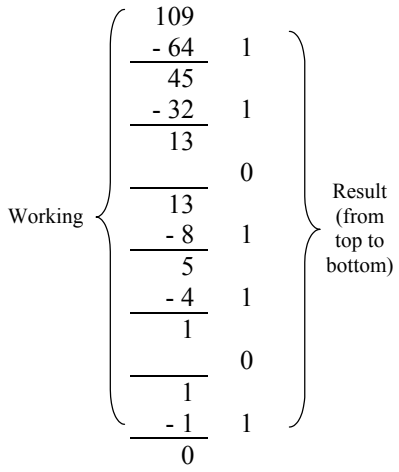


Fig 10.6
Setting out for working using method 1.

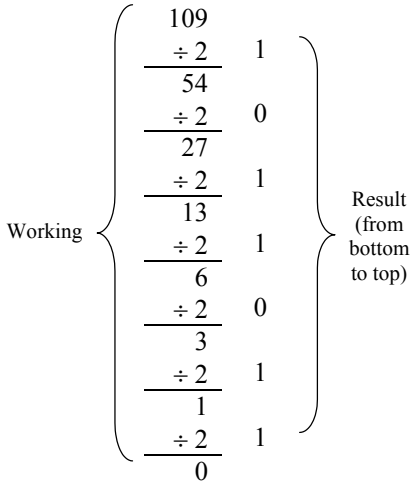


Fig 10.7
Setting out for working using method 2.

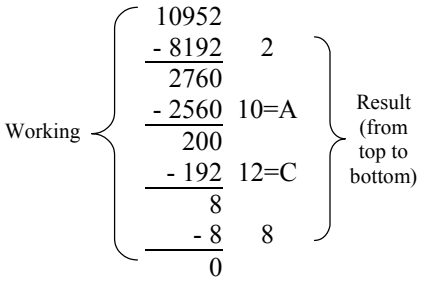


Fig 10.8
Setting out for working using method 1.

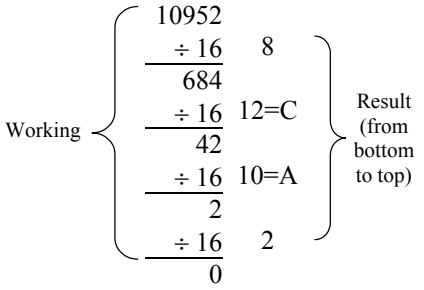


Fig 10.9
Setting out for working using method 2.

Twos complement (Line and table of values)	OHPC10-340a

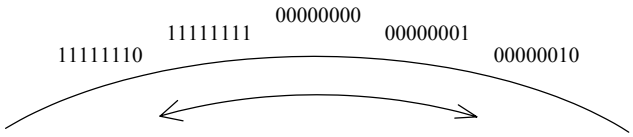


Fig 10.13
Section of an 8 bit circular number line.

Positive	
Decimal	Binary
1	00000001
2	00000010
3	00000011
4	00000100
5	00000101
6	00000110
7	00000111
8	00001000
9	00001001
10	00001010
11	00001011
12	00001100
13	00001101
14	00001110
15	00001111
16	00010000
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	
31	
32	

Negative	
Decimal	Binary
-1	11111111
-2	11111110
-3	11111101
-4	11111100
-5	11111011
-6	11111010
-7	11111001
-8	11111000
-9	11110111
-10	11110110
-11	11110101
-12	11110100
-13	11110011
-14	11110010
-15	11110001
-16	11110000
-17	
-18	
-19	
-20	
-21	
-22	
-23	
-24	
-25	
-26	
-27	
-28	
-29	
-30	
-31	
-32	

Positive	
Decimal	Binary
97	
98	
99	
100	
101	
102	
103	
104	
105	
106	
107	
108	
109	
110	
111	
112	
113	
114	
115	
116	
117	
118	
119	
120	
121	
122	
123	
124	
125	
126	
127	01111111

Negative	
Decimal	Binary
-97	
-98	
-99	
-100	
-101	
-102	
-103	
-104	
-105	
-106	
-107	
-108	
-109	
-110	
-111	
-112	
-113	
-114	
-115	
-116	
-117	
-118	
-119	
-120	
-121	
-122	
-123	
-124	
-125	
-126	
-127	
--128	10000000

Etc…
(Let's cheat and skip a few!)

Phew finished!

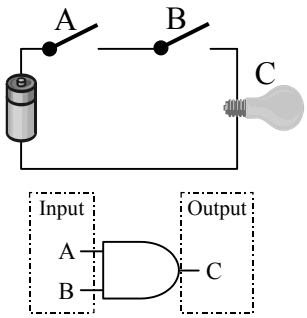
```
BEGIN BinaryDivision
  Get Num1,Num2
  Divisor = Num2
  WHILE Left hand bit of divisor is 0
    Shift Divisor left
  ENDWHILE
  Remainder = Num1
  Quotient = 0
  WHILE Divisor ≥ Num2
    Shift Quotient left
    IF Remainder ≥ Divisor THEN
      Remainder = Remainder – Divisor
      Quotient = Quotient + 1
    ENDIF
    Shift Divisor right
  ENDWHILE
END BinaryDivision
```

Fig 10.29
Algorithm for binary division.

Divisor	Remainder	Quotient

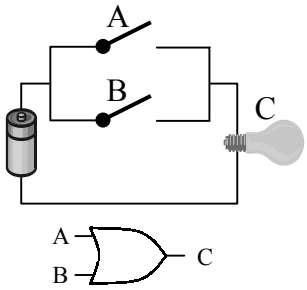
Logic gates ***Fig 10.32 to 10.39***

OHPC10-356a



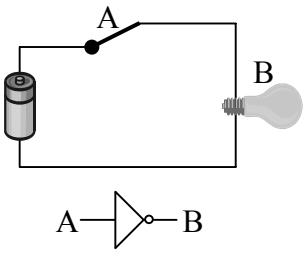
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

AND gate



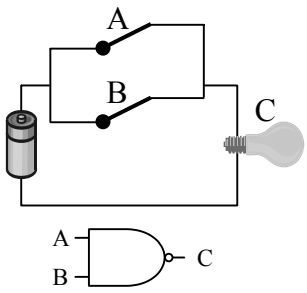
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

OR gate



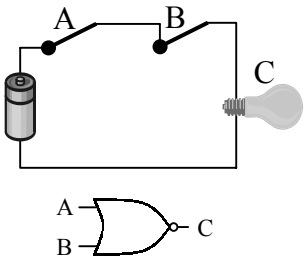
A	B
0	1
1	0

NOT gate



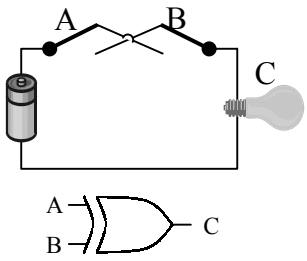
A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

NAND gate



A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

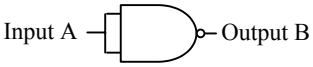
NOR gate



A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

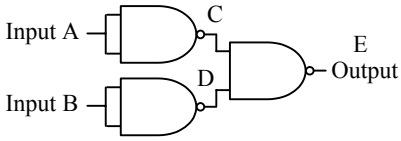
XOR gate

Logic gates produced with only NAND gates	OHPC10-360a



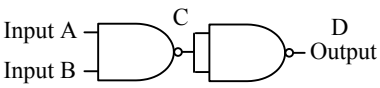
NOT gate

A	B
0	
1	



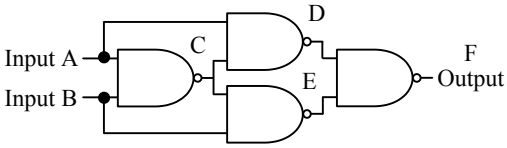
OR gate

A	B	C	D	E
0	0			
0	1			
1	0			
1	1			



AND gate

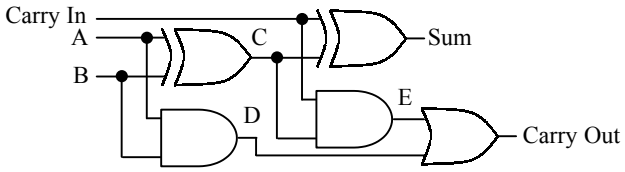
A	B	C	D
0	0		
0	1		
1	0		
1	1		



XOR gate

A	B	C	D	E	F
0	0				
0	1				
1	0				
1	1				

Full adder and truth table	**OHPC10-361a**



A full adder

Carry In	A	B	C	D	E	Carry Out	Sum
0	0	0					
0	0	1					
0	1	0					
0	1	1					
1	0	0					
1	0	1					
1	1	0					
1	1	1					

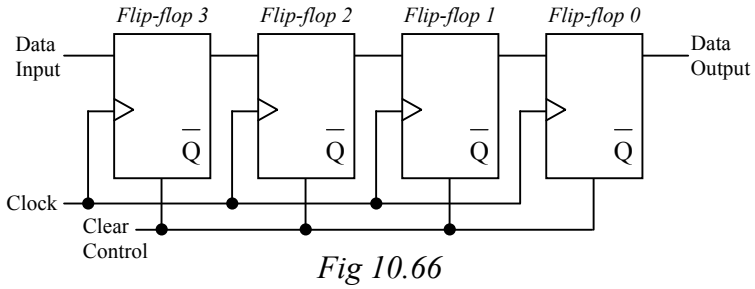


Fig 10.66
Serial in – serial out shift register

Clock	Data Input	Flip-flop			
		3	2	1	0

Clock	Data Input	Flip-flop			
		3	2	1	0