An Introduction to Microprocessors

Costas Foudas, Imperial College,
Rm: 508, x47590

Outline:

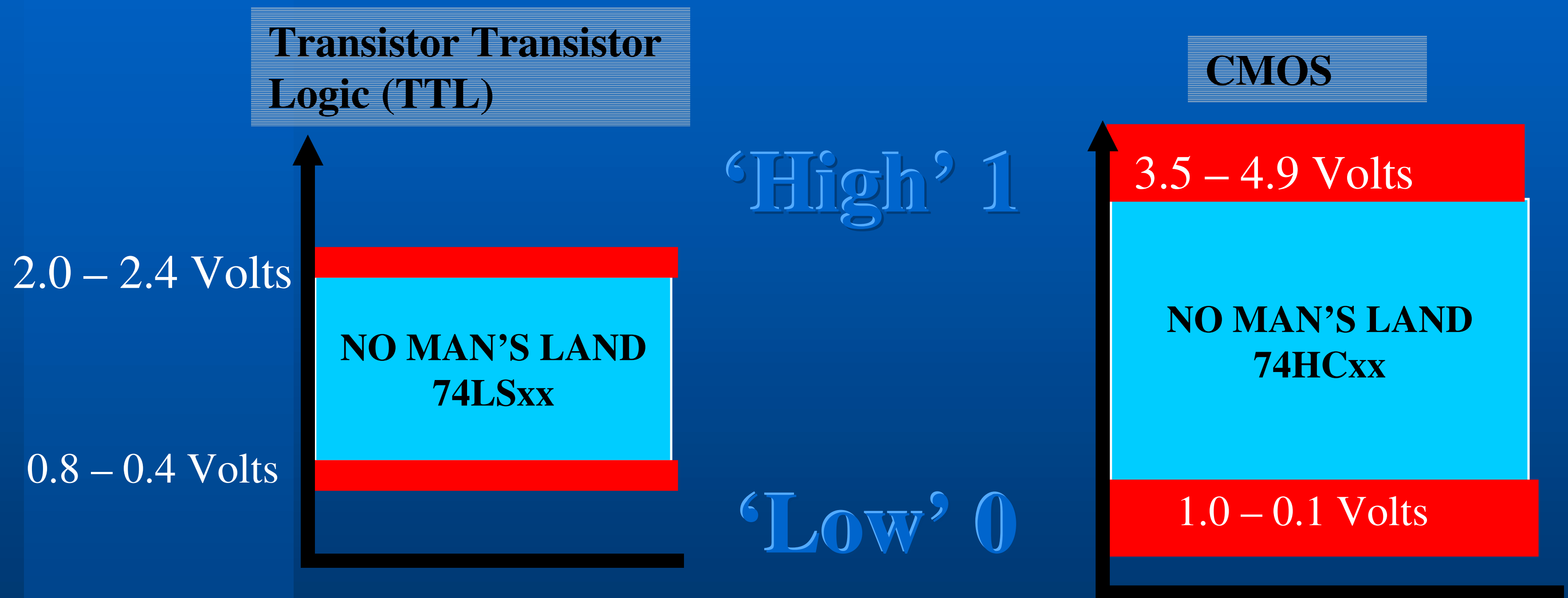
- Numerical representations
- Registers and Adders
- ATmega103 architecture
- AVR assembly language
- Elementary example program
- STUDIO3.52
- AVR Assembler
- Downloading with PonyProg

Numbers in a computer:

- Computers store arithmetic units called ***bits.***
- Each bit is represented by an electrical signal which is either ***high*** or ***low*** (voltage levels).
- ***high*** $\Rightarrow$ 1 and ***low*** $\Rightarrow$ 0 Normal Logic
- Often ***high*** $\Rightarrow$ 0 and ***low*** $\Rightarrow$ 1 Inverse Logic

High and Low

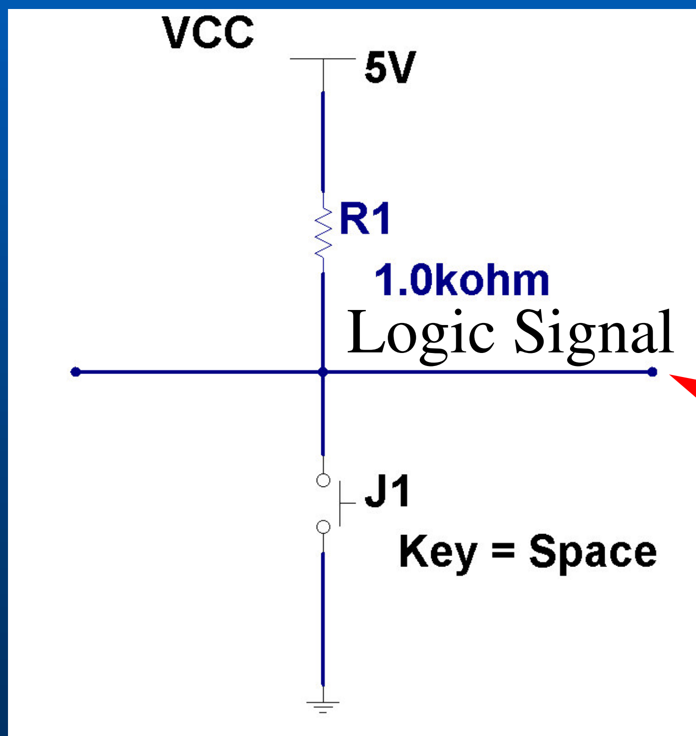
In this course we use TTL or TTL-like (CMOS) technology :



It is actually a bit more complicated than this since there are actually different thresholds for inputs and outputs and noise margins (indicated here in RED) but this may be addressed later If time permits.

Making a Zero or and One

- **How do you actually make a 0 or 1 ?**

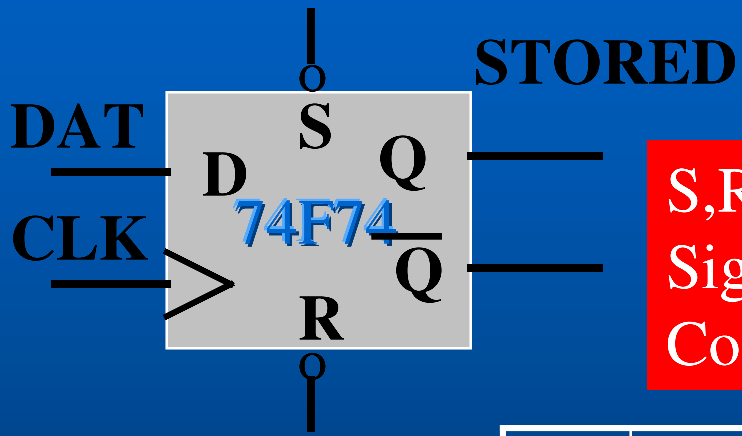


It is clear that depending upon the switch position the line will be either ‘0’ or ‘1’

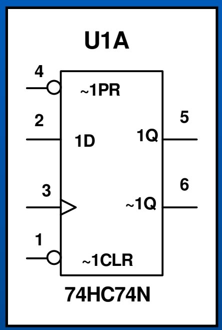
Storing Zeros and Ones: Registers

- **Registers** are electronic devices capable of storing 0 or 1
- **D-FLIP-FLOPs** are the most elementary registers which can store one bit
- **8 DFFs** clocked together make an one byte register

The D-Flip-Flop (DFF)



S,R, Q-bar are Signals with Complement Logic



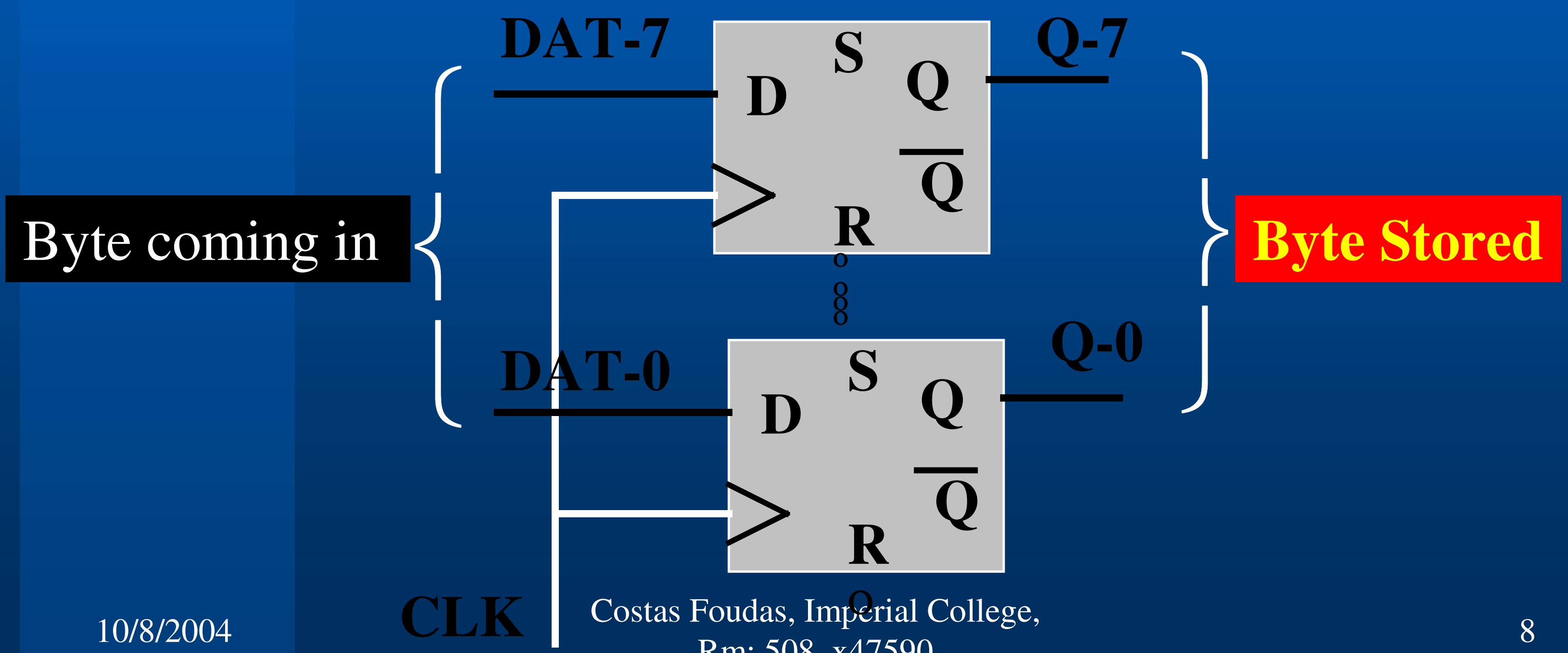
One can Set or Reset (Clear) the DFF using S or R

On the rising edge of the clock the data are transferred and stored in Q.

CLK	D	S	R	Q	!Q
X	X	L	H	H	L
X	X	H	L	L	H
↑	H	H	H	H	L
↑	L	H	H	L	H
X	X	L	L	H	H

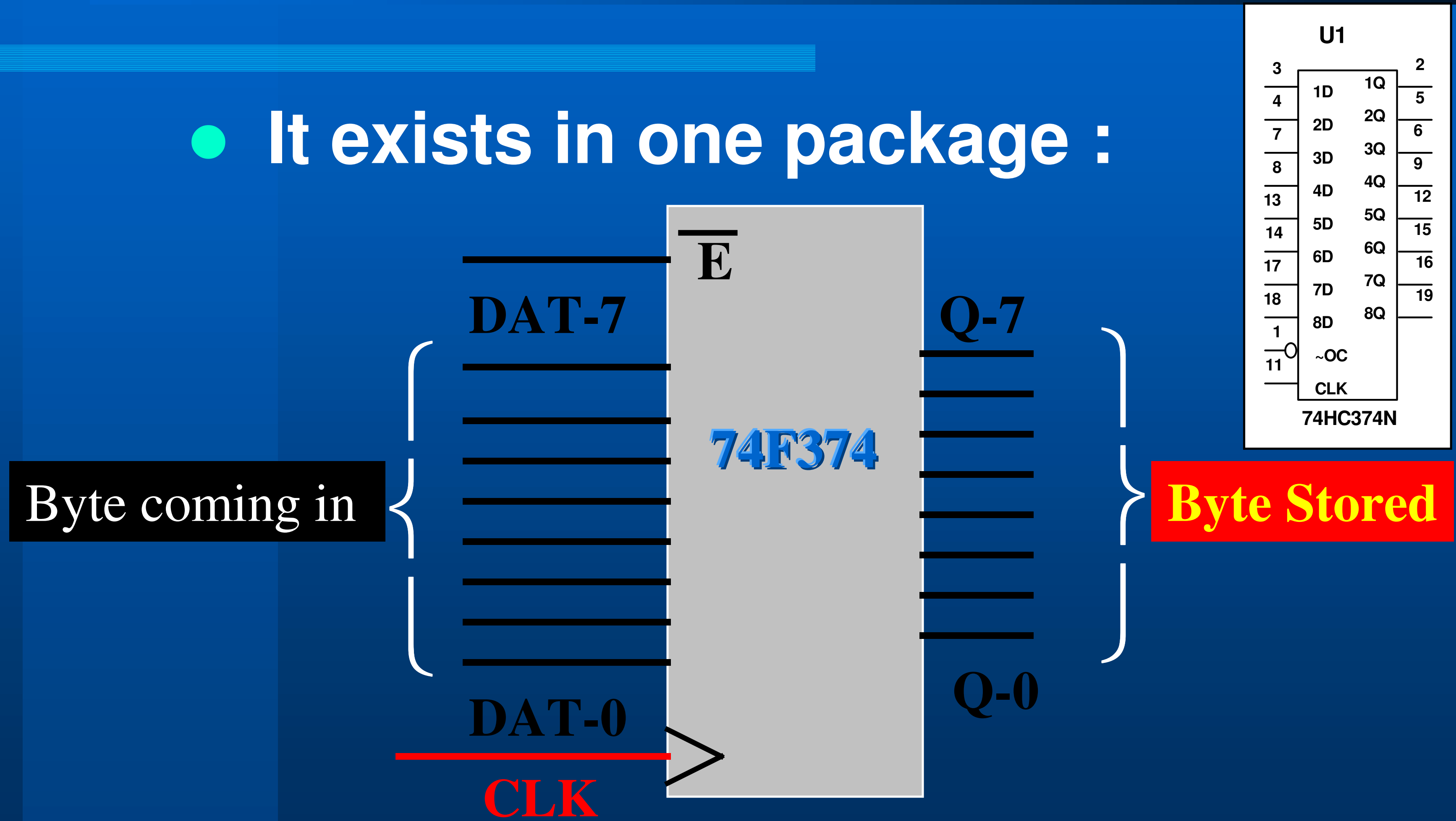
Byte Register

Byte register that stores a byte



Byte Register I

- It exists in one package :



Bits & Bytes :

- Bit 1,0
- Nibble 4 bits
- Byte 8 bits
- Word 16 bits or 2 bytes
- 1Kbyte = 1024 Bytes = 8Kbits
- 1Mbyte = 1024 Kbytes = 8*1024 Kbits
- 1Gbyte = 1024 Mbytes =……

Binary Representation

- This representation is based on powers of 2. Any number can be expressed in terms of 0 and 1

Example: $5 = 101_2 = 1 * 2^2 + 0 * 2^1 + 1 * 2^0$

Example: $9 = 1001_2 = 1 * 2^3 + 0 * 2^2 + 0 * 2^1 + 1 * 2^0$

Exercise: Convert any way you can the numbers 19, 38, 58 from decimal to binary. (use calculator or C program)

Hexadecimal Representation

- **This representation is based on powers of 16. Any number can be expressed in terms of:**

  0, 1, 2,…,9,A,B,C,D,E,F **(0,1,2,…,9,10,11,12,13,14,15)**

Example: $256 = 100_{16} = 1 * 16^2 + 0 * 16^1 + 0 * 16^0$

Example: $1002 = 3EA_{16} = 3 * 16^2 + 14 * 16^1 + 10 * 16^0$

Exercise: Convert any way you can the numbers 1492, 3481, 558 from decimal to hex. (use calculator or C program)

Boolean Operations

- **NOT: NOT(101) = 010**
- **AND: AND(10101 ; 01010) = 0**
- **OR   : OR(10101; 01010) = 11111**
- **SHIFT L: SHIFT L(111) = 1110**
- **SHIFT R: SHIFT R(111) = 011**

Why shift is
Important ?
Try SHIFT R(011)

Exercise: (1) Find NOT(AAA)
(2) Find OR(AAA; 555)
(3) Find AND (AEB123; FFF000)

Negative Numbers

- **How do you represent negative numbers ?**

Using 2's Complement Arithmetic

Recipe : Take the complement of the number and add 1.

Example: Consider the number $3 = 011_2$
Then -3 is NOT(011) + 1 = 100 + 1 = 101

Exercise: Consider a 4 bit machine. Derive all possible positive and negative numbers

4-bit Negative Numbers

- Consider a 4 bit machine. Find all positive and negative numbers you can have:

Integer	Sign Magnitude	2's compl ement
+7	0111	0111
+6	0110	0110
+5	0101	0101
+4	0100	0100
+3	0011	0011
+2	0010	0010
+1	0001	0001
0	0000	0000
-1	1001	1111
-2	1010	1110
-3	1011	1101
-4	1100	1100
-5	1101	1011
-6	1110	1010
-7	1111	1001
-8	1000 (-0)	1000

Characters

- The English characters you see on your computer screen are made also by using numbers.
- There is an one-to-one correspondence between all characters and a set of byte numbers world-wide.
- The computers know to interpret these bytes as characters.

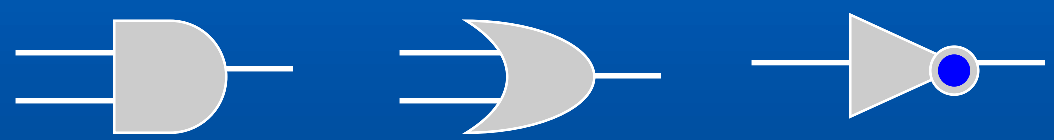
Characters and the ASCII System

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK	(acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL	(bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS	(backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	(device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN	(synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB	(end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN	(cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM	(end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB	(substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC	(escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS	(file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS	(group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US	(unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.asciitable.com

How do the computers do all these ?

- **You may remember from the first year the gates that form an AND, OR, NOT:**



- **Any Digital device can be made out of either ORs and NOTs or ANDs and NOTs.**

- **Truth Tables :**

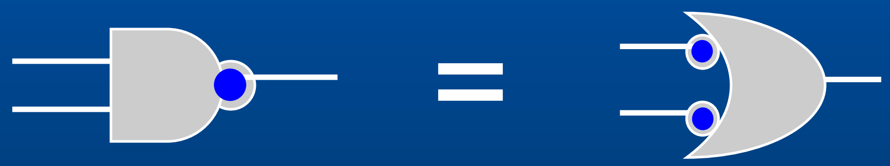
A	B	AND
0	0	0
0	1	0
1	0	0
1	1	1

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

A	NOT
0	1
1	0

DeMorgan's Theorem

- **You can swap ANDs with ORs if at the same time you invert all inputs and outputs :**



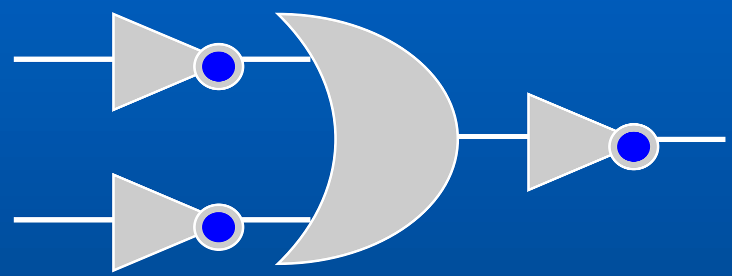
Exercise: Write to truth table for both and prove that this is correct

Costas Foudas, Imperial College,
Rm: 508, x47590

An AND out of NOTs and ORs

- Exercise: Test the claim that you can make <u>any</u> logic device exclusively out of NOTs and ORs by making and AND out of NOTs and ORs:

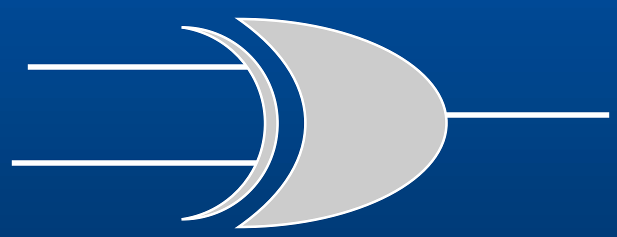
Answer:



- One can test explicitly that this device has an identical truth table as the AND gate.

Exercise: Exclusive OR

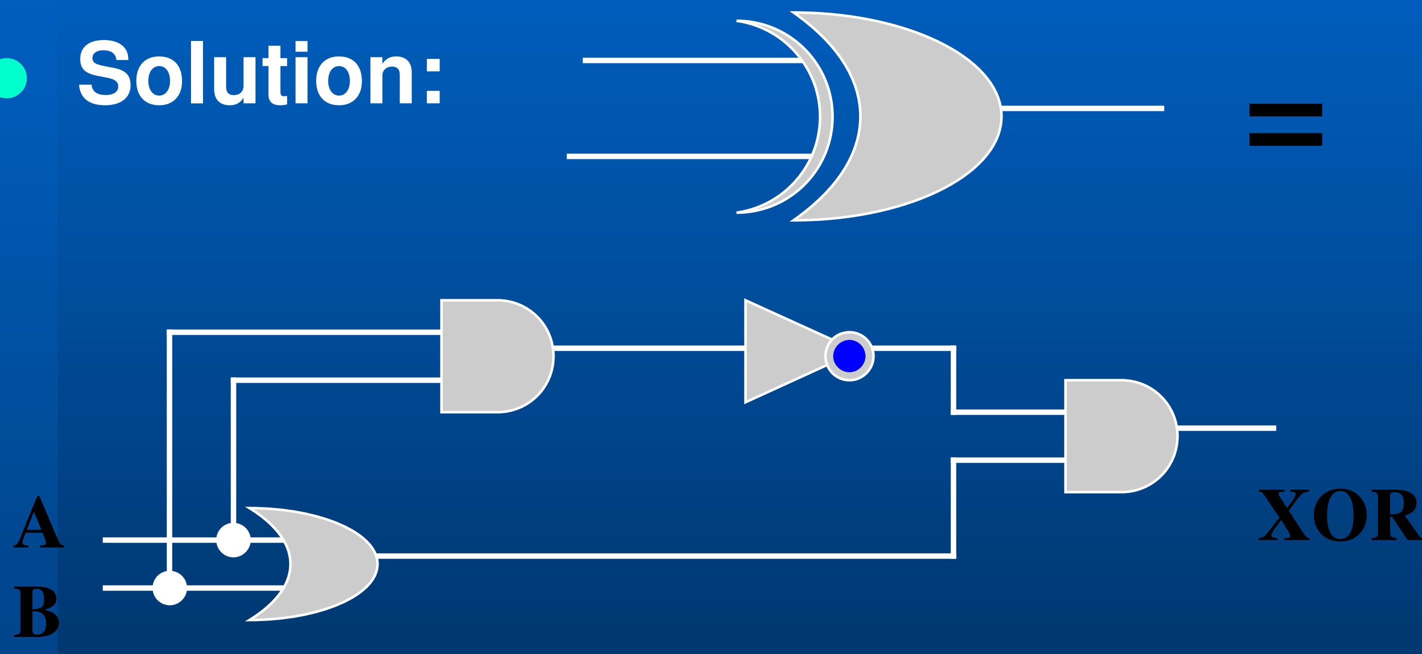
- Construct an exclusive OR gate using OR, AND, AND NOT:



A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

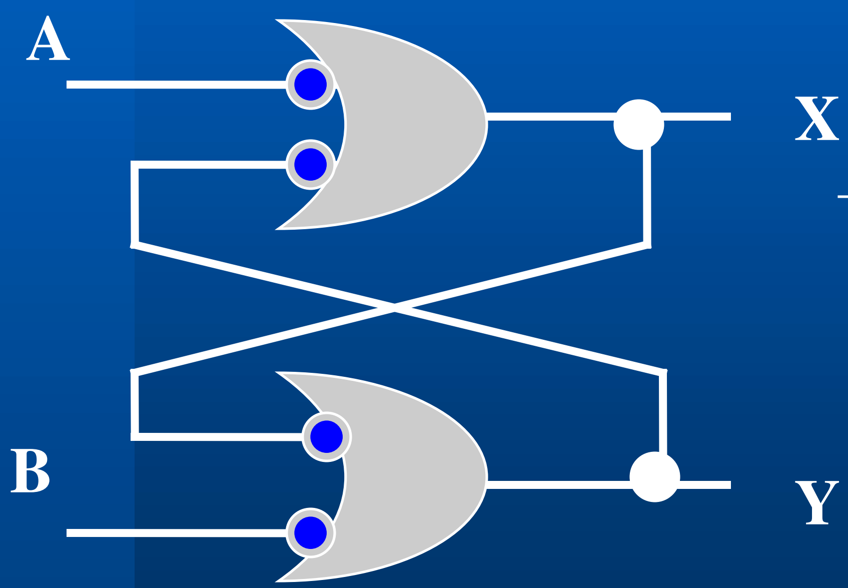
The Exclusive OR

- **Solution:**



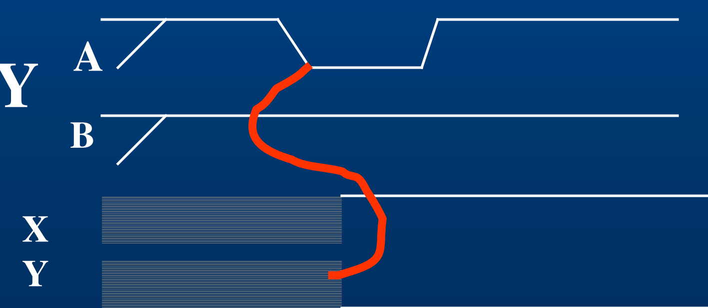
The D-Flip Flop

- Making a DFF using gates



A	B	X	Y
H	H	H	L
H	H	L	H
H	L	L	H
L	H	H	L
L	L	H	H

Undefined



How do Computers Add ?

- **Make a 2 bit adder with a carry_in and a carry out :**

Cin	A	B	SUM	Cout
0	1	0	1	0
0	0	1	1	0
0	0	0	0	0
0	1	1	0	1
1	1	0	0	1
1	0	1	0	1
1	0	0	1	0
1	1	1	1	1

BINARY ADDITION EXAMPLE

(3-BIT MACHINE)

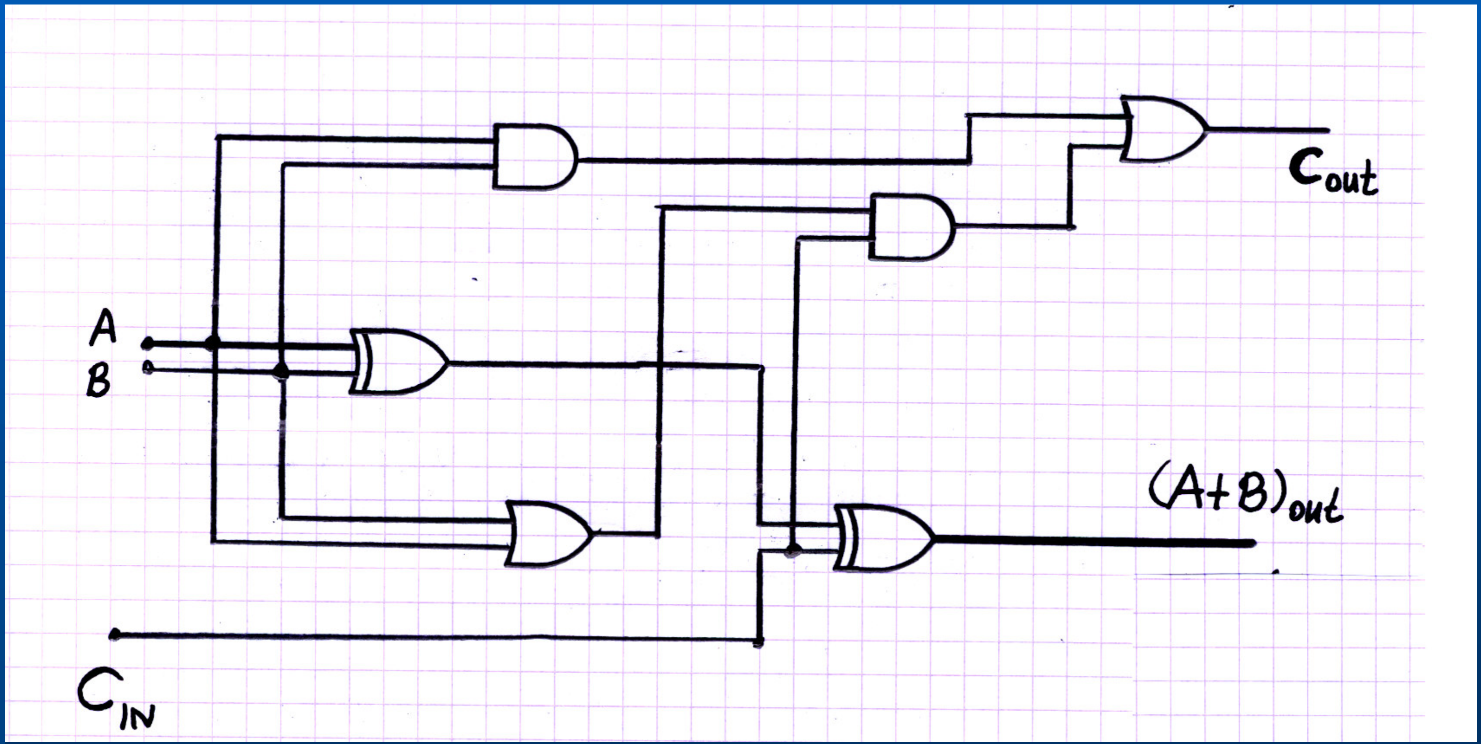
$A = 111_2$, $B = 111_2$, $C = A+B$

$$\begin{array}{rrl} & 111 & (A) \\ + & 111 & (B) \\ \hline & 1\underbrace{110}_{\text{Result}} & (C) \end{array}$$

Carry

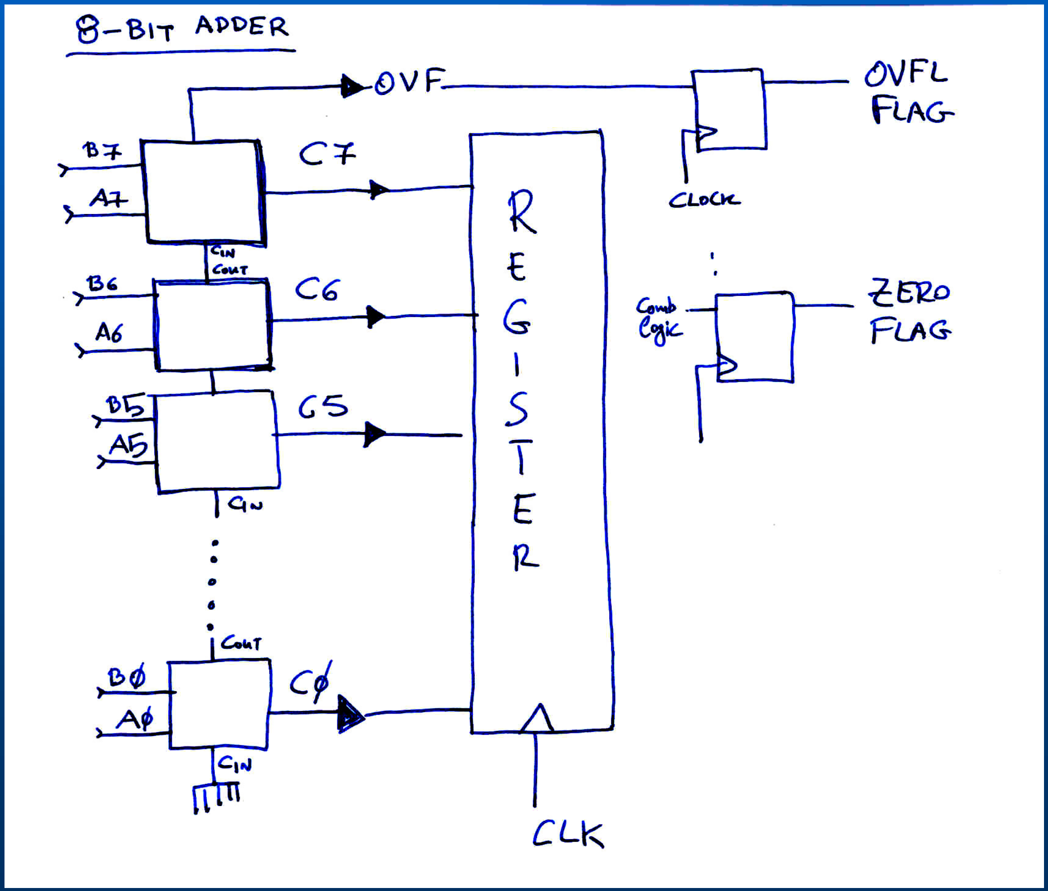
Two Bit Adder with Carry

- **Answer:**

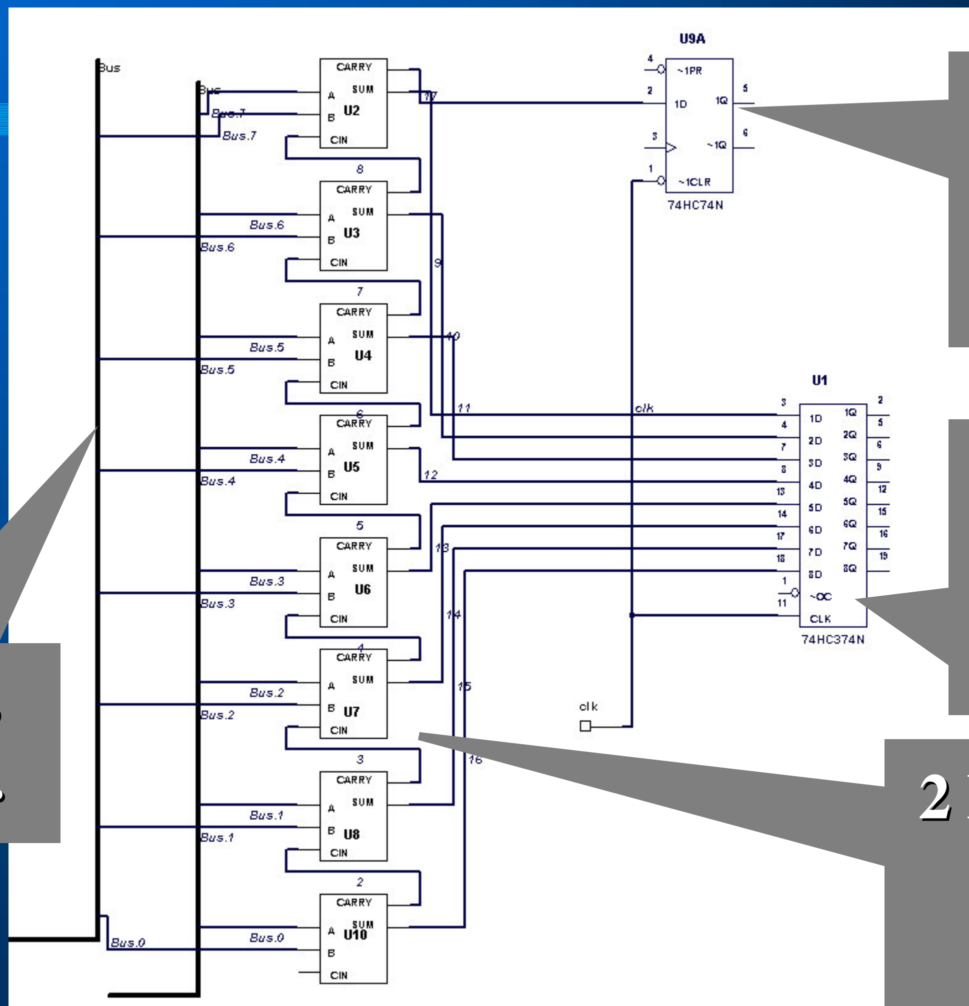


Arithmetic Logic Unit (ALU)

- Center of every computer:



You can actually design one yourself with what you already know



The Arithmetic Overflow Flag is stored here

The result is stored in this register

Eight Bits Wide Bus.

2 Bit adders with carry-in and carry-out