- **Compute the 2's complement of X**
  - **invert each bit of X (i.e. 1's complement of X)**
  - **add one to the result of the inversion**

01101	(X)	11001	(X)
10010	(1's comp)		(1's comp)
+ 1		+ 1	
10011	(-X)		(-X)

- **Knowing this is useful as a computer normally does not provide subtraction operation**
  - **X-Y = X+(-Y)**

Two's Complement Signed Integers

MS bit is sign bit

There is only one representation for 0

Range of an n-bit number: -2^{n-1} through $2^{n-1} - 1$.

- **The most negative number (-2^{n-1}) has no positive counterpart.**

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

Converting Binary (2's C) to Decimal

1. **If leading bit is one, take two's complement to get a positive number.**
2. **Represent the number as a power of 2 expression and calculate the value of the expression**
3. **If original number was negative, add a minus sign.**

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

Convert an 8-bit 2's complement number:

$$X = 01101000_{two}$$
$$= 1*2^6+1*2^5+0*2^4+1*2^3+0*2^2+0*2^1+0*2^0$$
$$= 2^6+2^5+2^3$$
$$= 64+32+8$$
$$= 104_{ten}$$

More Examples

$X = 00100111_{two}$
$=$
$=$

$X = 11100110_{two}$
$-X = 00011010$
$= 2^4+2^3+2^1 = 16+8+2$
$= 26_{ten}$
$X = -26_{ten}$

$11110110_{two}=$

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

Converting Decimal to Binary (2's C)

Repeated *Division*

1. **Find magnitude of decimal number. (ignore the sign)**
2. **Divide by two – the remainder is the least significant bit and the quotient is used in the next round of division.**
3. **Keep dividing by two until quotient is zero, writing remainders from right to left.**
4. **Append a zero as the MS bit if the original number was positive; if original number was negative, take two's complement.**

$X = 104_{ten}$

104/2	=	52 r0	*bit 0*
52/2	=	26 r0	*bit 1*
26/2	=	13 r0	*bit 2*
13/2	=	6 r1	*bit 3*
6/2	=	3 r0	*bit 4*
3/2	=	1 r1	*bit 5*
1/2	=	0 r1	*bit 6*

$X = 01101000_{two}$

$X = -16_{ten}$

Get the magnitude 16_{ten}

$16_{ten} = 00010000_{two}$

Since the original number is negative, compute the 2's complement to get 11110000_{two}

So, $X = -16_{ten} = 11110000_{two}$

32_{10} =
-12_{10} =

Agenda

1. **Why does computer use binary numbers**
2. **Conversion between unsigned decimal and binary numbers**
3. **Representing signed numbers**
4. **Arithmetic Operations**
   - **Addition**
   - **Subtraction**
   - **Sign Extension**
   - **Overflow**
5. **Logical Operations**
6. **Shifting**
7. **Hexadecimal & Octal Notation**
8. **ASCII characters**

Addition

2's comp. addition is just binary addition.

- **assume all integers have the same number of bits**
- **Sign bit is treated the same as other bits**
- **for now, assume that sum fits in n-bit 2's comp. representation**
- **ignore carry out**

Assuming 8-bit 2's complement numbers.

```
  01101000  (104)        11110110  (-10)
+ 11110000  (-16)     +            (-9)
  01011000  (88)                   (-19)
```

Subtraction

assume all integers have the same number of bits

X – Y = X + (-Y)

- **"–Y" is obtained by calculating Y's 2's complement**

for now, assume that difference fits in n-bit 2's comp. representation

ignore carry out

Assuming 8-bit 2's complement numbers.

	01101000	(104)		11110110	(-10)
–	00010000	(16)	–		(-9)
	01101000	(104)		11110110	(-10)
+	11110000	(-16)	+		(9)
	01011000	(88)			(-1)

Sign Extension

To add two numbers, we must represent them with the same number of bits.

If we just pad with zeroes on the left:

4-bit		**8-bit**	
0100	(4)	00000100	(still 4)
1100	(-4)	00001100	(12, not -4)

Instead, replicate the MS bit -- the sign bit:

4-bit		**8-bit**	
0100	(4)	00000100	(still 4)
1100	(-4)	11111100	(still -4)

Overflow

If operands are too big, then sum cannot be represented as an *n*-bit 2's comp number.

	01000	(8)		11000	(-8)
+	01001	(9)	+	10111	(-9)
	10001	(-15)		01111	(+15)

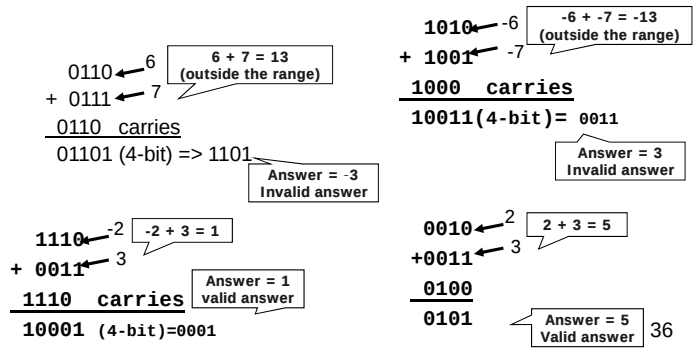
We have overflow if:

- **signs of both operands are the same, and**
- **sign of sum is different.**

Overflow

Example: 4-bit

- **Two's complement - 8 <= x <= 7**
- **Examples:**



Agenda

1. Why does computer use binary numbers
2. Conversion between unsigned decimal and binary numbers
3. Representing signed numbers
4. Arithmetic Operations
5. Logical Operations
6. Shifting
7. Hexadecimal & Octal Notation
8. ASCII characters

Logical Operations

Operations on logical TRUE or FALSE

- two states -- takes one bit to represent: TRUE=1, FALSE=0

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

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

A	NOT A
0	1
1	0

View *n*-bit number as a collection of *n* logical values

- operation applied to each bit independently

Examples of Logical Operations

AND

- useful for clearing bits
  - AND with zero = 0
  - AND with one = no change

```
  11000101
AND 00001111
  00000101
```

OR

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

```
  11000101
OR 00001111
  11001111
```

NOT

- unary operation -- one argument
- flips every bit
- Obtain 1's complement of the number

```
NOT 11000101
  00111010
```


Bit Operations: And

Assume 2's complement 8 bits representation from this point

Example

```
  0011 1100
 &0001 1111
```

Exercise

```
  1010 0001
 &0101 1111
```

Or

Example

```
  0011 1100
 |0001 1111
```

Exercise

```
  1010 0001
 |0101 1111
```


Agenda

1. **Why does computer use binary numbers**
2. **Conversion between unsigned decimal and binary numbers**
3. **Representing signed numbers**
4. **Arithmetic Operations**
5. **Logical Operations**
6. **Shifting**
7. **Hexadecimal & Octal Notation**
8. **ASCII characters**

Shifting

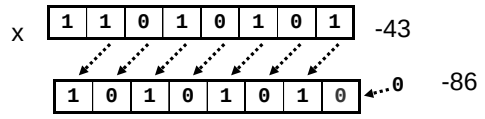
Shift operators move bits to the left or to the right.

- **Used to shift the bit patterns left and right.**
- **Shift a number to right/left by one bit corresponds to divide/multiply the number by 2**
- **left shift and right shift corresponding to << and >>.**
  - **The first operand is the value to be shifted**
  - **The second gives the number of bit positions to shift**
    - **0111 >> 1 (shift 0111 to the right by one bit)**
- **The right shift fills the vacated bits with the sign bit.**
- **The left shift instruction fills the vacated bits with zeros.**

Left shift <<

Examples

- **1101 0101 << 1**

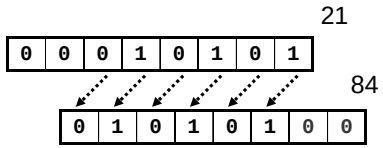


What is the result of
1101 0101 << 2

Left shift <<

Examples

- **0001 0101 << 2**



Left Shift

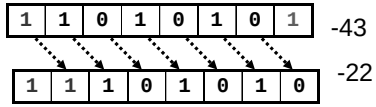
Exercise

0001 0101 << 2

Right shift >>

Examples

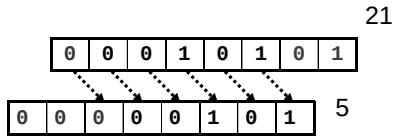
- **1101 0101 >> 1**



Right shift >>

Examples

- **0001 0101 >> 2**



Right Shift

Exercise

0001 0101 >> 2

Agenda

1. **Why does computer use binary numbers**
2. **Conversion between unsigned decimal and binary numbers**
3. **Representing signed numbers**
4. **Arithmetic Operations**
5. **Logical Operations**
6. **Shifting**
7. **Hexadecimal & Octal Notation**
8. **ASCII characters**

Hexadecimal Notation

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 corrupt long string of 1's and 0's**
- **each hexadecimal digit has a range [0 .. 15]**
- **A hexadecimal number has a 0x prefix, e.g. 0x12A9**

Binary	Hex	Decimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7

Binary	Hex	Decimal
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

This is not a new machine representation, just a convenient way to write the number.

Converting from Binary to Hexadecimal

Every four bits is a hex digit.

- **start grouping from right-hand side:**

01110101000111101001101011 1

011 1010 1000 1111 0100 1101 0111

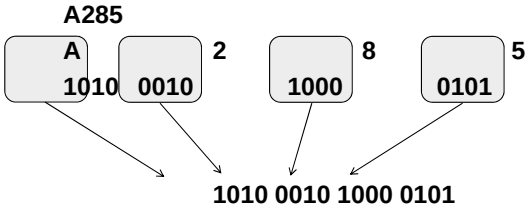
⇩ ⇩ ⇩ ⇩ ⇩ ⇩ ⇩

3 A 8 F 4 D 7

1011001010000101_2 =

Converting from Hexadecimal to Binary

Convert each hexadecimal digit to a 4-bit binary number and combine all the numbers together



$18AC_{16}$ =

Octal Notation

It is often convenient to write binary (base-2) numbers as octal (base-8) numbers instead.

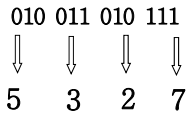
- **fewer digits – three bits per octal digit**
- **Used to denote the permission code in Unix**

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

Converting from Binary to Octal

Every three bits is a octal digit.

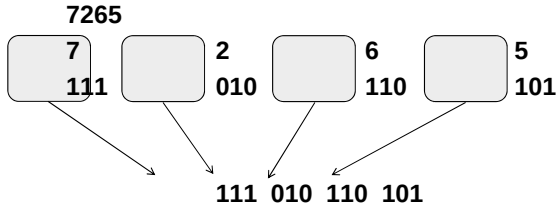
- **start grouping from right-hand side:** 010011010111



001010000101_2 =

Converting from Octal to Binary

Convert each octal digit to a 3-bit binary number and combine all the numbers together



1476_8 =

Agenda

1. Why does computer use binary numbers
2. Conversion between unsigned decimal and binary numbers
3. Representing signed numbers
4. Arithmetic Operations
5. Logical Operations
6. Shifting
7. Hexadecimal & Octal Notation
8. ASCII characters

Text: ASCII Characters

ASCII: Maps 128 characters to 7-bit code.

- both printable and non-printable (ESC, DEL, …) characters

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

Hexadecimal characters ASCII code

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', …)?

Are 128 characters enough? (http://www.unicode.org/)

7 bits can only represent 128 characters

we also need to represent characters in languages other than English

- **unicode: 16 bits to represent a character**