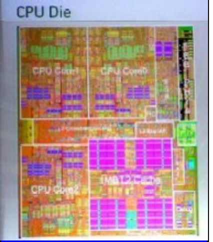


COMP 273

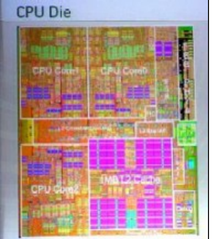
Introduction to Computer Systems

COMP 273

Digital Logic (Part 3)

Data Representation and Logic Gates

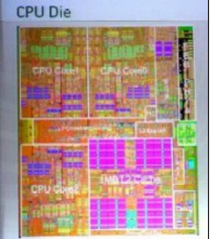
Prof. Joseph Vybihal



COMP 273
Introduction to Computer Systems

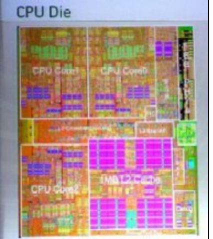
Announcements

- Assignment #1 is given out
- TA Information on web site
- Tutorial:
  - Logisim
    - Thu Jan 22 at 2pm in TR 3120
    - Mon Jan 26 at 3pm in TR 3120



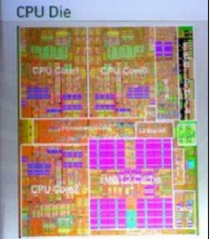
At Home

- Readings:
  - Wikipedia Unicode and ASCII
  - Wikipedia Logic Gates
  - http://www.kpsec.freeuk.com/gates.htm
  - http://www.ee.surrey.ac.uk/Projects/Labview/gatesfunc/index.html#example
- Try out exercises from the textbook
  - Fixed point mathematics
- Experiment
  - Download Logisim (Google / My Courses)



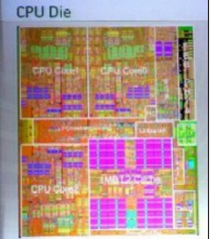
Today's Class

- Data Representation (data types)
- Introduction to Logic Gates
  - Basic gates
  - Flip Flops



Part 1

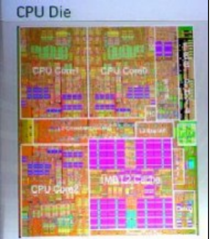
Standard Computer Representation of Data



COMP 273
Introduction to Computer Systems

Registers vs RAM

- Register
  - A special machine that stores a single value, possibly formatted
    - Formatted of IR, or
    - formatted GP Register
- RAM
  - General purpose memory with no special format but organized in 8-bit byte units each of which are addressable
    - Like an array



First Standard Type: **Characters**

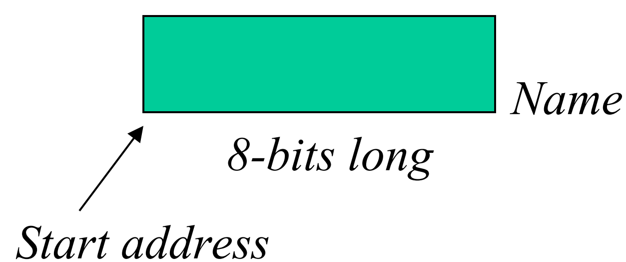
Tabulated Binary Encoding Standard

Ctrl	Dec	Hex	Char	Code	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
^@	0	00		NUL	32	20		64	40	@	96	60	`
^A	1	01		SOH	33	21	!	65	41	A	97	61	a
^B	2	02		STX	34	22	"	66	42	B	98	62	b
^C	3	03		ETX	35	23	#	67	43	C	99	63	c
^D	4	04		EOT	36	24	$	68	44	D	100	64	d
^E	5	05		ENQ	37	25	%	69	45	E	101	65	e
^F	6	06		ACK	38	26	&	70	46	F	102	66	f
^G	7	07		BEL	39	27	'	71	47	G	103	67	g
^H	8	08		BS	40	28	(	72	48	H	104	68	h
^I	9	09		HT	41	29	)	73	49	I	105	69	i
^J	10	0A		LF	42	2A	*	74	4A	J	106	6A	j
^K	11	0B		VT	43	2B	+	75	4B	K	107	6B	k
^L	12	0C		FF	44	2C	,	76	4C	L	108	6C	l
^M	13	0D		CR	45	2D	-	77	4D	M	109	6D	m
^N	14	0E		SO	46	2E	.	78	4E	N	110	6E	n
^O	15	0F		SI	47	2F	/	79	4F	O	111	6F	o
^P	16	10		DLE	48	30	0	80	50	P	112	70	p
^Q	17	11		DC1	49	31	1	81	51	Q	113	71	q
^R	18	12		DC2	50	32	2	82	52	R	114	72	r
^S	19	13		DC3	51	33	3	83	53	S	115	73	s
^T	20	14		DC4	52	34	4	84	54	T	116	74	t
^U	21	15		NAK	53	35	5	85	55	U	117	75	u
^V	22	16		SYN	54	36	6	86	56	V	118	76	v
^W	23	17		ETB	55	37	7	87	57	W	119	77	w
^X	24	18		CAN	56	38	8	88	58	X	120	78	x
^Y	25	19		EM	57	39	9	89	59	Y	121	79	y
^Z	26	1A		SUB	58	3A	:	90	5A	Z	122	7A	z
^[	27	1B		ESC	59	3B	;	91	5B	[	123	7B	{
^\	28	1C		FS	60	3C	<	92	5C	\	124	7C	|
^]	29	1D		GS	61	3D	=	93	5D	]	125	7D	}
^^	30	1E	▲	RS	62	3E	>	94	5E	^	126	7E	~
^-	31	1F	▼	US	63	3F	?	95	5F	_	127	7F	⌂*

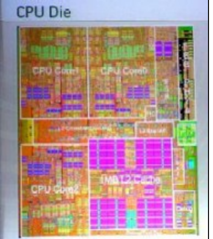
* ASCII code 127 has the code DEL. Under MS-DOS, this code has the same effect as ASCII 8 (BS). The DEL code can be generated by the CTRL + BKSP key.

ASCII or UNICODE

SPACE = 32 = 00100000
A = 65 = 01000001
a = 97 = 01100001
8 bits



No sign bit



Second Standard Type: **Strings**

Contiguous sequence of characters terminated by NULL, or
Contiguous sequence of chars proceeded by a byte count.

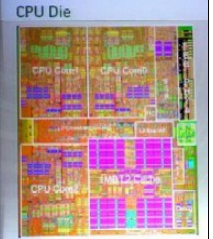
E.G. HELLO = 72, 69, 76, 76, 79

NULL

With NULL = 01001000 01000101 01001100 01001100 01001111 00000000
With Count = 00000101 01001000 01000101 01001100 01001100 01001111

5 bytes

Strings are composed of char.
Char is a built in property of the CPU not strings.
Strings are supported through <u>software</u> – Combining char & integer



COMP 273

Introduction to Computer Systems

Third Standard Type: **Integer**

Let *size* be a fixed number of bits, “the size in bits” then a number is represented in raw signed binary or 2’s comp.

Sign bit

S	Magnitude

Bit size

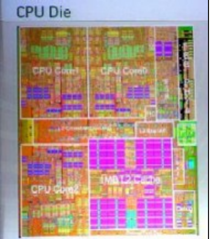
Sign bit

2’s Complement

Bit size

Used by ALU for subtraction

16, 32 or 64 bits, normally



Fourth Standard Type: **Fixed Point**

I.E. $1000.25 = 0.100025 \times 10^4$

Sign

0	4	100025

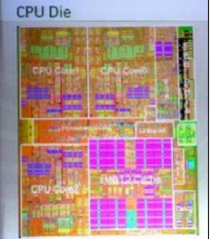
Exponent *Mantissa*

0 = +ve
1 = -ve

Raw Binary + Offset

Not two's complement

Note: do not need to store the leading zero and decimal place.



COMP 273
Introduction to Computer Systems

Basic Format

31	30 29 28 27 26 25 24 23	22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
s	exponent	significand
1 bit	8 bits	23 bits

32 bit version

In general, floating-point numbers are of the form

$$(-1)^S \times F \times 2^E$$

31	30 29 28 27 26 25 24 23 22 21 20	19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
s	exponent	significand
1 bit	11 bits	20 bits

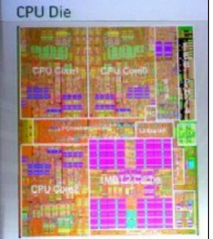
significand (continued)
32 bits

64 bit version

The IEEE format

The IEEE format comes with 3 different levels of precision:
Single, **Double**, and **Quad**

		Single	Double	Quad
Number of bits taken by:				
	Sign	1	1	1
	Exponent	8	11	15
	Fractional mantissa	23	52	111
	Total	32	64	128
Exponent:				
	Bias	127	1023	16383
	Range of (biased) exponent:	0..255	0..2047	0..32767



$$(-1)^S \times (1 + Sig) \times 2^{(Exp + Bias)}$$

$\frac{1}{8} = \frac{1}{2^3} = \frac{1}{1000_2} = 0.001_2$

a)

$-69.125_{10} = 1000101\,.\,001_2 = 1.000101001 * 2^6$

Mantissa:
Throw away the leading 1 to end up with a fractional mantissa of 000101001

True exponent : 6 (note we can drop the 1 bit as well)
Biased exponent = $6 + 127 = 133_{10} = 10000101_2$.
The number is negative, the sign bit is 1.

Write the sign bit, exponent and mantissa in the IEEE format:

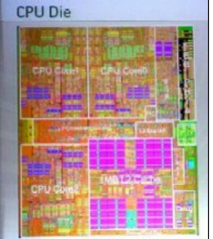
1 10000101 00010100100000000000000

Pack the three groups of bits into one word to get the final answer:
11000010100010100100000000000000

It is good practice to represent the final answer in hexadecimal : **C28A4000**

In RAM

Note that the fractional mantissa had only 9 bits, so we had to add zeros at the right end to get a 23-bit mantissa as required by the IEEE single precision format.



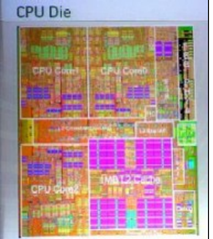
b)

In double precision, bias = 1023.
True exponent : 6
Biased exponent : $6 + 1023 = 1029_{10} = 10000000101_2$

The fractional mantissa is still 000101001, you just need to add enough zeros at its right end to get a 52-bit mantissa.

Sign bit : 1

The answer is:
$1\ 10000000101\ 000101001000....0_2 = \mathbf{C051480000000000_{16}}$



Basic Representation

Example Show the IEEE 754 binary representation of the number -0.75_{ten} in single and double precision.

Answer The number -0.75_{ten} is also

$$-3/4_{ten} \text{ or } -3/2^2{}_{ten}$$

It is also represented by the binary fraction:

$$-11_{two}/2^2{}_{ten} \text{ or } -0.11_{two}$$

In scientific notation, the value is

$$-0.11_{two} \times 2^0$$

and in normalized scientific notation, it is

$$-1.1_{two} \times 2^{-1}$$

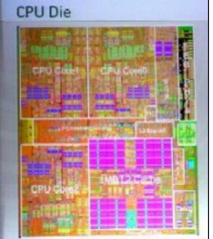
The general representation for a single precision number is

$$(-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent} - 127)}$$

and so when we add the bias 127 to the exponent of $-1.1_{two} \times 2^{-1}$, the result is

$$(-1)^1 \times (1 + .1000\ 0000\ 0000\ 0000\ 0000\ 000_{two}) \times 2^{(126-127)}$$

The single precision binary representation of -0.75_{ten} is then



COMP 273
Introduction to Computer Systems

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 bit 8 bits 23 bits

The double precision representation is

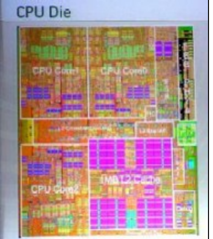
$$(-1)^1 \times (1 + .1000\,0000\,0000\,0000\,0000\,0000\,0000\,0000\,0000\,0000\,0000\,0000\,0000_{two}) \times 2^{(1022-1023)}$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

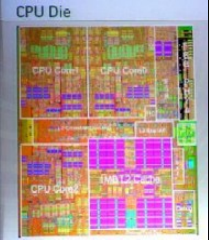
1 bit 11 bits 20 bits

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

32 bits

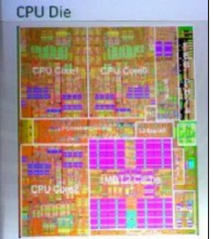


Addition and Multiplication with real numbers



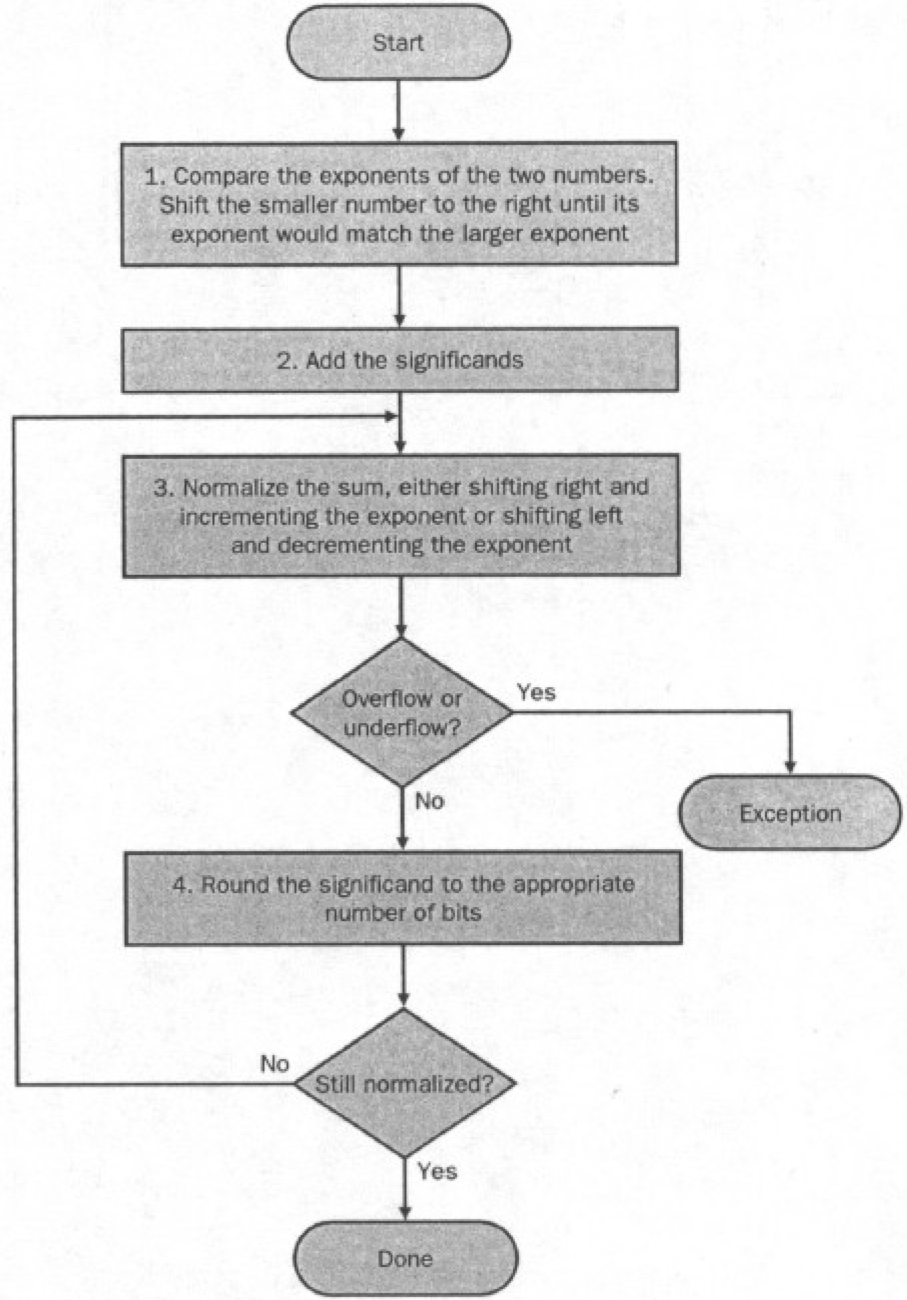
Basic Addition Algorithm

1. Normalize number
2. Round if necessary
3. Shift values to same exponent
4. Add bases
5. Normalize result
6. Round if necessary
7. Update the sign



COMP 273
Introduction to Computer Systems

Addition Algorithm



Addition Example

Example Try adding the numbers 0.5_{ten} and -0.4375_{ten} in binary using the algorithm in Figure 4.44.

Answer Let's first look at the binary version of the two numbers in normalized scientific notation, assuming that we keep 4 bits of precision:

$$
\begin{aligned}
0.5_{ten} &= 1/2_{ten} &&= 1/2^1{}_{ten} \\
&= 0.1_{two} &&= 0.1_{two} \times 2^0 &&= 1.000_{two} \times 2^{-1} \\
-0.4375_{ten} &= -7/16_{ten} &&= -7/2^4{}_{ten} \\
&= -0.0111_{two} &&= -0.0111_{two} \times 2^0 &&= -1.110_{two} \times 2^{-2}
\end{aligned}
$$

Now we follow the algorithm:

Step 1. The significand of the number with the lesser exponent ($-1.11_{two} \times 2^{-2}$) is shifted right until its exponent matches the larger number:

$$-1.110_{two} \times 2^{-2} = -0.111_{two} \times 2^{-1}$$

COMP 273
Introduction to Computer Systems

Step 2. Add the significands:

$$1.0_{two} \times 2^{-1} + (-0.111_{two} \times 2^{-1}) = 0.001_{two} \times 2^{-1}$$

Step 3. Normalize the sum, checking for overflow or underflow:

$$\begin{aligned} 0.001_{two} \times 2^{-1} &= 0.010_{two} \times 2^{-2} = 0.100_{two} \times 2^{-3} \\ &= 1.000_{two} \times 2^{-4} \end{aligned}$$

Since $127 \geq -4 \geq -126$, there is no overflow or underflow. (The biased exponent would be $-4 + 127$, or 123, which is between 1 and 254, the smallest and largest unreserved biased exponents.)

Step 4. Round the sum:

$$1.000_{two} \times 2^{-4}$$

The sum already fits exactly in 4 bits, so there is no change to the bits due to rounding.

This sum is then

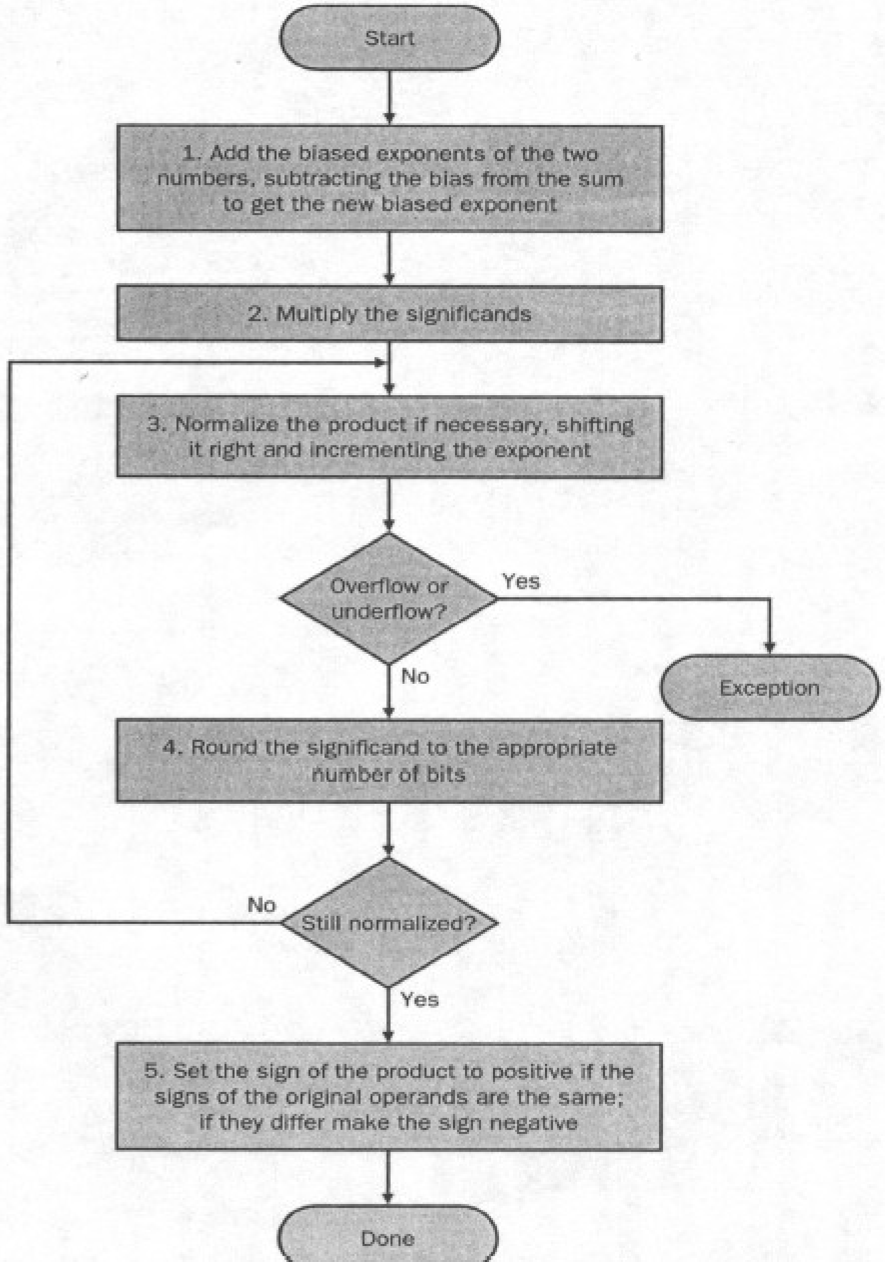
$$\begin{aligned} 1.000_{two} \times 2^{-4} &= 0.0001000_{two} = 0.0001_{two} \\ &= 1/2^4{}_{ten} = 1/16_{ten} = 0.0625_{ten} \end{aligned}$$

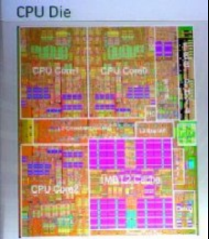
This sum is what we would expect from adding 0.5_{ten} to -0.4375_{ten}.

Basic Multiplication Algorithm

1. Remove exponent bias
2. Match exponents
3. Multiply bases
4. Normalize and round if needed
5. Update the sign

Multiplication Algorithm





Multiplication Example

Example

Let's try multiplying the numbers 0.5_{ten} and -0.4375_{ten} using the steps in Figure 4.46.

Answer

In binary, the task is multiplying $1.000_{two} \times 2^{-1}$ by $-1.110_{two} \times 2^{-2}$.

Step 1. Adding the exponents without bias:

$$-1 + (-2) = -3$$

or, using the biased representation:

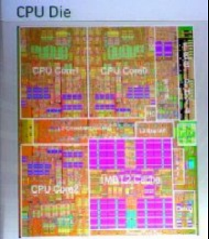
$$(-1 + 127) + (-2 + 127) - 127 = (-1 - 2) + (127 + 127 - 127)$$
$$= -3 + 127 = 124$$

Step 2. Multiplying the significands:

$$\begin{array}{r}
1.000_{two} \\
\times \quad 1.110_{two} \\
\hline
0000 \\
1000 \\
1000 \\
1000 \\
\hline
1110000_{two}
\end{array}$$

The product is $1.110000_{two} \times 2^{-3}$, but we need to keep it to 4 bits, so it is $1.110_{two} \times 2^{-3}$.

Step 3. Now we check the product to make sure it is normalized, and then check the exponent for overflow or underflow. The product is already normalized and, since $127 \geq -3 \geq -126$, there is no overflow or underflow. (Using the biased representation, $254 \geq 124 \geq 1$, so the exponent fits.)



Step 4. Rounding the product makes no change:

$$1.110_{\text{two}} \times 2^{-3}$$

Step 5. Since the signs of the original operands differ, make the sign of the product negative. Hence the product is

$$-1.110_{\text{two}} \times 2^{-3}$$

Converting to decimal to check our results:

$$-1.110_{\text{two}} \times 2^{-3} = -0.001110_{\text{two}} = -0.00111_{\text{two}}$$
$$= -7/2^5{}_{\text{ten}} = -7/32_{\text{ten}} = -0.21875_{\text{ten}}$$

The product of 0.5_{ten} and -0.4375_{ten} is indeed -0.21875_{ten}.

Fifth Standard Type: **Logical**

A single bit value: 0 = false and 1 = true

A	B	A and B
0	0	0
0	1	0
1	0	0
1	1	1

Clears Bits

A	B	A or B
0	0	0
0	1	1
1	0	1
1	1	1

Sets Bits

A	B	A xor B
0	0	0
0	1	1
1	0	1
1	1	0

Toggles Bits

A	not A
0	1
1	0

Inverts Bits

Sixth Standard Type: **Packed Decimal**

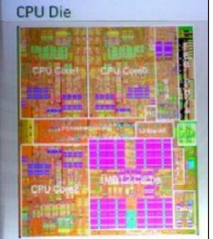
Use a nibble to represent each <u>digit</u> not entire binary number.

The Packed Decimal (or BCD, Binary Coded Decimal) code is a special way of representing decimal numbers in binary. It simply replaces each decimal digit by its 4-bit equivalent.

A number written in Packed Decimal is a number written in decimal, where the individual digits of the number are written in binary.

Example:

5372_{10} is written in Packed Decimal as 0101 0011 0111 0010.

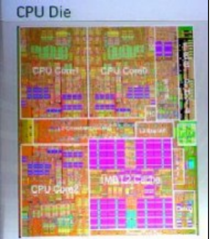


Do not confuse Packed Decimal arithmetic with Binary arithmetic:

Decimal	Packed Decimal	Binary
123	0001 0010 0011	001111011
+	+	+
289	0010 1000 1001	100100001
———	———————	———————
412	0100 0001 0010	110011100

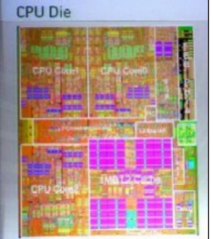
Advantage: very user-friendly

Disadvantages: wastes storage space
complicates hardware



Part 2

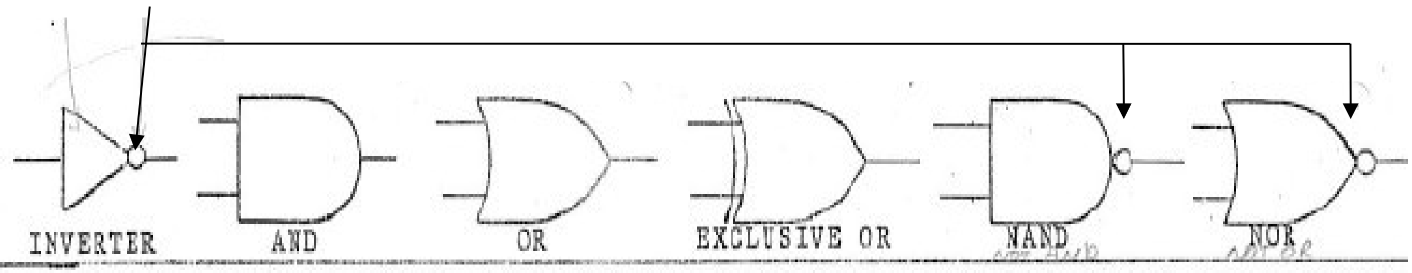
Logic Gates



COMP 273
Introduction to Computer Systems

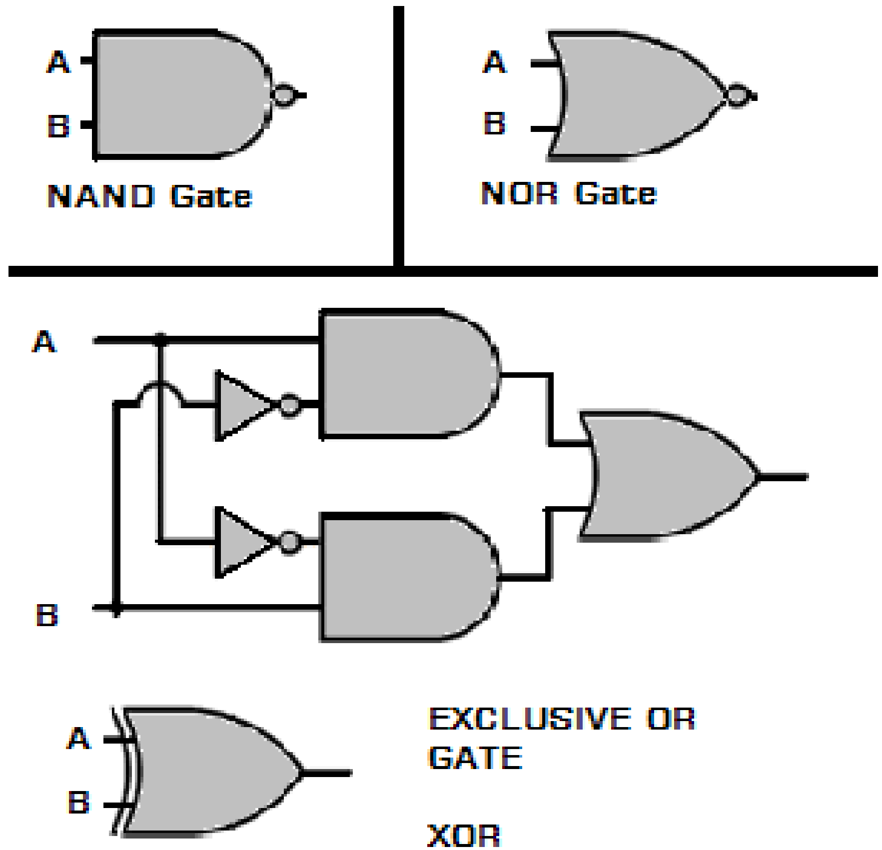
Basic Logic Circuits

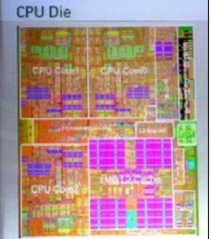
Means inverts



A	B	INVERTER	AND	OR	EXCLUSIVE OR	NAND	NOR
0	0	1	0	0	0	1	1
0	1		0	1	1	1	0
1	0	0	0	1	1	1	0
1	1		1	1	0	0	0

Uni-directional
(in this case left to right)


A
B
NAND Gate
A
B
NOR Gate
A
B
A
B
EXCLUSIVE OR
GATE
XOR

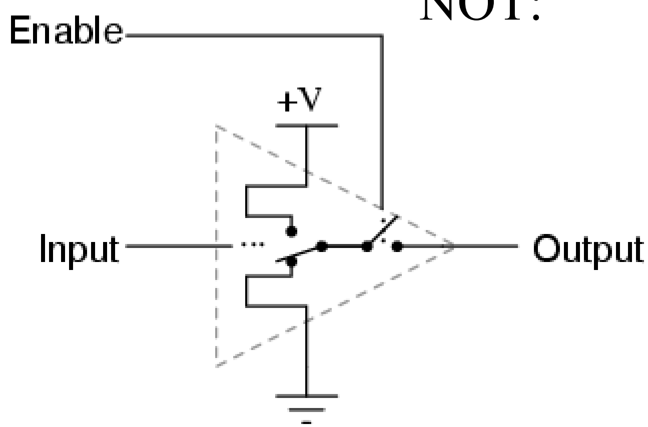


COMP 273
Introduction to Computer Systems

Low-level Construction

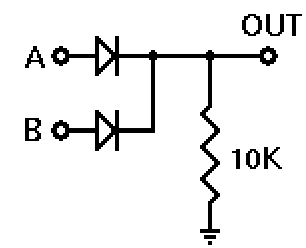
NOT:

Tristate buffer gate



OR: → C → *Output*

Connector

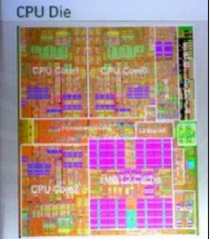


DL Circuit for OR
(Diode-Logic)

AND:

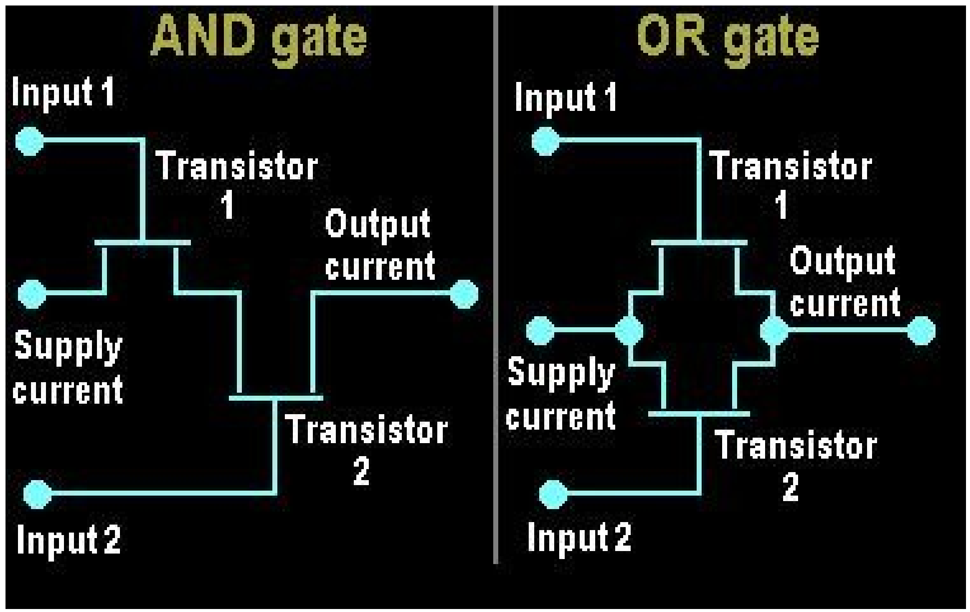
→ inverter → S →

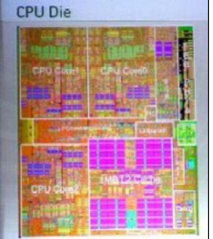
→ inverter → S → *Output*



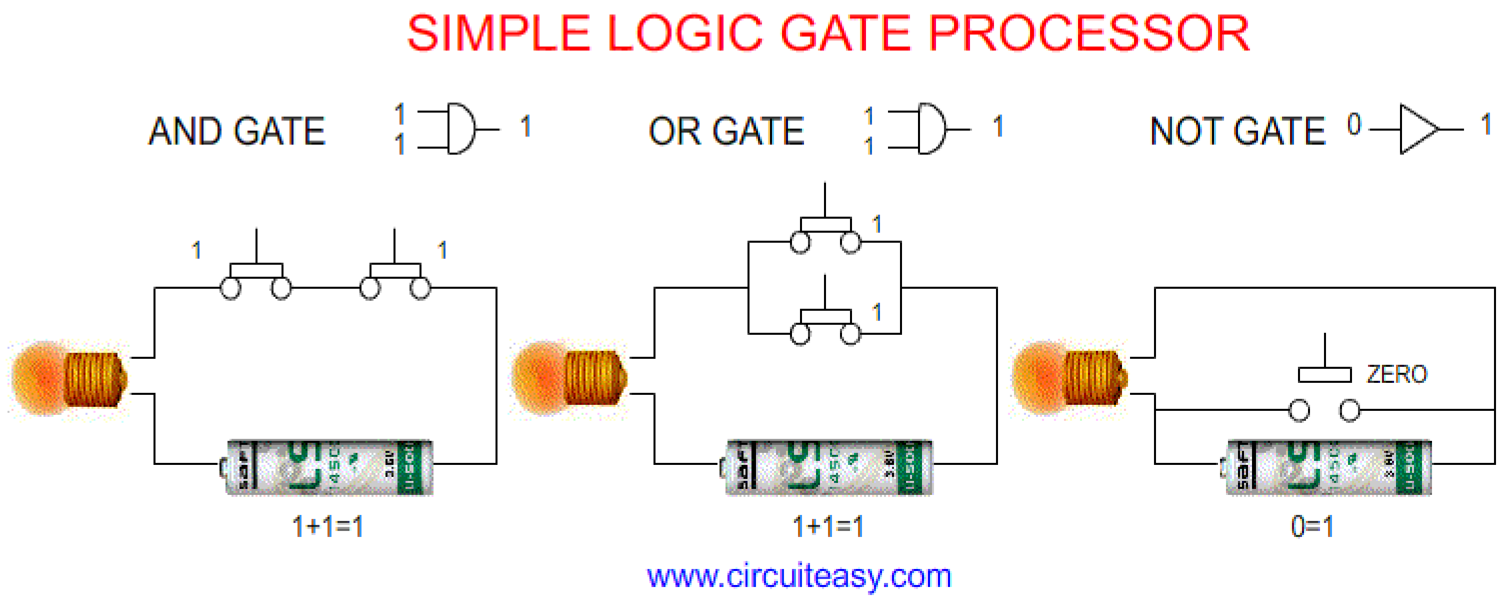
COMP 273
Introduction to Computer Systems

Low-level Construction

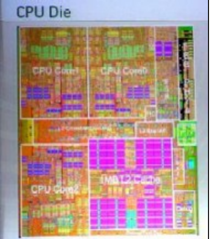




Low-level Construction

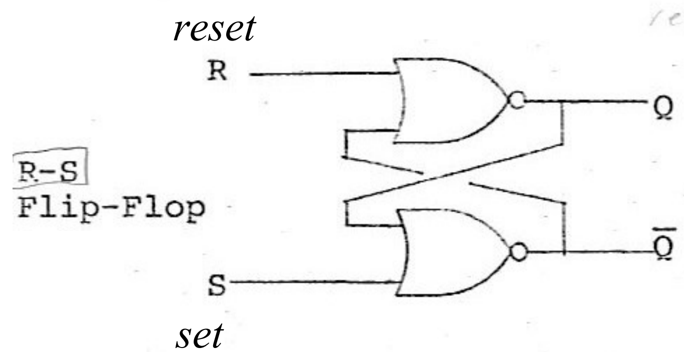


http://www.circuiteasy.com/animation/logic_gate.gif

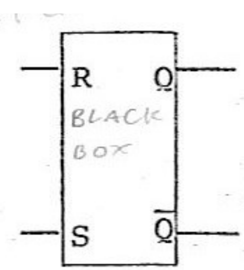


COMP 273
Introduction to Computer Systems

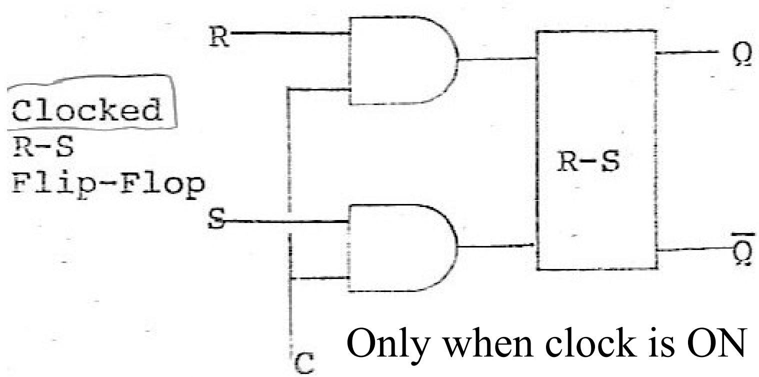
Basic Flip Flop Circuits

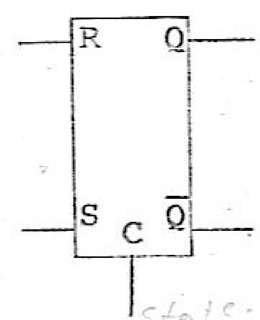


The Output



This is a Bit

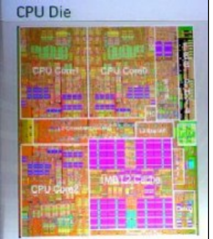




Only when clock is ON does the state change.

When R=1, S must be 0, When S=1, R must be 0.

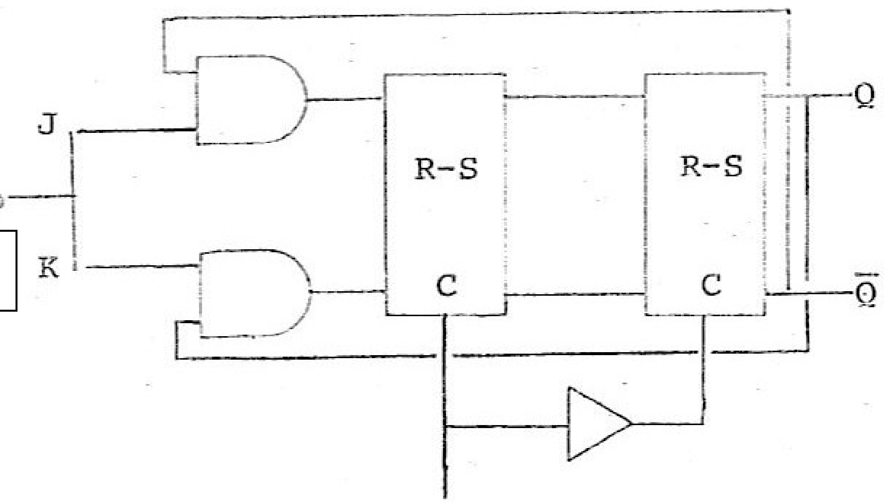
COMP 273
Introduction to Computer Systems

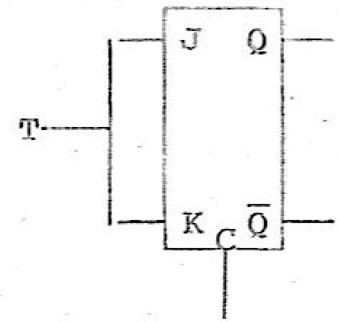


Basic Flip Flop Circuits

Clocked
J-K
Flip-Flop

(Counter)

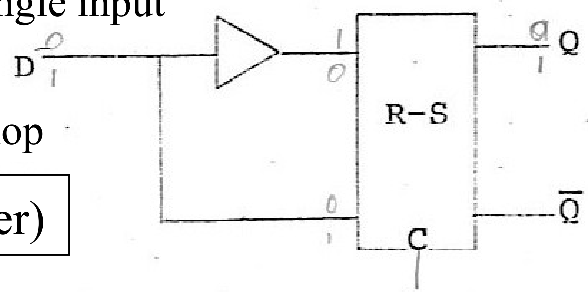


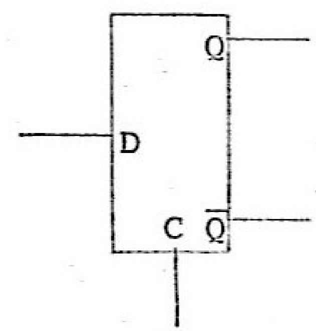


Single input

D Flip-Flop

(Register)





Arbitrary JPL engineer assignment of letters: J mean set and K mean reset;
J=1,K=0 SET; J=0,K=1 RESET; J=1,K=1 TOGGLE; J=0,K=0 NOTHING.