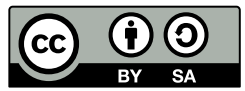
Number systems. Binary arithmetic

Saulius Gražulis

Vilnius, 2020

Vilnius University, Faculty of Mathematics and Informatics
Institute of Informatics



Binary arithmetic

Representable digits in our hardware: **0** and **1**

System:	dec	bin	oct	hex
Base:	10	2	8	16

Binary arithmetic

Representable digits in our hardware: **0** and **1**

System: / Base:	dec 10	bin 2	oct 8	hex 16
	0			

Binary arithmetic

Representable digits in our hardware: **0** and **1**

System:	dec	bin	oct	hex
Base:	10	2	8	16
	0			
	1			

Binary arithmetic

Representable digits in our hardware: **0** and **1**

System:	dec	bin	oct	hex
Base:	10	2	8	16
	0			
	1			
	2			

Binary arithmetic

Representable digits in our hardware: **0** and **1**

System:	dec	bin	oct	hex
Base:	10	2	8	16
	0			
	1			
	2			
	3			

Binary arithmetic

Representable digits in our hardware: **0** and **1**

System: Base:	dec 10	bin 2	oct 8	hex 16
	0			
	1			
	2			
	3			
	4			
	5			
	6			
	7			
	8			
	9			

Binary arithmetic

Representable digits in our hardware: **0** and **1**

System: Base:	dec 10	bin 2	oct 8	hex 16
	0			
	1			
	2			
	3			
	4			
	5			
	6			
	7			
	8			
	9			
	10			

Binary arithmetic

Representable digits in our hardware: **0** and **1**

System: Base:	dec 10	bin 2	oct 8	hex 16
	0			
	1			
	2			
	3			
	4			
	5			
	6			
	7			
	8			
	9			
	10			
	11			

Binary arithmetic

Representable digits in our hardware: **0** and **1**

System: Base:	dec 10	bin 2	oct 8	hex 16
	0			
	1			
	2			
	3			
	4			
	5			
	6			
	7			
	8			
	9			
	10			
	11			
	12			
	13			
	14			
	15			
	16			
	17			

Binary arithmetic

Representable digits in our hardware: **0** and **1**

System: Base:	dec 10	bin 2	oct 8	hex 16
	0	0		
	1			
	2			
	3			
	4			
	5			
	6			
	7			
	8			
	9			
	10			
	11			
	12			
	13			
	14			
	15			
	16			
	17			

Binary arithmetic

Representable digits in our hardware: **0** and **1**

System:	dec	bin	oct	hex
Base:	10	2	8	16
	0	0		
	1	1		
	2			
	3			
	4			
	5			
	6			
	7			
	8			
	9			
	10			
	11			
	12			
	13			
	14			
	15			
	16			
	17			

Binary arithmetic

Representable digits in our hardware: **0** and **1**

System:	dec	bin	oct	hex
Base:	10	2	8	16
	0	0		
	1	1		
	2	**10**		
	3			
	4			
	5			
	6			
	7			
	8			
	9			
	10			
	11			
	12			
	13			
	14			
	15			
	16			
	17			

Binary arithmetic

Representable digits in our hardware: **0** and **1**

System:	dec	bin	oct	hex
Base:	10	2	8	16
	0	0		
	1	1		
	2	10		
	3	11		
	4			
	5			
	6			
	7			
	8			
	9			
	10			
	11			
	12			
	13			
	14			
	15			
	16			
	17			

Binary arithmetic

Representable digits in our hardware: **0** and **1**

System: Base:	dec 10	bin 2	oct 8	hex 16
	0	0		
	1	1		
	2	10		
	3	11		
	4	**100**		
	5			
	6			
	7			
	8			
	9			
	10			
	11			
	12			
	13			
	14			
	15			
	16			
	17			

Binary arithmetic

Representable digits in our hardware: **0** and **1**

System: Base:	dec 10	bin 2	oct 8	hex 16
	0	0		
	1	1		
	2	10		
	3	11		
	4	100		
	5	101		
	6			
	7			
	8			
	9			
	10			
	11			
	12			
	13			
	14			
	15			
	16			
	17			

Binary arithmetic

Representable digits in our hardware: **0** and **1**

System:	dec	bin	oct	hex
Base:	10	2	8	16
	0	0		
	1	1		
	2	10		
	3	11		
	4	100		
	5	101		
	6	110		
	7			
	8			
	9			
	10			
	11			
	12			
	13			
	14			
	15			
	16			
	17			

Binary arithmetic

Representable digits in our hardware: **0** and **1**

System: Base:	dec 10	bin 2	oct 8	hex 16
	0	0		
	1	1		
	2	10		
	3	11		
	4	100		
	5	101		
	6	110		
	7	111		
	8			
	9			
	10			
	11			
	12			
	13			
	14			
	15			
	16			
	17			

Binary arithmetic

Representable digits in our hardware: **0** and **1**

System: Base:	dec 10	bin 2	oct 8	hex 16
	0	0		
	1	1		
	2	10		
	3	11		
	4	100		
	5	101		
	6	110		
	7	111		
	8	**1000**		
	9			
	10			
	11			
	12			
	13			
	14			
	15			
	16			
	17			

Binary arithmetic

Representable digits in our hardware: **0** and **1**

System: Base:	dec 10	bin 2	oct 8	hex 16
	0	0		
	1	1		
	2	10		
	3	11		
	4	100		
	5	101		
	6	110		
	7	111		
	8	1000		
	9	1001		
	10			
	11			
	12			
	13			
	14			
	15			
	16			
	17			

Binary arithmetic

Representable digits in our hardware: **0** and **1**

System: Base:	dec 10	bin 2	oct 8	hex 16
	0	0		
	1	1		
	2	10		
	3	11		
	4	100		
	5	101		
	6	110		
	7	111		
	8	1000		
	9	1001		
	10	1010		
	11	1011		
	12	1100		
	13	1101		
	14	1110		
	15	1111		
	16	10000		
	17	10001		

Binary arithmetic

Representable digits in our hardware: **0** and **1**

System: Base:	dec 10	bin 2	oct 8	hex 16
	0	0	0	
	1	1	1	
	2	10	2	
	3	11	3	
	4	100	4	
	5	101	5	
	6	110	6	
	7	111	7	
	8	1000		
	9	1001		
	10	1010		
	11	1011		
	12	1100		
	13	1101		
	14	1110		
	15	1111		
	16	10000		
	17	10001		

Binary arithmetic

Representable digits in our hardware: **0** and **1**

System:	dec	bin	oct	hex
Base:	10	2	8	16
	0	0	0	
	1	1	1	
	2	10	2	
	3	11	3	
	4	100	4	
	5	101	5	
	6	110	6	
	7	111	7	
	8	1000	10	
	9	1001		
	10	1010		
	11	1011		
	12	1100		
	13	1101		
	14	1110		
	15	1111		
	16	10000		
	17	10001		

Binary arithmetic

Representable digits in our hardware: **0** and **1**

System: Base:	dec 10	bin 2	oct 8	hex 16
	0	0	0	
	1	1	1	
	2	10	2	
	3	11	3	
	4	100	4	
	5	101	5	
	6	110	6	
	7	111	7	
	8	1000	10	
	9	1001	11	
	10	1010	12	
	11	1011	13	
	12	1100	14	
	13	1101	15	
	14	1110	16	
	15	1111	17	
	16	10000	20	
	17	10001	21	

Binary arithmetic

Representable digits in our hardware: **0** and **1**

System: Base:	dec 10	bin 2	oct 8	hex 16
	0	0	0	0
	1	1	1	1
	2	10	2	2
	3	11	3	3
	4	100	4	4
	5	101	5	5
	6	110	6	6
	7	111	7	7
	8	1000	10	8
	9	1001	11	9
	10	1010	12	
	11	1011	13	
	12	1100	14	
	13	1101	15	
	14	1110	16	
	15	1111	17	
	16	10000	20	
	17	10001	21	

Binary arithmetic

Representable digits in our hardware: **0** and **1**

System: Base:	dec 10	bin 2	oct 8	hex 16
	0	0	0	0
	1	1	1	1
	2	10	2	2
	3	11	3	3
	4	100	4	4
	5	101	5	5
	6	110	6	6
	7	111	7	7
	8	1000	10	8
	9	1001	11	9
	10	1010	12	A
	11	1011	13	
	12	1100	14	
	13	1101	15	
	14	1110	16	
	15	1111	17	
	16	10000	20	
	17	10001	21	

Binary arithmetic

Representable digits in our hardware: **0** and **1**

System: Base:	dec 10	bin 2	oct 8	hex 16
	0	0	0	0
	1	1	1	1
	2	10	2	2
	3	11	3	3
	4	100	4	4
	5	101	5	5
	6	110	6	6
	7	111	7	7
	8	1000	10	8
	9	1001	11	9
	10	1010	12	A
	11	1011	13	B
	12	1100	14	
	13	1101	15	
	14	1110	16	
	15	1111	17	
	16	10000	20	
	17	10001	21	

Binary arithmetic

Representable digits in our hardware: **0** and **1**

System:	dec	bin	oct	hex
Base:	10	2	8	16
	0	0	0	0
	1	1	1	1
	2	10	2	2
	3	11	3	3
	4	100	4	4
	5	101	5	5
	6	110	6	6
	7	111	7	7
	8	1000	10	8
	9	1001	11	9
	10	1010	12	A
	11	1011	13	B
	12	1100	14	C
	13	1101	15	
	14	1110	16	
	15	1111	17	
	16	10000	20	
	17	10001	21	

Binary arithmetic

Representable digits in our hardware: **0** and **1**

System: Base:	dec 10	bin 2	oct 8	hex 16
	0	0	0	0
	1	1	1	1
	2	10	2	2
	3	11	3	3
	4	100	4	4
	5	101	5	5
	6	110	6	6
	7	111	7	7
	8	1000	10	8
	9	1001	11	9
	10	1010	12	A
	11	1011	13	B
	12	1100	14	C
	13	1101	15	D
	14	1110	16	
	15	1111	17	
	16	10000	20	
	17	10001	21	

Binary arithmetic

Representable digits in our hardware: **0** and **1**

System: Base:	dec 10	bin 2	oct 8	hex 16
	0	0	0	0
	1	1	1	1
	2	10	2	2
	3	11	3	3
	4	100	4	4
	5	101	5	5
	6	110	6	6
	7	111	7	7
	8	1000	10	8
	9	1001	11	9
	10	1010	12	A
	11	1011	13	B
	12	1100	14	C
	13	1101	15	D
	14	1110	16	E
	15	1111	17	
	16	10000	20	
	17	10001	21	

Binary arithmetic

Representable digits in our hardware: **0** and **1**

System: Base:	dec 10	bin 2	oct 8	hex 16
	0	0	0	0
	1	1	1	1
	2	10	2	2
	3	11	3	3
	4	100	4	4
	5	101	5	5
	6	110	6	6
	7	111	7	7
	8	1000	10	8
	9	1001	11	9
	10	1010	12	A
	11	1011	13	B
	12	1100	14	C
	13	1101	15	D
	14	1110	16	E
	15	1111	17	F
	16	10000	20	
	17	10001	21	

Binary arithmetic

Representable digits in our hardware: **0** and **1**

System: Base:	dec 10	bin 2	oct 8	hex 16
	0	0	0	0
	1	1	1	1
	2	10	2	2
	3	11	3	3
	4	100	4	4
	5	101	5	5
	6	110	6	6
	7	111	7	7
	8	1000	10	8
	9	1001	11	9
	10	1010	12	A
	11	1011	13	B
	12	1100	14	C
	13	1101	15	D
	14	1110	16	E
	15	1111	17	F
	16	10000	20	10
	17	10001	21	

Binary arithmetic

Representable digits in our hardware: **0** and **1**

System:	dec	bin	oct	hex
Base:	10	2	8	16
	0	0	0	0
	1	1	1	1
	2	10	2	2
	3	11	3	3
	4	100	4	4
	5	101	5	5
	6	110	6	6
	7	111	7	7
	8	1000	10	8
	9	1001	11	9
	10	1010	12	A
	11	1011	13	B
	12	1100	14	C
	13	1101	15	D
	14	1110	16	E
	15	1111	17	F
	16	10000	20	10
	17	10001	21	11

Binary arithmetic

Representable digits in our hardware: **0** and **1**

System: Base:	dec 10	bin 2	oct 8	hex 16
	0	0	0	0
	1	1	1	1
	2	**10**	2	2
	3	11	3	3
	4	**100**	4	4
	5	101	5	5
	6	110	6	6
	7	111	7	7
	8	**1000**	**10**	8
	9	1001	11	9
	10	1010	12	A
	11	1011	13	B
	12	1100	14	C
	13	1101	15	D
	14	1110	16	E
	15	1111	17	F
	16	**10000**	20	**10**
	17	10001	21	11

Binary arithmetic

Representable digits in our hardware: **0** and **1**

System: Base:	dec 10	bin 2	oct 8	hex 16
	0	0	0	0
	1	1	1	1
	2	10	2	2
	3	11	3	3
	4	100	4	4
	5	101	5	5
	6	110	6	6
	7	111	7	7
	8	**1000**	**10**	8
	9	1001	11	9
	10	1010	12	A
	11	1011	13	B
	12	1100	14	C
	13	1101	15	D
	14	1110	16	E
	15	1111	17	F
	16	**10000**	**20**	**10**
	17	10001	21	11

Expressing numbers

In base b, an m-digit number is represented as a polynomial:

$$N = a_{m-1}b^{m-1} + a_{m-2}b^{m-2} + \cdots + a_2 b^2 + a_1 b + a_0$$

Example:

$$\begin{aligned} 110101_2 &= 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 \\ &= 32_{10} + 16_{10} + 4_{10} + 1_{10} \\ &= 53_{10} \end{aligned}$$

Another example:

$$16_{16} = 1 \cdot 16^1 + 6 \cdot 16^0 = 16_{10} + 6_{10} = 22_{10}$$

Expressing fractions

In base b, a fraction $f = 0.a_1 a_2 \ldots a_m$ is represented as a polynomial:

$$f = a_1 b^{-1} + a_2 b^{-2} + \cdots + a_{m-1} b^{-m+1} + a_m b^{-m}$$

Example:

$$\begin{aligned} 0.0110101_2 &= 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} + 0 \cdot 2^{-4} + \\ &\quad + 1 \cdot 2^{-5} + 0 \cdot 2^{-6} + 1 \cdot 2^{-7} \\ &= \frac{1}{4_{10}} + \frac{1}{8_{10}} + \frac{1}{32_{10}} + \frac{1}{128_{10}} \\ &= 0.4140625_{10} \end{aligned}$$

Converting into other number systems

Division by the system's base gives reminder that is the last digit of the number:

$$\begin{aligned} N &= a_{m-1}b^{m-1} + a_{m-2}b^{m-2} + \cdots + a_2 b^2 + a_1 b + a_0 \\ &= b(a_{m-1}b^{m-2} + a_{m-2}b^{m-3} + \cdots + a_2 b + a_1) + a_0 \end{aligned}$$

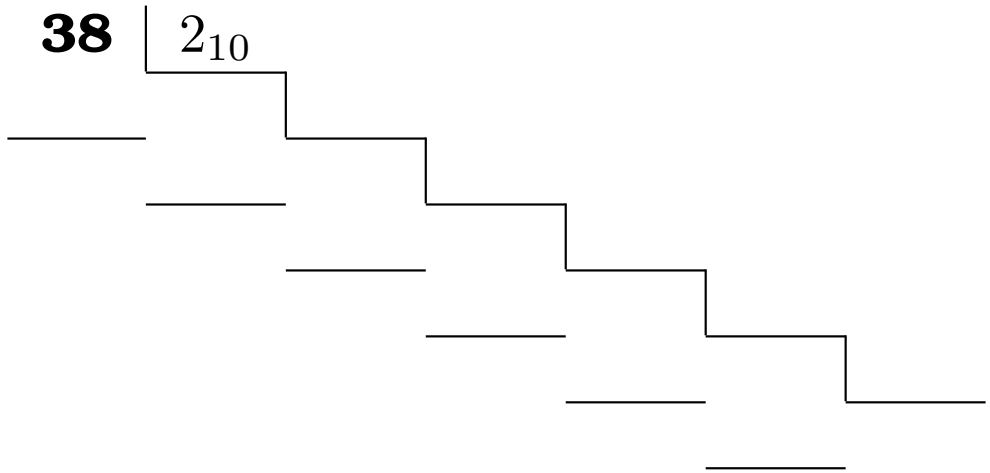
Converting into other number systems

Division by the system's base gives reminder that is the last digit of the number:

$$100110_2/10_2 = 10011.0_2 \quad (\text{quotient} = 10011_2, \text{reminder} = 0)$$

$$10011_2/10_2 = 1001.1_2 \quad (\text{quotient} = 1001_2, \text{reminder} = 1)$$

Thus, repeated division by a *base* gives us digits in that base as *reminders*:



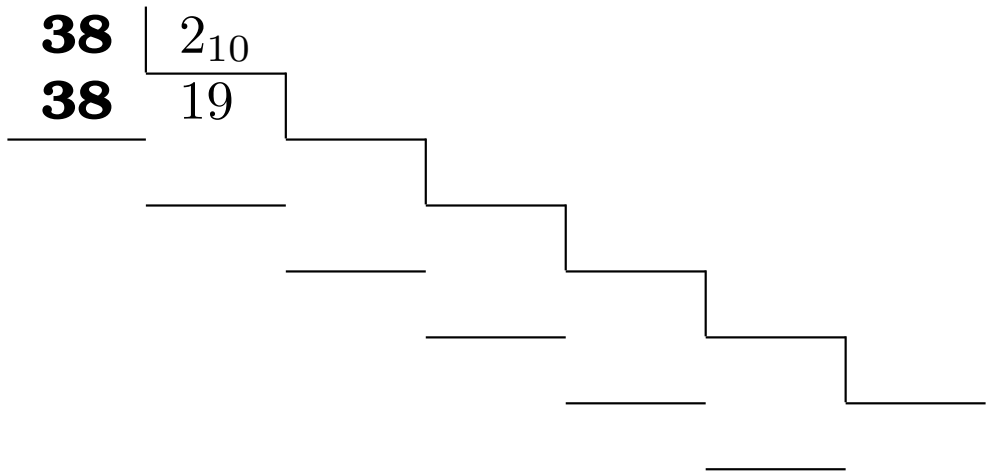
Converting into other number systems

Division by the system's base gives reminder that is the last digit of the number:

$$100110_2/10_2 = 10011.0_2 \quad (\text{quotient} = 10011_2, \text{reminder} = 0)$$

$$10011_2/10_2 = 1001.1_2 \quad (\text{quotient} = 1001_2, \text{reminder} = 1)$$

Thus, repeated division by a *base* gives us digits in that base as *reminders*:



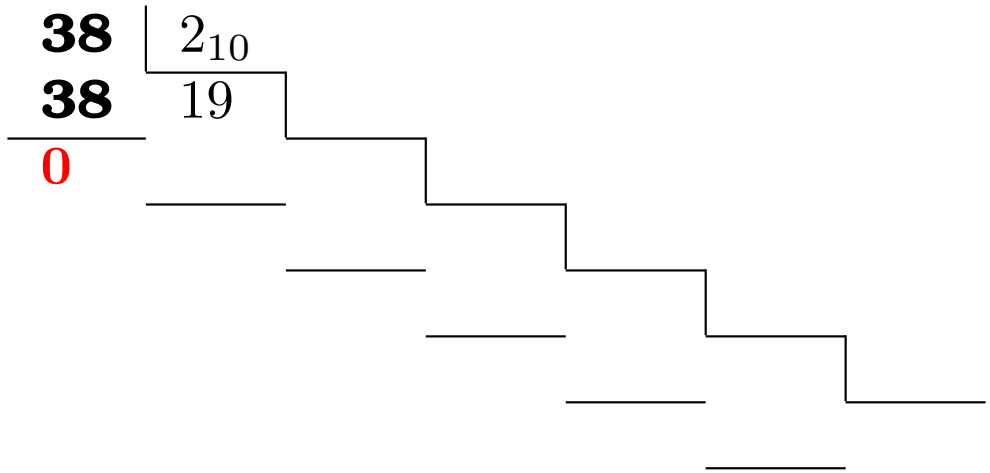
Converting into other number systems

Division by the system's base gives reminder that is the last digit of the number:

$$100110_2/10_2 = 10011.0_2 \quad (\text{quotient} = 10011_2, \text{reminder} = 0)$$

$$10011_2/10_2 = 1001.1_2 \quad (\text{quotient} = 1001_2, \text{reminder} = 1)$$

Thus, repeated division by a *base* gives us digits in that base as *reminders*:



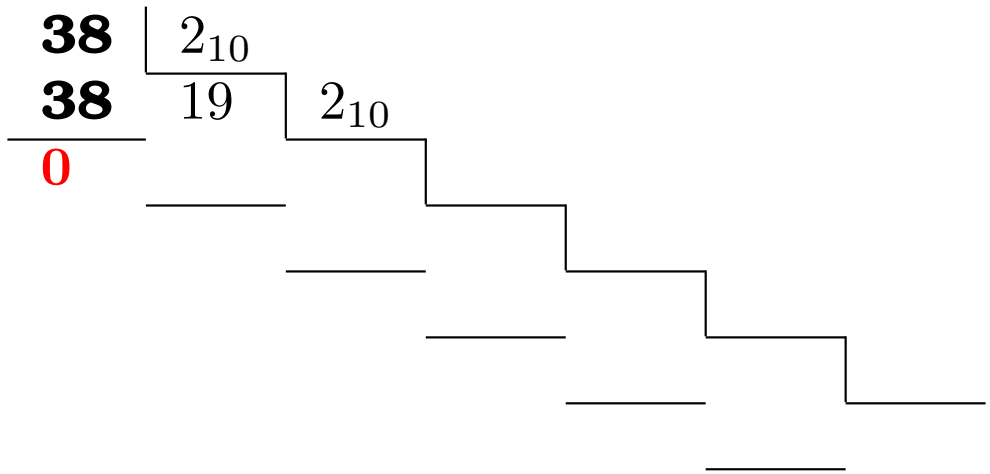
Converting into other number systems

Division by the system's base gives reminder that is the last digit of the number:

$$100110_2/10_2 = 10011.0_2 \quad (\text{quotient} = 10011_2, \text{reminder} = 0)$$

$$10011_2/10_2 = 1001.1_2 \quad (\text{quotient} = 1001_2, \text{reminder} = 1)$$

Thus, repeated division by a *base* gives us digits in that base as *reminders*:



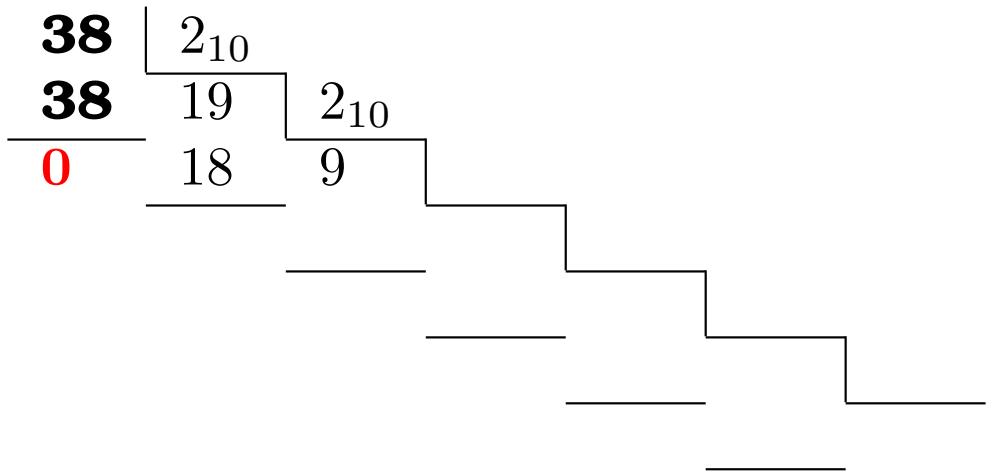
Converting into other number systems

Division by the system's base gives reminder that is the last digit of the number:

$$100110_2/10_2 = 10011.0_2 \quad (\text{quotient} = 10011_2, \text{reminder} = 0)$$

$$10011_2/10_2 = 1001.1_2 \quad (\text{quotient} = 1001_2, \text{reminder} = 1)$$

Thus, repeated division by a *base* gives us digits in that base as *reminders*:



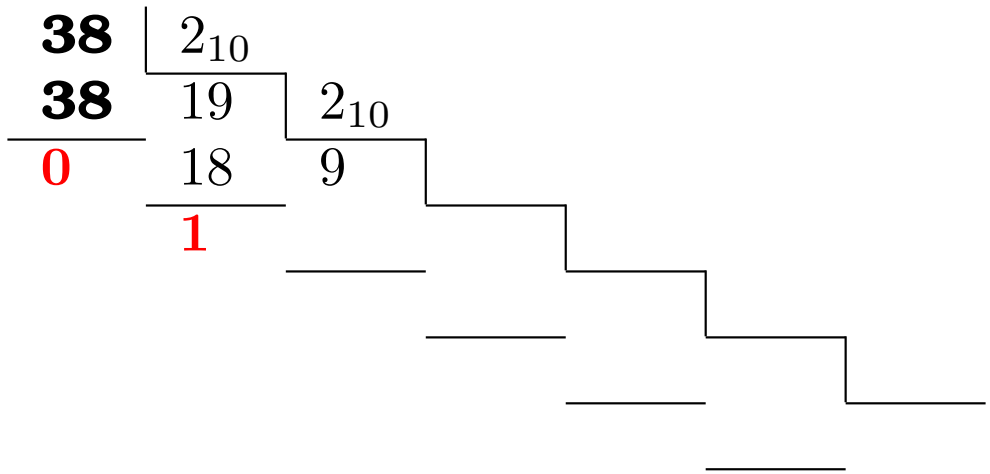
Converting into other number systems

Division by the system's base gives reminder that is the last digit of the number:

$$100110_2/10_2 = 10011.0_2 \quad (\text{quotient} = 10011_2, \text{reminder} = 0)$$

$$10011_2/10_2 = 1001.1_2 \quad (\text{quotient} = 1001_2, \text{reminder} = 1)$$

Thus, repeated division by a *base* gives us digits in that base as *reminders*:



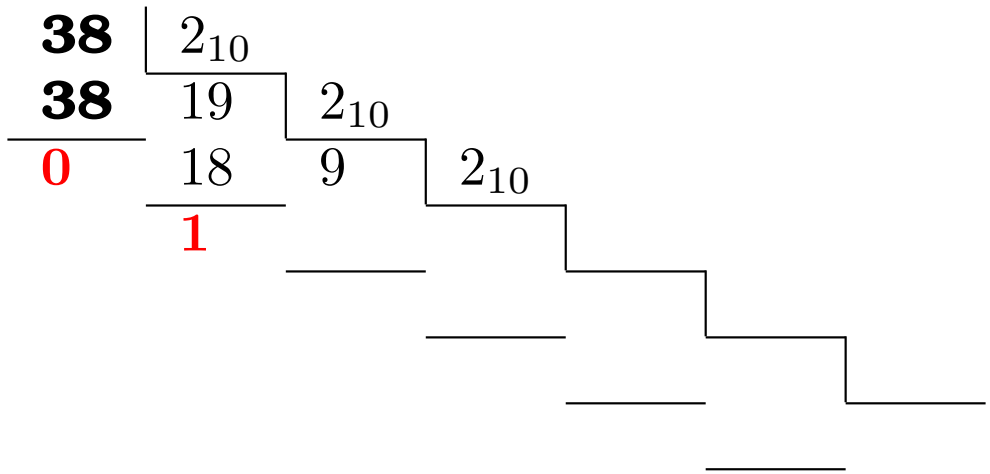
Converting into other number systems

Division by the system's base gives reminder that is the last digit of the number:

$$100110_2/10_2 = 10011.0_2 \quad (\text{quotient} = 10011_2, \text{reminder} = 0)$$

$$10011_2/10_2 = 1001.1_2 \quad (\text{quotient} = 1001_2, \text{reminder} = 1)$$

Thus, repeated division by a *base* gives us digits in that base as *reminders*:



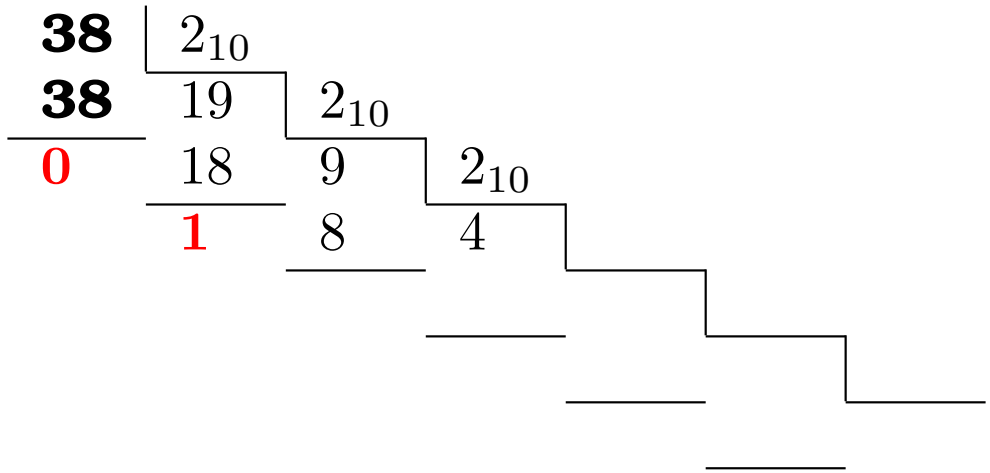
Converting into other number systems

Division by the system's base gives reminder that is the last digit of the number:

$$100110_2/10_2 = 10011.0_2 \quad (\text{quotient} = 10011_2, \text{reminder} = 0)$$

$$10011_2/10_2 = 1001.1_2 \quad (\text{quotient} = 1001_2, \text{reminder} = 1)$$

Thus, repeated division by a *base* gives us digits in that base as *reminders*:



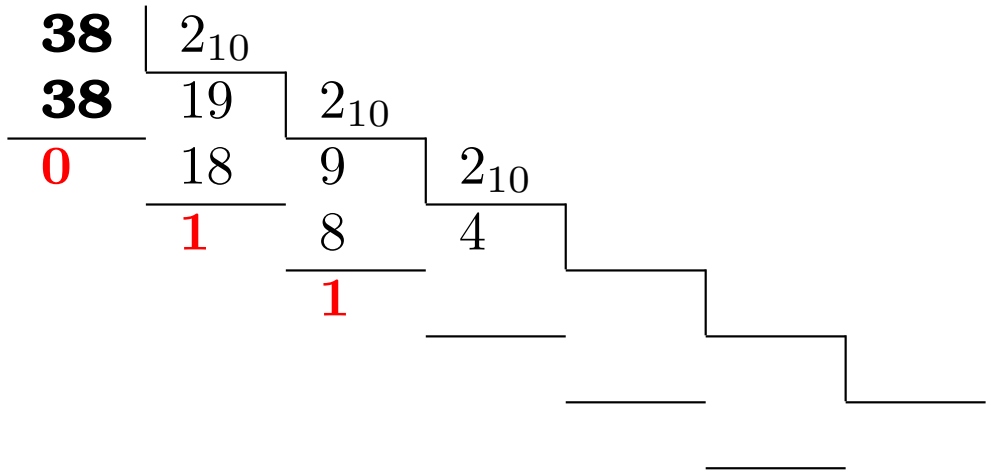
Converting into other number systems

Division by the system's base gives reminder that is the last digit of the number:

$$100110_2/10_2 = 10011.0_2 \quad (\text{quotient} = 10011_2, \text{reminder} = 0)$$

$$10011_2/10_2 = 1001.1_2 \quad (\text{quotient} = 1001_2, \text{reminder} = 1)$$

Thus, repeated division by a *base* gives us digits in that base as *reminders*:



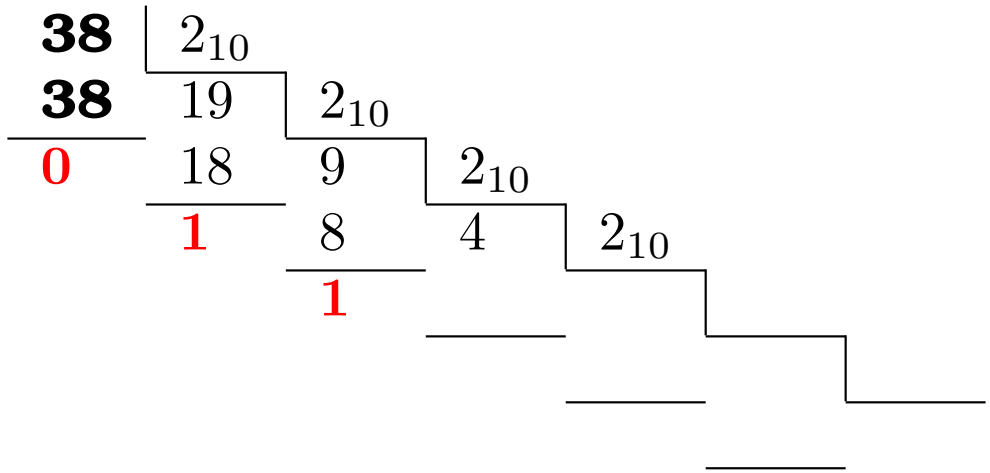
Converting into other number systems

Division by the system's base gives reminder that is the last digit of the number:

$$100110_2/10_2 = 10011.0_2 \quad (\text{quotient} = 10011_2, \text{reminder} = 0)$$

$$10011_2/10_2 = 1001.1_2 \quad (\text{quotient} = 1001_2, \text{reminder} = 1)$$

Thus, repeated division by a *base* gives us digits in that base as *reminders*:



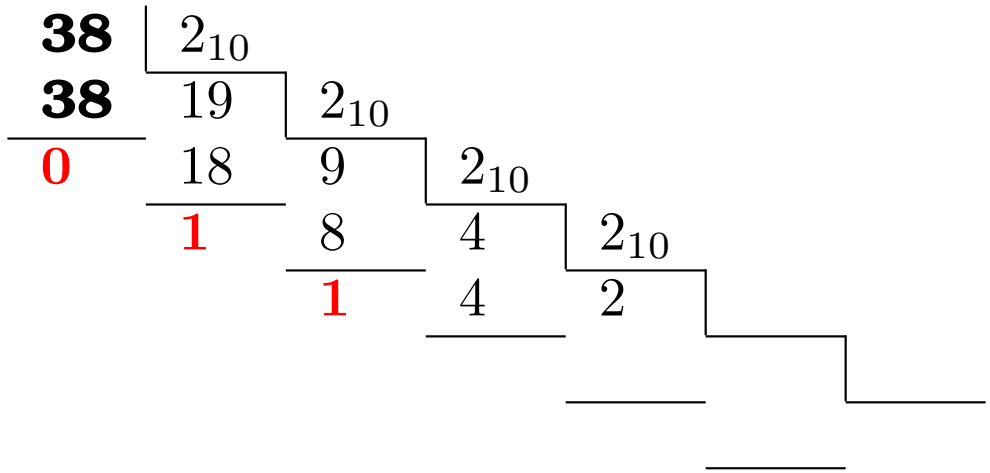
Converting into other number systems

Division by the system's base gives reminder that is the last digit of the number:

$$100110_2/10_2 = 10011.0_2 \quad (\text{quotient} = 10011_2, \text{reminder} = 0)$$

$$10011_2/10_2 = 1001.1_2 \quad (\text{quotient} = 1001_2, \text{reminder} = 1)$$

Thus, repeated division by a *base* gives us digits in that base as *reminders*:



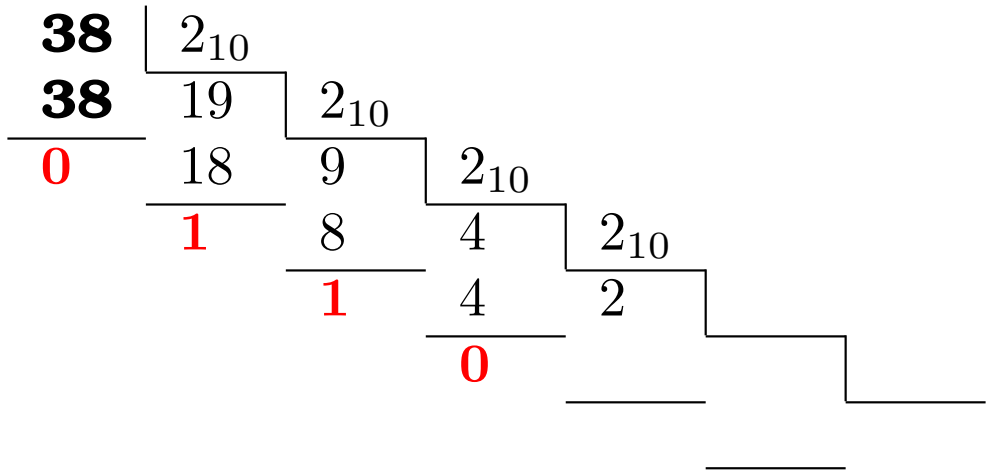
Converting into other number systems

Division by the system's base gives reminder that is the last digit of the number:

$$100110_2/10_2 = 10011.0_2 \quad (\text{quotient} = 10011_2, \text{reminder} = 0)$$

$$10011_2/10_2 = 1001.1_2 \quad (\text{quotient} = 1001_2, \text{reminder} = 1)$$

Thus, repeated division by a *base* gives us digits in that base as *reminders*:



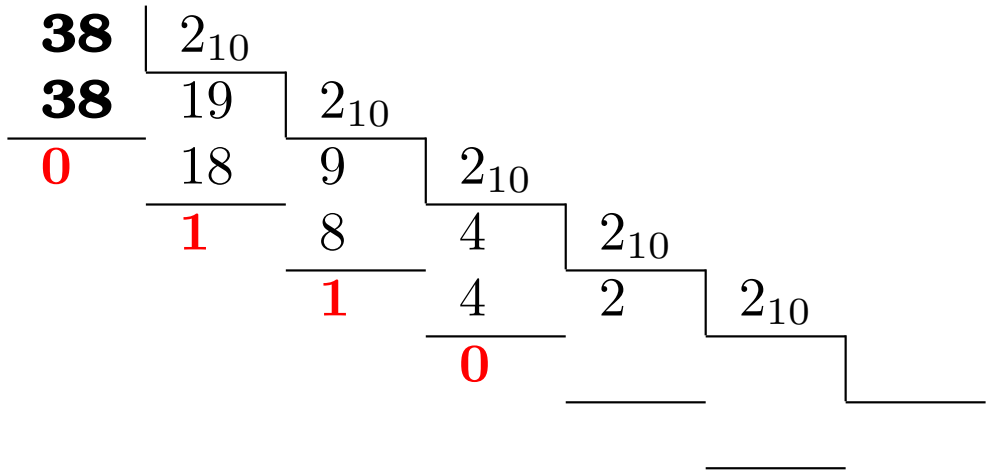
Converting into other number systems

Division by the system's base gives reminder that is the last digit of the number:

$$100110_2/10_2 = 10011.0_2 \quad (\text{quotient} = 10011_2, \text{reminder} = 0)$$

$$10011_2/10_2 = 1001.1_2 \quad (\text{quotient} = 1001_2, \text{reminder} = 1)$$

Thus, repeated division by a *base* gives us digits in that base as *reminders*:



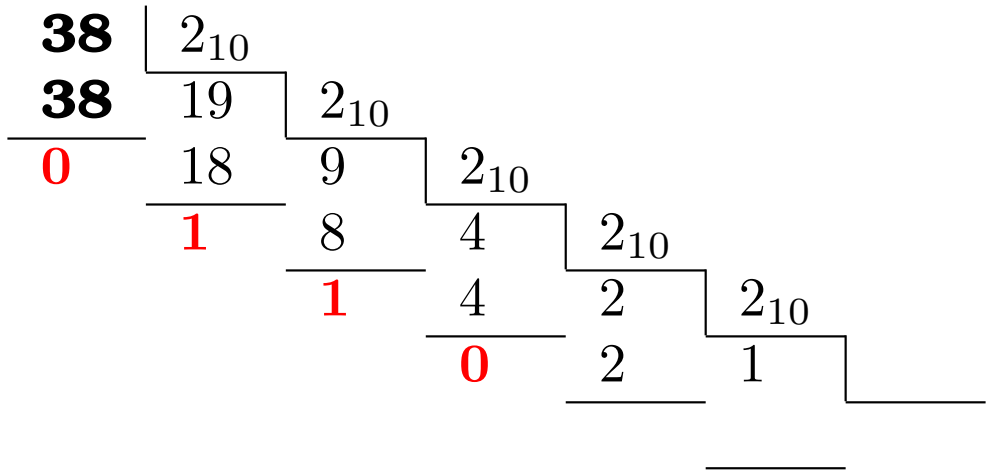
Converting into other number systems

Division by the system's base gives reminder that is the last digit of the number:

$$100110_2/10_2 = 10011.0_2 \quad (\text{quotient} = 10011_2, \text{reminder} = 0)$$

$$10011_2/10_2 = 1001.1_2 \quad (\text{quotient} = 1001_2, \text{reminder} = 1)$$

Thus, repeated division by a *base* gives us digits in that base as *reminders*:



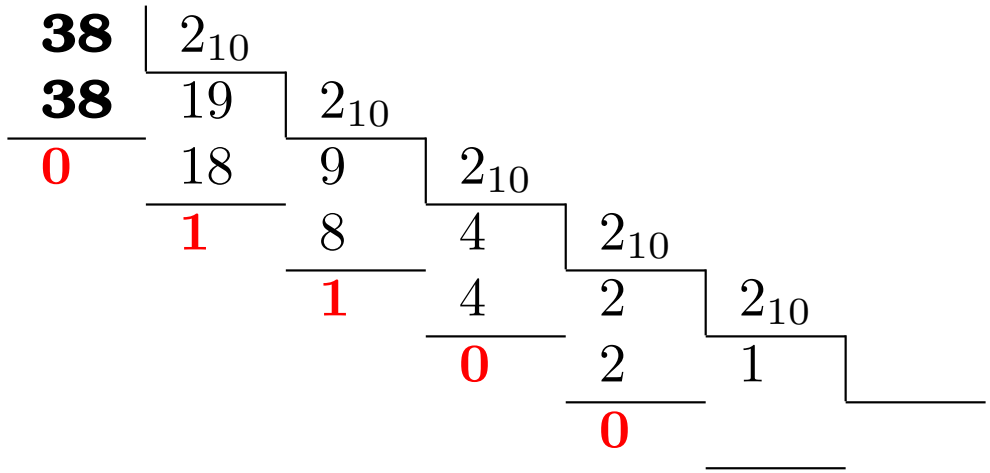
Converting into other number systems

Division by the system's base gives reminder that is the last digit of the number:

$$100110_2/10_2 = 10011.0_2 \quad (\text{quotient} = 10011_2, \text{reminder} = 0)$$

$$10011_2/10_2 = 1001.1_2 \quad (\text{quotient} = 1001_2, \text{reminder} = 1)$$

Thus, repeated division by a *base* gives us digits in that base as *reminders*:



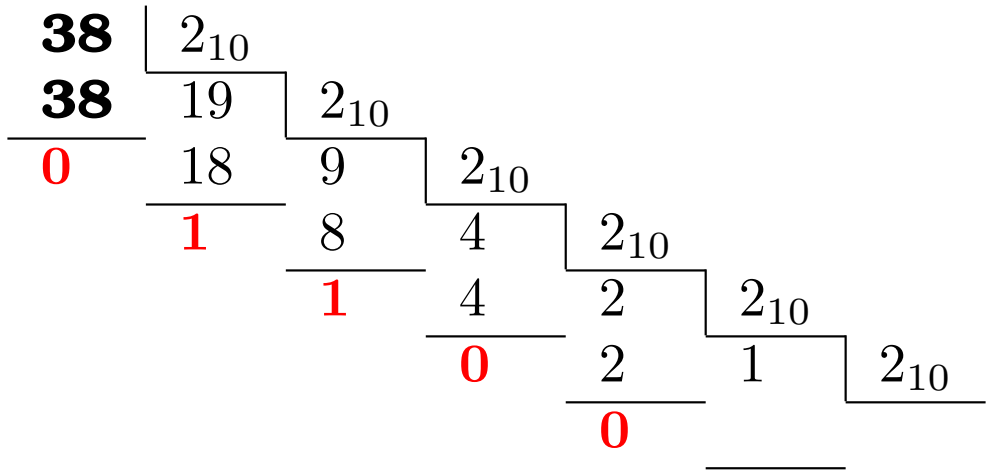
Converting into other number systems

Division by the system's base gives reminder that is the last digit of the number:

$$100110_2/10_2 = 10011.0_2 \quad (\text{quotient} = 10011_2, \text{reminder} = 0)$$

$$10011_2/10_2 = 1001.1_2 \quad (\text{quotient} = 1001_2, \text{reminder} = 1)$$

Thus, repeated division by a *base* gives us digits in that base as *reminders*:



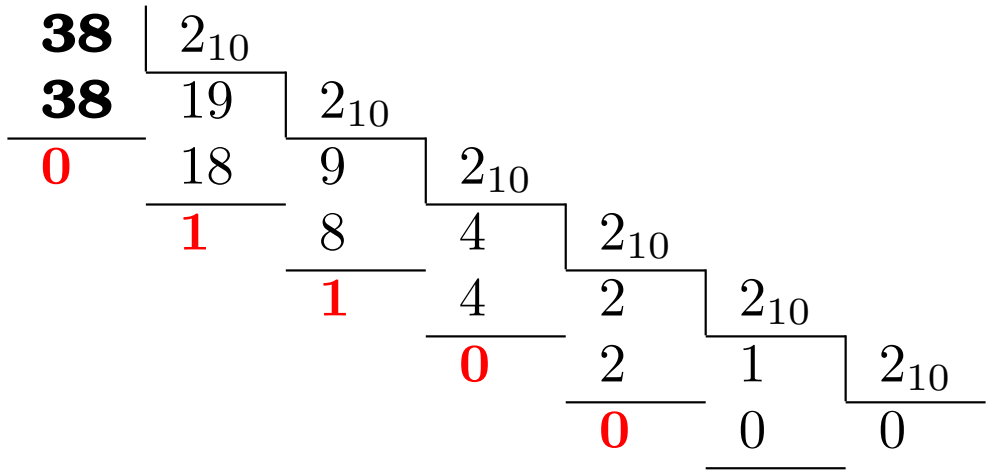
Converting into other number systems

Division by the system's base gives reminder that is the last digit of the number:

$$100110_2/10_2 = 10011.0_2 \quad (\text{quotient} = 10011_2, \text{reminder} = 0)$$

$$10011_2/10_2 = 1001.1_2 \quad (\text{quotient} = 1001_2, \text{reminder} = 1)$$

Thus, repeated division by a *base* gives us digits in that base as *reminders*:



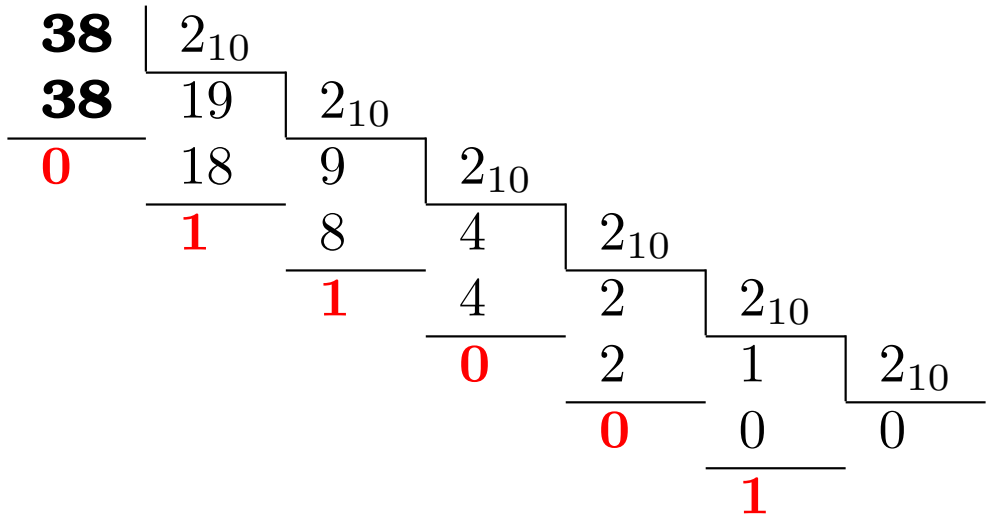
Converting into other number systems

Division by the system's base gives reminder that is the last digit of the number:

$$100110_2/10_2 = 10011.0_2 \quad (\text{quotient} = 10011_2, \text{reminder} = 0)$$

$$10011_2/10_2 = 1001.1_2 \quad (\text{quotient} = 1001_2, \text{reminder} = 1)$$

Thus, repeated division by a *base* gives us digits in that base as *reminders*:



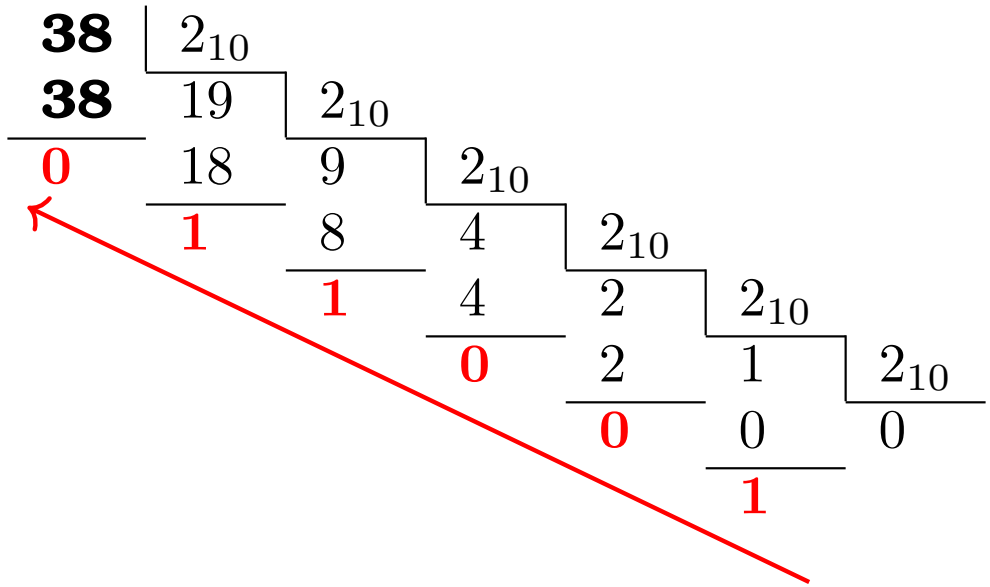
Converting into other number systems

Division by the system's base gives reminder that is the last digit of the number:

$$100110_2/10_2 = 10011.0_2 \quad (\text{quotient} = 10011_2, \text{reminder} = 0)$$

$$10011_2/10_2 = 1001.1_2 \quad (\text{quotient} = 1001_2, \text{reminder} = 1)$$

Thus, repeated division by a *base* gives us digits in that base as *reminders*:



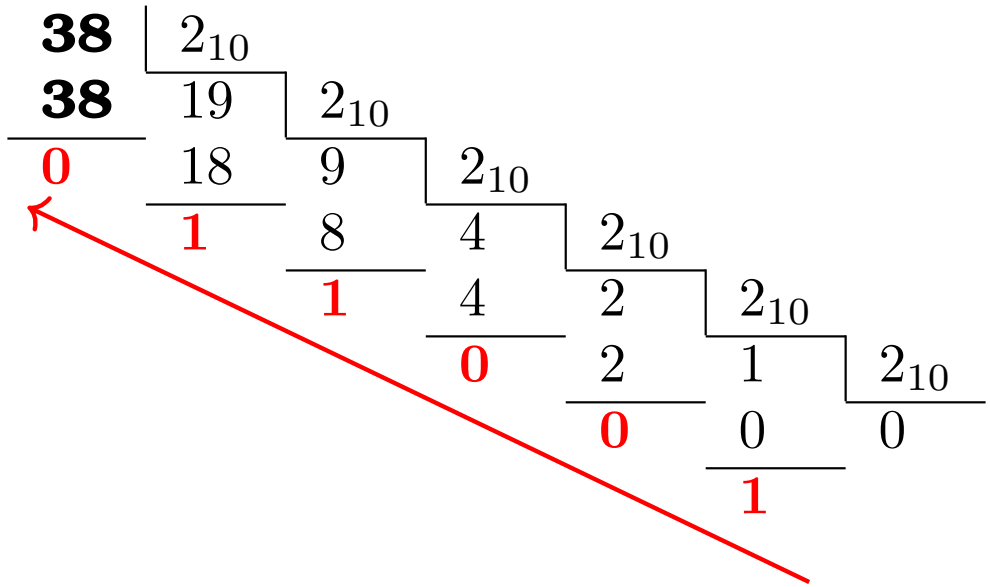
Converting into other number systems

Division by the system's base gives reminder that is the last digit of the number:

$$100110_2/10_2 = 10011.0_2 \quad (\text{quotient} = 10011_2, \text{reminder} = 0)$$

$$10011_2/10_2 = 1001.1_2 \quad (\text{quotient} = 1001_2, \text{reminder} = 1)$$

Thus, repeated division by a *base* gives us digits in that base as *reminders*:



$$100110_2 = 2^5 + 2^2 + 2^1 = 32_{10} + 4_{10} + 2_{10} = \mathbf{38_{10}}$$

Converting fractions

Fractions can be multiplied by the new number base to yield consequently digits after the fraction point:

Converting fractions

Fractions can be multiplied by the new number base to yield consequently digits after the fraction point:

$$0.01101_2 \times 10_2 = 0.1101_2; \quad \lfloor 0.01101_2 \times 10_2 \rfloor = \lfloor 0.1101_2 \rfloor = \mathbf{0}$$

Converting fractions

Fractions can be multiplied by the new number base to yield consequently digits after the fraction point:

$$0.01101_2 \times 10_2 = 0.1101_2; \quad \lfloor 0.01101_2 \times 10_2 \rfloor = \lfloor 0.1101_2 \rfloor = \mathbf{0}$$

$$\mathbf{0}.1101_2 \times 10_2 = 1.101_2; \lfloor 0.1101_2 \times 10_2 \rfloor = \lfloor 1.101_2 \rfloor = \mathbf{1}$$

Converting fractions

Fractions can be multiplied by the new number base to yield consequently digits after the fraction point:

$$0.01101_2 \times 10_2 = 0.1101_2; \quad \lfloor 0.01101_2 \times 10_2 \rfloor = \lfloor 0.1101_2 \rfloor = \mathbf{0}$$

$$\mathbf{0}.1101_2 \times 10_2 = 1.101_2; \lfloor 0.1101_2 \times 10_2 \rfloor = \lfloor 1.101_2 \rfloor = \mathbf{1}$$

$$1.101_2 - \lfloor 1.101_2 \rfloor = \mathbf{0}.101_2$$

Converting fractions

Fractions can be multiplied by the new number base to yield consequently digits after the fraction point:

$$0.01101_2 \times 10_2 = 0.1101_2; \quad \lfloor 0.01101_2 \times 10_2 \rfloor = \lfloor 0.1101_2 \rfloor = \mathbf{0}$$

$$\mathbf{0}.1101_2 \times 10_2 = 1.101_2; \lfloor 0.1101_2 \times 10_2 \rfloor = \lfloor 1.101_2 \rfloor = \mathbf{1}$$

$$1.101_2 - \lfloor 1.101_2 \rfloor = \mathbf{0}.101_2$$

$$\mathbf{0}.101_2 \times 10_2 = 1.01_2; \lfloor 1.01_2 \rfloor = \mathbf{1}$$

Converting fractions

Fractions can be multiplied by the new number base to yield consequently digits after the fraction point:

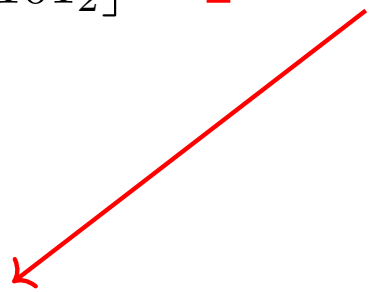
$$0.01101_2 \times 10_2 = 0.1101_2; \quad \lfloor 0.01101_2 \times 10_2 \rfloor = \lfloor 0.1101_2 \rfloor = \mathbf{0}$$

$$\mathbf{0}.1101_2 \times 10_2 = 1.101_2; \lfloor 0.1101_2 \times 10_2 \rfloor = \lfloor 1.101_2 \rfloor = \mathbf{1}$$

$$1.101_2 - \lfloor 1.101_2 \rfloor = \mathbf{0}.101_2$$

$$\mathbf{0}.101_2 \times 10_2 = 1.01_2; \lfloor 1.01_2 \rfloor = \mathbf{1}$$

$$\ldots$$



Converting fractions

Fractions can be multiplied by the new number base to yield consequently digits after the fraction point:

Convert $\mathbf{0.625_{10}}$ to binary.

Converting fractions

Fractions can be multiplied by the new number base to yield consequently digits after the fraction point:

Convert $\mathbf{0.625_{10}}$ to binary.

$$0.625_{10} \times 2_{10} = 1.25_{10}; \quad \lfloor 0.625_{10} \times 2_{10} \rfloor = \lfloor 1.25_{10} \rfloor = \mathbf{1}$$

Converting fractions

Fractions can be multiplied by the new number base to yield consequently digits after the fraction point:

Convert $\mathbf{0.625_{10}}$ to binary.

$$0.625_{10} \times 2_{10} = 1.25_{10}; \quad \lfloor 0.625_{10} \times 2_{10} \rfloor = \lfloor 1.25_{10} \rfloor = \mathbf{1}$$

$$1.25_{10} - \lfloor 1.25_{10} \rfloor = 0.25_{10}$$

Converting fractions

Fractions can be multiplied by the new number base to yield consequently digits after the fraction point:

Convert $\mathbf{0.625_{10}}$ to binary.

$$0.625_{10} \times 2_{10} = 1.25_{10}; \quad \lfloor 0.625_{10} \times 2_{10} \rfloor = \lfloor 1.25_{10} \rfloor = \mathbf{1}$$

$$1.25_{10} - \lfloor 1.25_{10} \rfloor = 0.25_{10}$$

$$0.25_{10} \times 2_{10} = 0.5_{10}; \quad \lfloor 0.25_{10} \times 2_{10} \rfloor = \lfloor 0.5_{10} \rfloor = \mathbf{0}$$

Converting fractions

Fractions can be multiplied by the new number base to yield consequently digits after the fraction point:

Convert $\mathbf{0.625_{10}}$ to binary.

$$0.625_{10} \times 2_{10} = 1.25_{10}; \quad \lfloor 0.625_{10} \times 2_{10} \rfloor = \lfloor 1.25_{10} \rfloor = \mathbf{1}$$

$$1.25_{10} - \lfloor 1.25_{10} \rfloor = 0.25_{10}$$

$$0.25_{10} \times 2_{10} = 0.5_{10}; \quad \lfloor 0.25_{10} \times 2_{10} \rfloor = \lfloor 0.5_{10} \rfloor = \mathbf{0}$$

$$0.5_{10} - \lfloor 0.5_{10} \rfloor = 0.5_{10}$$

Converting fractions

Fractions can be multiplied by the new number base to yield consequently digits after the fraction point:

Convert $\mathbf{0.625_{10}}$ to binary.

$$0.625_{10} \times 2_{10} = 1.25_{10}; \quad \lfloor 0.625_{10} \times 2_{10} \rfloor = \lfloor 1.25_{10} \rfloor = \mathbf{1}$$

$$1.25_{10} - \lfloor 1.25_{10} \rfloor = 0.25_{10}$$

$$0.25_{10} \times 2_{10} = 0.5_{10}; \quad \lfloor 0.25_{10} \times 2_{10} \rfloor = \lfloor 0.5_{10} \rfloor = \mathbf{0}$$

$$0.5_{10} - \lfloor 0.5_{10} \rfloor = 0.5_{10}$$

$$0.5_{10} \times 2_{10} = 1.0_{10}; \lfloor 1.0_{10} \rfloor = \mathbf{1}$$

Converting fractions

Fractions can be multiplied by the new number base to yield consequently digits after the fraction point:

Convert $\mathbf{0.625_{10}}$ to binary.

$$0.625_{10} \times 2_{10} = 1.25_{10}; \quad \lfloor 0.625_{10} \times 2_{10} \rfloor = \lfloor 1.25_{10} \rfloor = \mathbf{1}$$

$$1.25_{10} - \lfloor 1.25_{10} \rfloor = 0.25_{10}$$

$$0.25_{10} \times 2_{10} = 0.5_{10}; \quad \lfloor 0.25_{10} \times 2_{10} \rfloor = \lfloor 0.5_{10} \rfloor = \mathbf{0}$$

$$0.5_{10} - \lfloor 0.5_{10} \rfloor = 0.5_{10}$$

$$0.5_{10} \times 2_{10} = 1.0_{10}; \lfloor 1.0_{10} \rfloor = \mathbf{1}$$

$$1.0_{10} - \lfloor 1.0_{10} \rfloor = 1 - 1 = 0 \Rightarrow \textbf{end}$$

Converting fractions

Fractions can be multiplied by the new number base to yield consequently digits after the fraction point:

Convert $\mathbf{0.625_{10}}$ to binary.

$$0.625_{10} \times 2_{10} = 1.25_{10}; \quad \lfloor 0.625_{10} \times 2_{10} \rfloor = \lfloor 1.25_{10} \rfloor = \mathbf{1}$$

$$1.25_{10} - \lfloor 1.25_{10} \rfloor = 0.25_{10}$$

$$0.25_{10} \times 2_{10} = 0.5_{10}; \quad \lfloor 0.25_{10} \times 2_{10} \rfloor = \lfloor 0.5_{10} \rfloor = \mathbf{0}$$

$$0.5_{10} - \lfloor 0.5_{10} \rfloor = 0.5_{10}$$

$$0.5_{10} \times 2_{10} = 1.0_{10}; \lfloor 1.0_{10} \rfloor = \mathbf{1}$$

$$1.0_{10} - \lfloor 1.0_{10} \rfloor = 1 - 1 = 0 \Rightarrow \textbf{end}$$

Converting fractions

Fractions can be multiplied by the new number base to yield consequently digits after the fraction point:

Convert $\mathbf{0.625_{10}}$ to binary.

$$0.625_{10} \times 2_{10} = 1.25_{10}; \quad \lfloor 0.625_{10} \times 2_{10} \rfloor = \lfloor 1.25_{10} \rfloor = \mathbf{1}$$

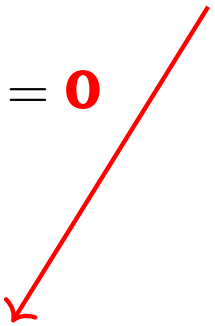
$$1.25_{10} - \lfloor 1.25_{10} \rfloor = 0.25_{10}$$

$$0.25_{10} \times 2_{10} = 0.5_{10}; \quad \lfloor 0.25_{10} \times 2_{10} \rfloor = \lfloor 0.5_{10} \rfloor = \mathbf{0}$$

$$0.5_{10} - \lfloor 0.5_{10} \rfloor = 0.5_{10}$$

$$0.5_{10} \times 2_{10} = 1.0_{10}; \lfloor 1.0_{10} \rfloor = \mathbf{1}$$

$$1.0_{10} - \lfloor 1.0_{10} \rfloor = 1 - 1 = 0 \Rightarrow \textbf{end}$$



0.101_2

Converting fractions

Fractions can be multiplied by the new number base to yield consequently digits after the fraction point:

Convert $\mathbf{0.625_{10}}$ to binary.

$$0.625_{10} \times 2_{10} = 1.25_{10}; \quad \lfloor 0.625_{10} \times 2_{10} \rfloor = \lfloor 1.25_{10} \rfloor = \mathbf{1}$$

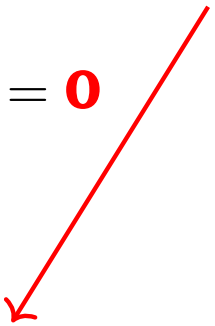
$$1.25_{10} - \lfloor 1.25_{10} \rfloor = 0.25_{10}$$

$$0.25_{10} \times 2_{10} = 0.5_{10}; \quad \lfloor 0.25_{10} \times 2_{10} \rfloor = \lfloor 0.5_{10} \rfloor = \mathbf{0}$$

$$0.5_{10} - \lfloor 0.5_{10} \rfloor = 0.5_{10}$$

$$0.5_{10} \times 2_{10} = 1.0_{10}; \lfloor 1.0_{10} \rfloor = \mathbf{1}$$

$$1.0_{10} - \lfloor 1.0_{10} \rfloor = 1 - 1 = 0 \Rightarrow \mathbf{end}$$



$$0.101_2 = 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = 0.5_{10} + 0.125_{10} = \mathbf{0.625_{10}}$$

Converting to/from base 8 and base 16

Dividing by 8_{10} gives the **3** last binary digits as a reminder:

$$1101\,1011_2/8_{10} \equiv 1101\,1011_2/1000_2 = 1101\,1.\mathbf{011}_2$$

Dividing by 16_{10} gives the **4** last binary digits as a reminder:

$$1101\,1011_2/16_{10} \equiv 1101\,1011_2/10000_2 = 1101\,.\mathbf{1011}_2$$

Therefore:

To convert to octal (hexadecimal), group your binary digits by 3 (4) and convert each group to a corresponding digit:

Octal digits			Hex digits			
000	0		0000	0	1000	8
001	1		0001	1	1001	9
010	2		0010	2	1010	A
011	3		0011	3	1011	B
100	4		0100	4	1100	C
101	5		0101	5	1101	D
110	6		0110	6	1110	E
111	7		0111	7	1111	F

$\longleftarrow$

Example (octal):

$01011011_2 = \mathbf{001\ 011\ 011_2} = \mathbf{133_8}$

$\longrightarrow$

Example (hexadecimal):

$01011011_2 = \mathbf{0101\ 1011_2} = \mathbf{5B_{16}}$

XXI century: converting with computer

Manual:

```
man perlfaq4
```

Binary $\rightarrow$ decimal:

```
perl -le 'print 0b_0110_1011'
107
```

Hex $\rightarrow$ binary:

```
perl -e 'printf "%016b\n", 0x23FF'
0010001111111111
```

Octal $\rightarrow$ hex:

```
perl -e 'printf "%02X\n", 0133'
5B
```

Caution: this works as expected only for small numbers ($|n| < 2^{31}$)!

XXI century: converting fractions

Convert binary fraction $0.0110\,1011_2 = 0110\,1011_2/1\,0000\,0000_2$ to decimal:

```
perl -le 'print 0b_0110_1011 / (2**8)'
0.41796875
```

XXI century: converting fractions

Convert binary fraction $0.0110\,1011_2 = 0110\,1011_2/1\,0000\,0000_2$ to decimal:

```
perl -le 'print 0b_0110_1011 / (2**8)'
0.41796875
```

Check it:

```
perl -e '$x=0.41796875; print "0."; while($x) {$x*=2; print int($x); $x-=int($x)}; print "\n"'
0.01101011
```

Caution: this method (and the check) works ***only*** if the fraction is **exactly** representable in our computer!

Convert 0.1_{10}

$$0.1_{10} = 1/10_{10}$$

File: `"examples/code/one-tenth.c"`

```c
#include <stdio.h>
#include <stdlib.h>

int main( int argc, char *argv[] )
{
    int n = 1;  // numerator
    int d = 10; // denominator
    int b = 2;  // the base of the target number system
    int s = 0;  // current step
    int maxstep = 20;

    /* Get the max. digit count: */
    printf( "%d.", n/d );
    while( n != 0 && s < maxstep ) {
        n *= b;
        printf( "%d", n/d ); //  |_ b*(n/d) _|
        n %= d;              //  b*(n/d) - |_ b*(n/d) _|
        s++;
        if( s % 4 == 0 ) {
            printf( " " );   // print space after each 4 digits
        }
    }
    printf( "\n" );

    return 0;
}
```

Convert 0.1_{10}

$$0.1_{10} = 1/10_{10}$$

```
sh$ examples/code/one-tenth
0.0001 1001 1001 1001 1001
```

$$0.1_{10} = 0.0001\,\overline{1001}_2$$

Infinite periodic (repeating) binary fraction!

If the fully reduced denominator of a fraction has a prime factor that is ***not*** a factor of a number base, then expansion is non-terminating in that base.

E.g. $1/10_{10}$:

$$10_{10} = 2 \times 5; \quad 5 \text{ is not a factor of } 2$$
$$\Rightarrow \quad 1/10_{10} = 0.0001\,\overline{1001}_2 \text{ is non-terminating}$$

Binary arithmetic

/.../ Būtinas minimumas yra sudėtis, nes atimtis tai yra sudėtis su papildomu kodu, daugyba – daug sudėčių cikle, o dalyba – daug atimčių.

The single required operation is addition, since subtraction is addition in complementary code, multiplication is multiple additions in a loop and division is multiple subtractions.

Antanas Mitašiūnas (2016)
"Kompiuterių architektūra", p. 9

Addition:

$$A + B$$

		B	
	+	0	1
A	0	0	1
	1	1	10

Binary arithmetic

/.../ Būtinas minimumas yra sudėtis, nes atimtis tai yra sudėtis su papildomu kodu, daugyba – daug sudėčių cikle, o dalyba – daug atimčių.

The single required operation is addition, since subtraction is addition in complementary code, multiplication is multiple additions in a loop and division is multiple subtractions.

Antanas Mitašiūnas (2016)
"Kompiuterių architektūra", p. 9

Addition:

$A + B$

A \ B	0	1
0	0	1
1	1	10

Multiplication:

$A \times B$

A \ B	0	1
0	0	0
1	0	1

Binary arithmetic

Addition:

$$A + B$$

A \ B	0	1
0	0	1
1	1	10

Multiplication:

$$A \times B$$

A \ B	0	1
0	0	0
1	0	1

$$\begin{array}{r} 11101 \\ + \quad 11001 \\ \hline \end{array}$$

Binary arithmetic

Addition:

$$A + B$$

A \ B	0	1
0	0	1
1	1	10

Multiplication:

$$A \times B$$

A \ B	0	1
0	0	0
1	0	1

$$\begin{array}{rr} & {\scriptstyle 1} \\ + & 11101 \\ & 11001 \\ \hline & 0 \end{array}$$

Binary arithmetic

Addition:

$$A + B$$

A \ B	0	1
0	0	1
1	1	10

Multiplication:

$$A \times B$$

A \ B	0	1
0	0	0
1	0	1

$$\begin{array}{rr} & 1 \\ & 11101 \\ + & 11001 \\ \hline & 10 \end{array}$$

Binary arithmetic

Addition:

$$A + B$$

A \ B	0	1
0	0	1
1	1	10

Multiplication:

$$A \times B$$

A \ B	0	1
0	0	0
1	0	1

$$\begin{array}{rr} & 1 \\ & 11101 \\ + & 11001 \\ \hline & 110 \end{array}$$

Binary arithmetic

Addition:

$A + B$

A \ B	0	1
0	0	1
1	1	10

Multiplication:

$A \times B$

A \ B	0	1
0	0	0
1	0	1

$$\begin{array}{rr} & {\scriptstyle 1\quad 1} \\ & 11101 \\ + & 11001 \\ \hline & 0110 \end{array}$$

Binary arithmetic

Addition:

$$A + B$$

A \ B	0	1
0	0	1
1	1	10

Multiplication:

$$A \times B$$

A \ B	0	1
0	0	0
1	0	1

$$\begin{array}{rr} & {\scriptstyle 1\,1} \quad {\scriptstyle 1} \\ & 1\,1\,1\,0\,1 \\ + & 1\,1\,0\,0\,1 \\ \hline & 1\,0\,1\,1\,0 \end{array}$$

Binary arithmetic

Addition:

$$A + B$$

A \ +	B: 0	B: 1
0	0	1
1	1	10

Multiplication:

$$A \times B$$

A \ ×	B: 0	B: 1
0	0	0
1	0	1

$$\begin{array}{r} {\scriptstyle 11\quad 1} \\ 11101 \\ + \quad 11001 \\ \hline 110110 \end{array}$$

Binary arithmetic

Addition:

$$A + B$$

A \ B	0	1
0	0	1
1	1	10

$$\begin{array}{rr} & {\scriptstyle 1\,1} \quad {\scriptstyle 1} \\ & 1\,1\,1\,0\,1 \\ + & 1\,1\,0\,0\,1 \\ \hline & 1\,1\,0\,1\,1\,0 \end{array}$$

Multiplication:

$$A \times B$$

A \ B	0	1
0	0	0
1	0	1

$$\begin{array}{rr} & 1\,1\,0\,1 \\ \times & 1\,0\,1 \\ \hline & \\ & \\ & \\ \hline & \\ & \\ \hline \end{array}$$

Binary arithmetic

Addition:

$A + B$

A \ B	0	1
0	0	1
1	1	10

$$\begin{array}{r} {\scriptstyle 1\,1\quad\ 1} \\ 11101 \\ +\ 11001 \\ \hline 110110 \end{array}$$

Multiplication:

$A \times B$

A \ B	0	1
0	0	0
1	0	1

$$\begin{array}{r} 1101 \\ \times\quad 101 \\ \hline 1101 \\ \\ \hline \end{array}$$

Binary arithmetic

Addition:

$$A + B$$

A \ B	0	1
0	0	1
1	1	10

$$\begin{array}{rr} & {\scriptstyle 1\,1} \quad {\scriptstyle 1} \\ & 1\,1\,1\,0\,1 \\ + & 1\,1\,0\,0\,1 \\ \hline & 1\,1\,0\,1\,1\,0 \end{array}$$

Multiplication:

$$A \times B$$

A \ B	0	1
0	0	0
1	0	1

$$\begin{array}{rr} & 1101 \\ \times & 101 \\ \hline & \\ & 1101 \\ & 0000 \\ & \\ \hline & \end{array}$$

Binary arithmetic

Addition:

$A + B$

A \ B	0	1
0	0	1
1	1	10

$$\begin{array}{r} {\scriptstyle 1\,1\quad\;\,1} \\ 11101 \\ +\;\;11001 \\ \hline 110110 \end{array}$$

Multiplication:

$A \times B$

A \ B	0	1
0	0	0
1	0	1

$$\begin{array}{r} 1101 \\ \times\quad\;\;101 \\ \hline 1101 \\ 0000 \\ 1101 \\ \hline \end{array}$$

Binary arithmetic

Addition:

$A + B$

A \ B	0	1
0	0	1
1	1	10

$$\begin{array}{r} {\scriptstyle 1\,1\quad\ 1} \\ 11101 \\ +\ 11001 \\ \hline 110110 \end{array}$$

Multiplication:

$A \times B$

A \ B	0	1
0	0	0
1	0	1

$$\begin{array}{r} 1101 \\ \times\quad 101 \\ \hline 111 \\ 1101 \\ 0000 \\ 1101 \\ \hline 1000001 \end{array}$$

Bitwise logic operations

We will abbreviate the term *binary digit* to *bit*.

Useful operations can be performed on the single bits of a number independently.

Mask bits:

$$\begin{array}{rr} & 1101\,1010\,0111 \\ \text{AND} & 0001\,1111\,1000 \\ \hline & 0001\,1010\,0000 \end{array}$$

Bitwise logic operations

We will abbreviate the term *binary digit* to *bit*.

Useful operations can be performed on the single bits of a number independently.

Mask bits:

```
      1101 1010 0111
AND   0001 1111 1000
-----------------------
      0001 1010 0000
```

Set bits:

```
      1101 1010 0111
OR    0001 1111 1000
-----------------------
      1101 1111 1111
```

Bitwise logic operations

We will abbreviate the term *binary digit* to *bit*.

Useful operations can be performed on the single bits of a number independently.

Mask bits:

$$\begin{array}{rr} & 1101\,1010\,0111 \\ \text{AND} & 0001\,1111\,1000 \\ \hline & 0001\,1010\,0000 \end{array}$$

Set bits:

$$\begin{array}{rr} & 1101\,1010\,0111 \\ \text{OR} & 0001\,1111\,1000 \\ \hline & 1101\,1111\,1111 \end{array}$$

Questions:

$$2\text{B OR NOT } 2\text{B} = ??$$

Bitwise logic operations

We will abbreviate the term *binary digit* to *bit*.

Useful operations can be performed on the single bits of a number independently.

Mask bits:

$$\begin{array}{rr} & 1101\,1010\,0111 \\ \text{AND} & 0001\,1111\,1000 \\ \hline & 0001\,1010\,0000 \end{array}$$

Set bits:

$$\begin{array}{rr} & 1101\,1010\,0111 \\ \text{OR} & 0001\,1111\,1000 \\ \hline & 1101\,1111\,1111 \end{array}$$

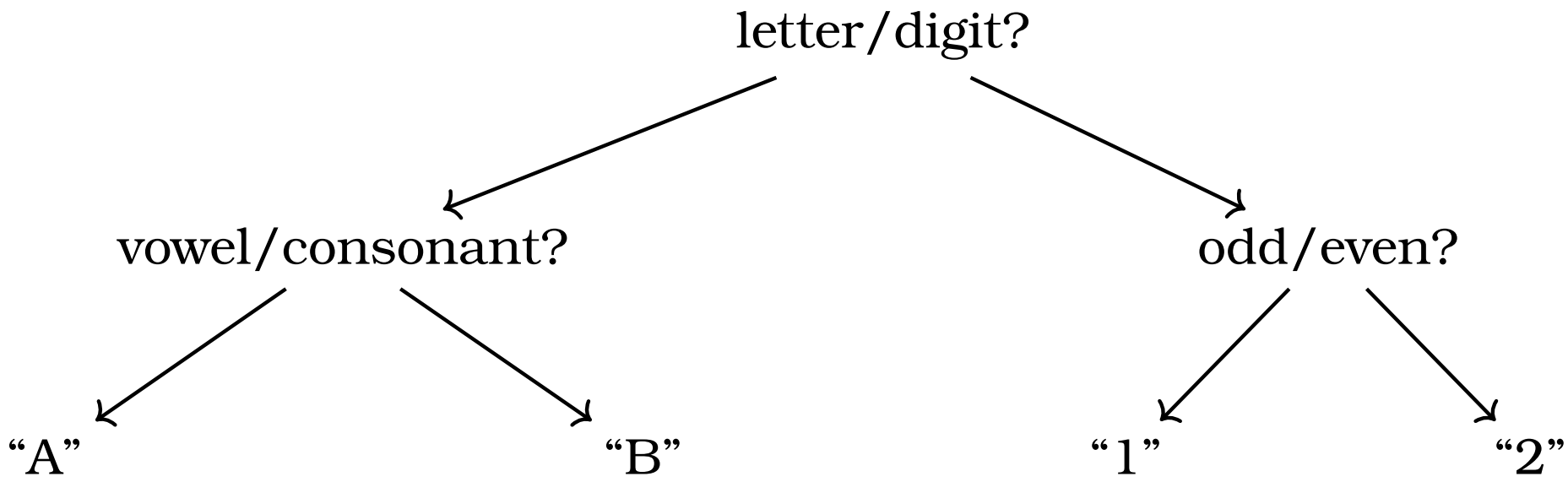
Questions:

$$2\text{B OR NOT } 2\text{B} = ??$$

What is the bit mask to check whether the number is odd?

Intuitive understanding of information content

Find out which of four symbols was transmitted by asking yes/no questions: “A”, “B”, “1”, “2”.



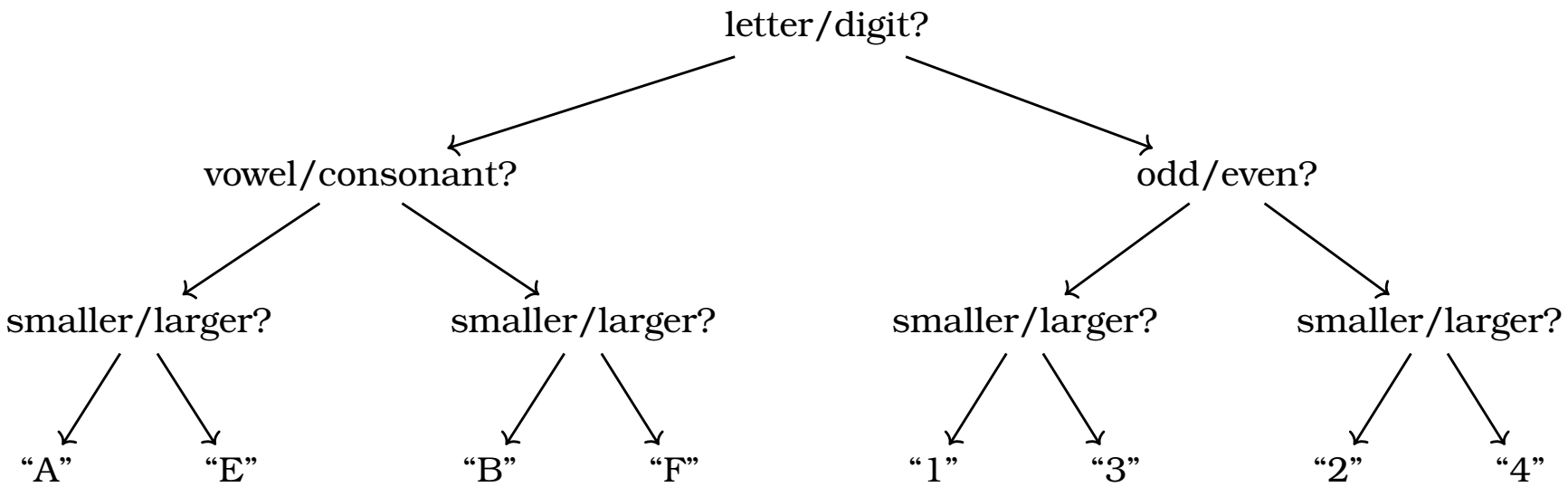
$$N_{\text{symbols}} = N_{\text{leaves}} = 2^{n_{\text{questions}}} = 2^{n_{\text{layers}}}$$

$$n_{\text{questions}} = n_{\text{layers}} = \log_2 N_{\text{symbols}} = \log_2 \frac{1}{p_{\text{symbol}}} = -\log_2 p_{\text{symbol}}$$

$$n_{\text{q}} = \log_2 N_{\text{s}} = \log_2 4 = -\log_2 \tfrac{1}{4} = \log_2 2^2 = 2$$

Intuitive understanding of information (2)

One out of 8 symbols: “A”, “B”, “E”, “F”, “1”, “2”, “3”, “4”.



$$N_{\text{symbols}} = N_{\text{leaves}} = 2^{n_{\text{questions}}} = 2^{n_{\text{layers}}}$$

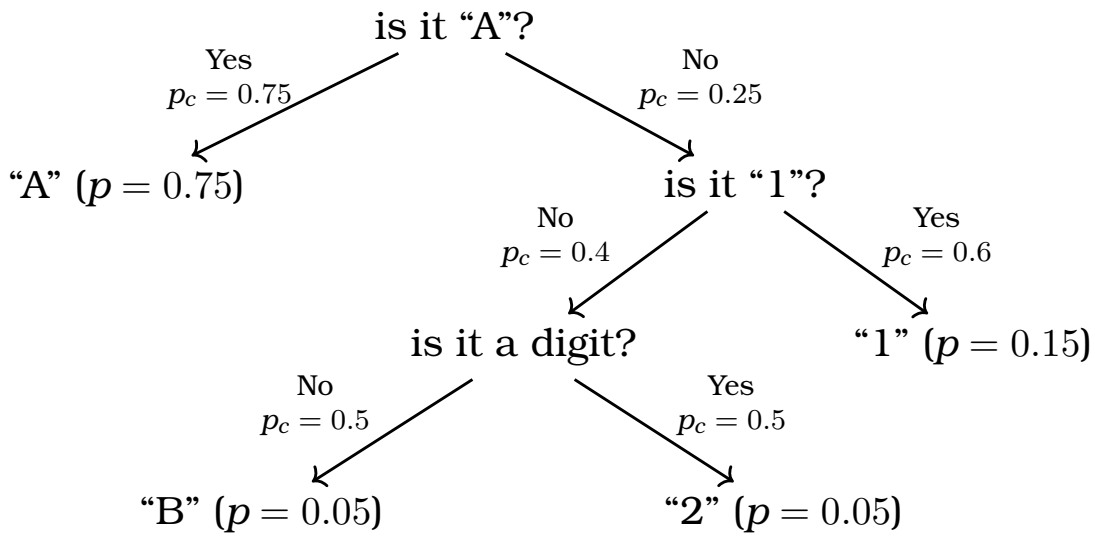
$$n_{\text{questions}} = n_{\text{layers}} = \log_2 N_{\text{symbols}} = \log_2 \frac{1}{p_{\text{symbol}}} = -\log_2 p_{\text{symbol}}$$

$$n_{\text{q}} = \log_2 N_{\text{s}} = \log_2 8 = -\log_2 \tfrac{1}{8} = \log_2 2^3 = 3$$

Info content depends in probability

One out of 4 symbols: “A”, “B”, “1”, “2”

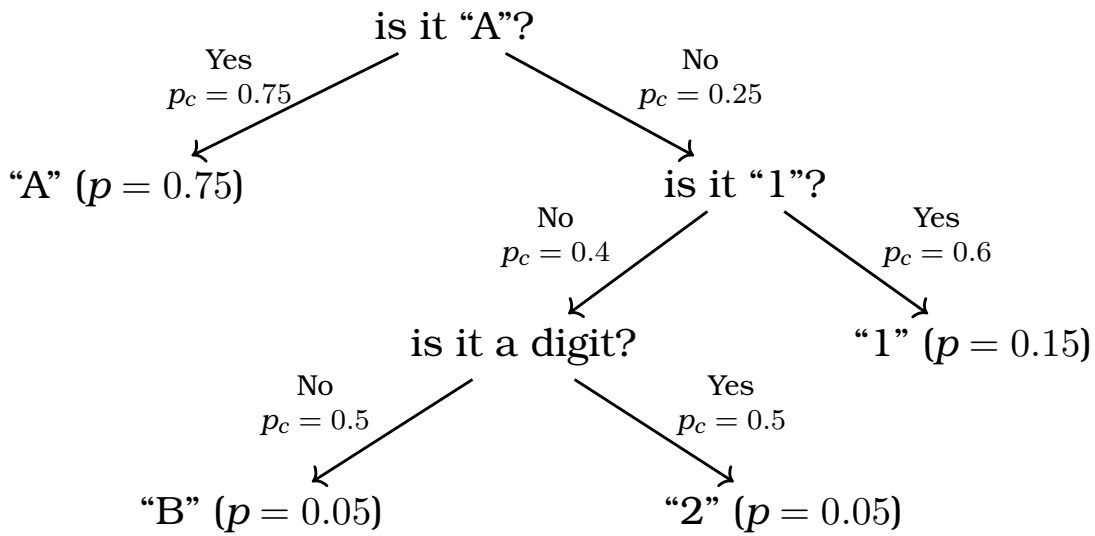
“AAA1AAABA1A2AAA1AAAA”
(20 symbols)



Info content depends in probability

One out of 4 symbols: “A”, “B”, “1”, “2”

“AAA1AAABA1A2AAA1AAAA”
(20 symbols)

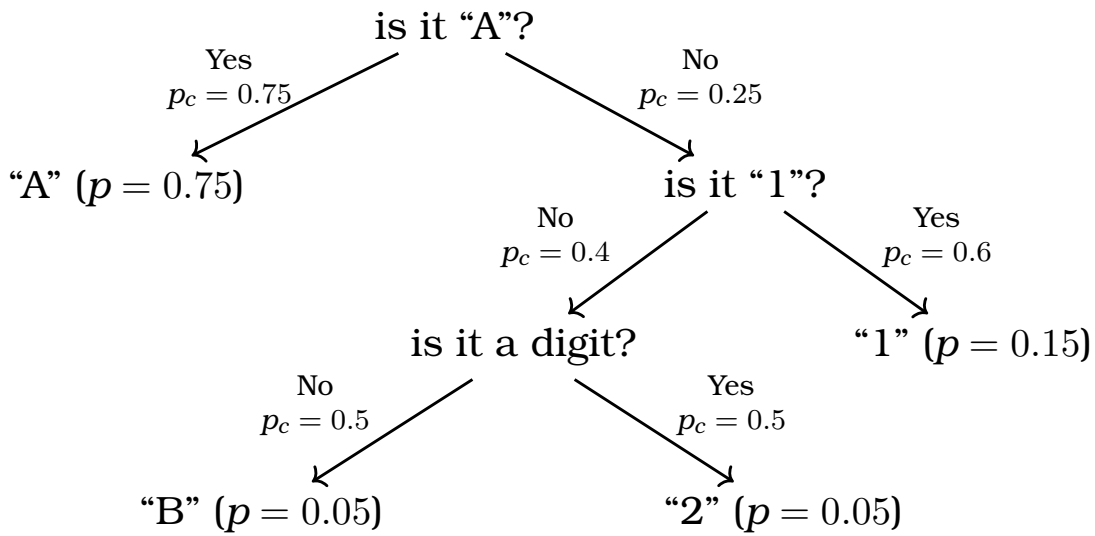


For equal probabilities: $n_{\mathrm{q}} = \log_2 N_{\mathrm{s}} = \log_2 \frac{1}{p_{\mathrm{s}}} = -\log_2 p_{\mathrm{s}} = \log_2 4 = \mathbf{2}$

Info content depends in probability

One out of 4 symbols: “A”, “B”, “1”, “2”

“AAA1AAABA1A2AAA1AAAA”
(20 symbols)

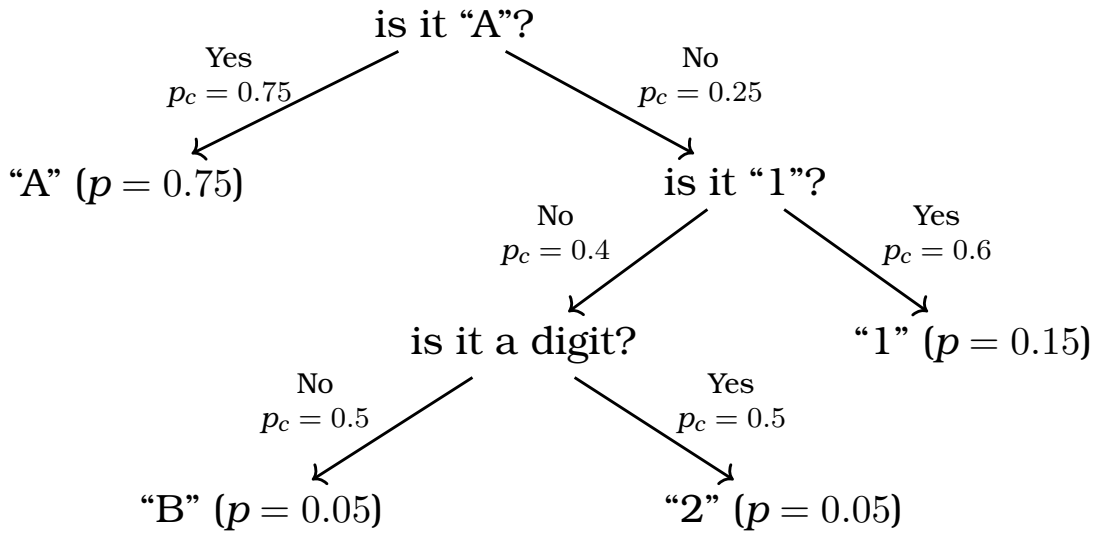


For equal probabilities: $n_\mathrm{q} = \log_2 N_\mathrm{s} = \log_2 \frac{1}{p_\mathrm{s}} = -\log_2 p_\mathrm{s} = \log_2 4 = \mathbf{2}$
For the specified question sequence (Huffman encoding):
$n_\mathrm{q} = 0.75 + 2 \times 0.15 + 2 \times 3 \times 0.05 = \mathbf{1.35}$

Info content depends in probability

One out of 4 symbols: “A”, “B”, “1”, “2”

“AAA1AAABA1A2AAA1AAAA”
(20 symbols)



For equal probabilities: $n_\mathrm{q} = \log_2 N_\mathrm{s} = \log_2 \frac{1}{p_\mathrm{s}} = -\log_2 p_\mathrm{s} = \log_2 4 = \mathbf{2}$
For the specified question sequence (Huffman encoding):
$n_\mathrm{q} = 0.75 + 2 \times 0.15 + 2 \times 3 \times 0.05 = \mathbf{1.35}$
Theoretical limit: $n_\mathrm{q} = -\sum_{i=1}^{4} p_i \log_2 p_i = \mathbf{1.15}$

Bits and bytes. Information content

Assume we have a source that can send messages from the set $S = \{M_1, \ldots, M_n\},\ n \in \mathbb{N}$.
Then:

$$S = \{M_1, \ldots, M_n\}$$

$$p_i = Pr(M_i), \quad i = 1 \ldots n$$

$$I_{M_i} = -\log p_i$$

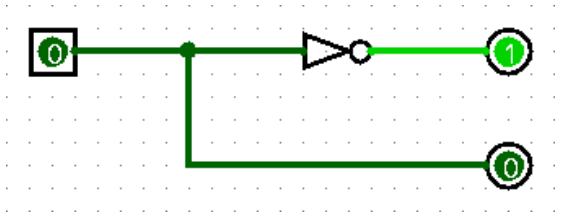
$$H = -\sum_{i=1}^{n} p_i \log p_i$$

Here:

$Pr(M_i)$	–	probability of message M_i;
I_{M_i}	–	information content of message M_i;
H	–	entropy of the source, or average information content per message

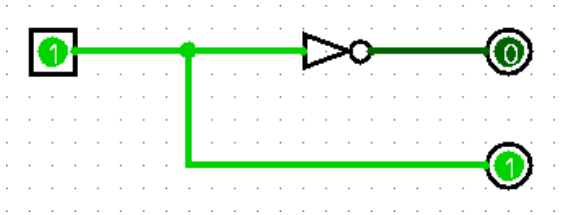
Two level signal $\neq$ 1 bit

Two-wire single bit representation:



Two level signal $\neq$ 1 bit

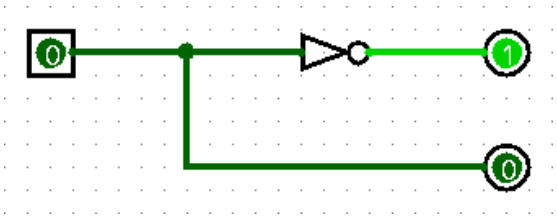
Two-wire single bit representation:



Two level signal $\neq$ 1 bit

$$
\begin{aligned}
Pr(\text{"0"}) &= Pr(\text{"1"}) = 0.5 \\
H &= -2 \cdot 0.5 \cdot \log_2 0.5 \\
&= -2 \cdot (-0.5) \\
&= \textbf{1 bit/message}
\end{aligned}
$$

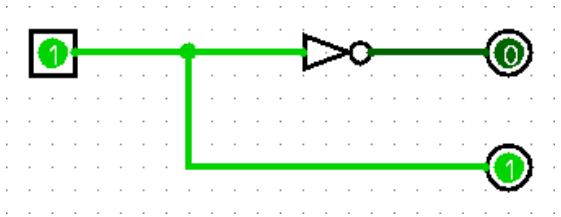
$$
\begin{aligned}
Pr(\text{"01"}) &= Pr(\text{"10"}) = 0.5 \\
Pr(\text{"00"}) &= Pr(\text{"11"}) = 0 \\
H &= - Pr(\text{"01"}) \cdot \log_2 Pr(\text{"01"}) \\
&\quad - Pr(\text{"10"}) \cdot \log_2 Pr(\text{"10"}) \\
&\quad - \lim_{Pr(\text{"00"}) \to 0} Pr(\text{"00"}) \cdot \log_2 Pr(\text{"00"}) \\
&\quad - \lim_{Pr(\text{"11"}) \to 0} Pr(\text{"11"}) \cdot \log_2 Pr(\text{"11"}) \\
&= - 2 \cdot 0.5 \cdot (-1) \\
&= \textbf{1 bit/message}
\end{aligned}
$$



Two level signal $\neq$ 1 bit

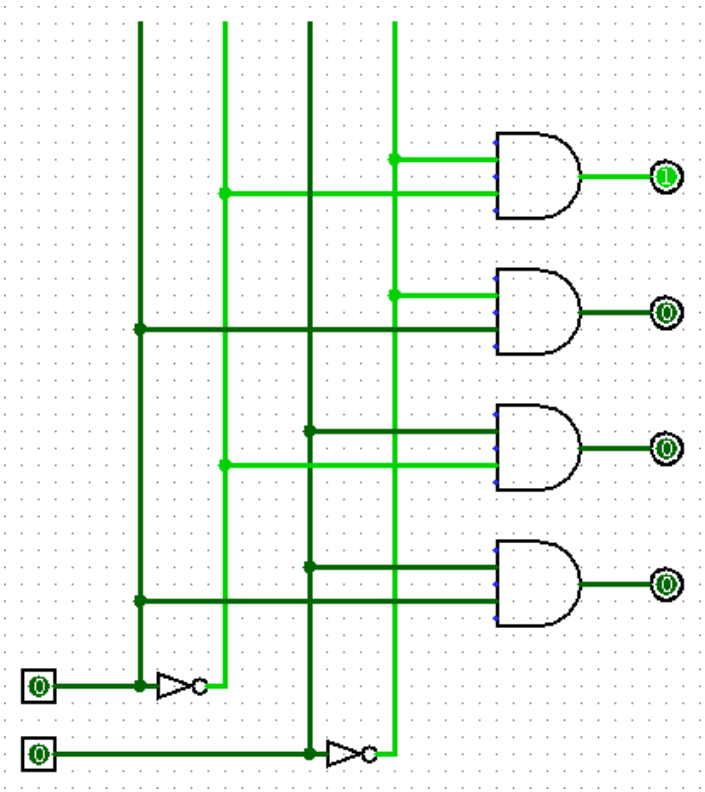
$$
\begin{aligned}
Pr(\text{"0"}) &= Pr(\text{"1"}) = 0.5 \\
H &= -2 \cdot 0.5 \cdot \log_2 0.5 \\
&= -2 \cdot (-0.5) \\
&= \textbf{1 bit/message}
\end{aligned}
$$

$$
\begin{aligned}
Pr(\text{"01"}) &= Pr(\text{"10"}) = 0.5 \\
Pr(\text{"00"}) &= Pr(\text{"11"}) = 0 \\
H &= - Pr(\text{"01"}) \cdot \log_2 Pr(\text{"01"}) \\
&\quad - Pr(\text{"10"}) \cdot \log_2 Pr(\text{"10"}) \\
&\quad - \lim_{Pr(\text{"00"}) \to 0} Pr(\text{"00"}) \cdot \log_2 Pr(\text{"00"}) \\
&\quad - \lim_{Pr(\text{"11"}) \to 0} Pr(\text{"11"}) \cdot \log_2 Pr(\text{"11"}) \\
&= - 2 \cdot 0.5 \cdot (-1) \\
&= \textbf{1 bit/message}
\end{aligned}
$$



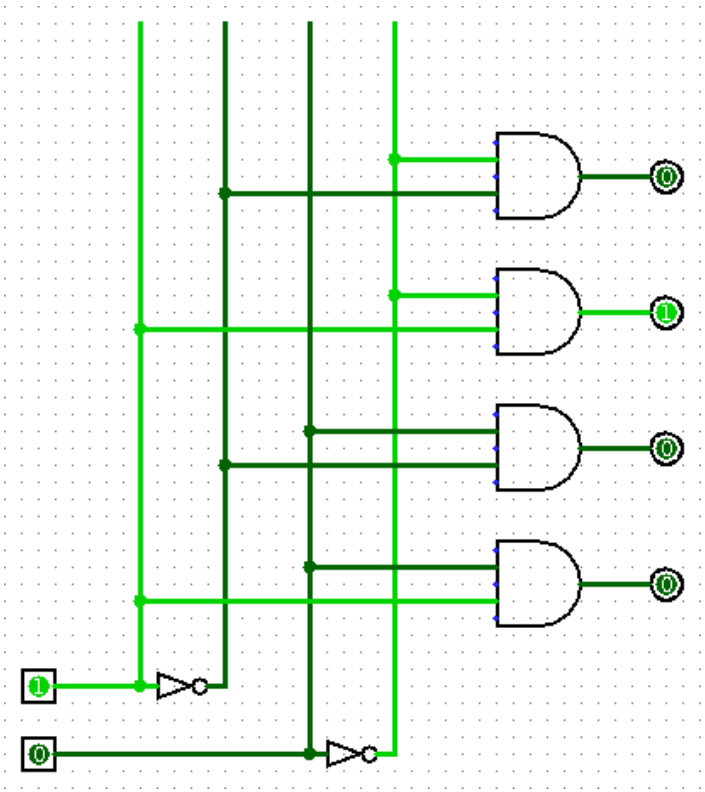
Two level signal $\neq$ 1 bit

Decoder 1-of-4



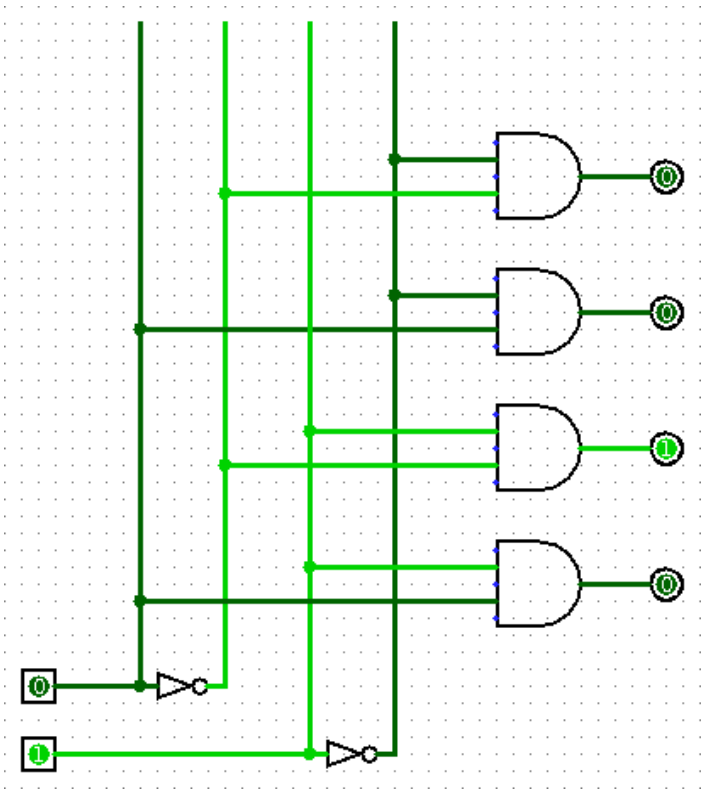
Two level signal $\neq$ 1 bit

Decoder 1-of-4



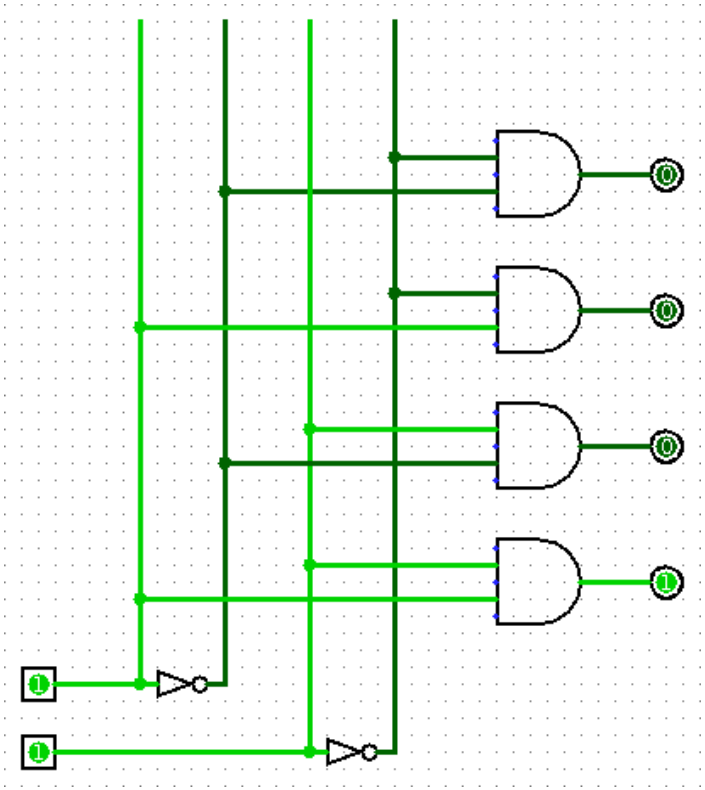
Two level signal $\neq$ 1 bit

Decoder 1-of-4



Two level signal $\neq$ 1 bit

Decoder 1-of-4



Half-adder

Addition:

$A + B$

A \ +	B: 0	B: 1
0	0	1
1	1	10

Sum:

A	**B**	**S**
0	0	0
0	1	1
1	0	1
1	1	0

Carry:

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

Half-adder

Addition:

$A + B$

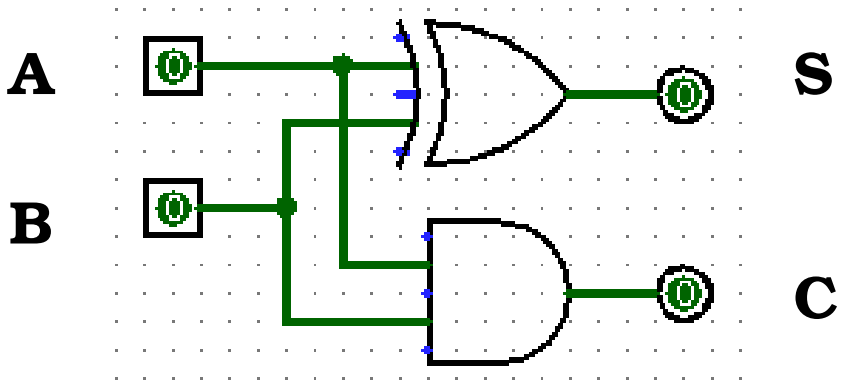
A \ +	B = 0	B = 1
0	0	1
1	1	10

Sum:

A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

Carry:

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



Half-adder

Addition:

$A + B$

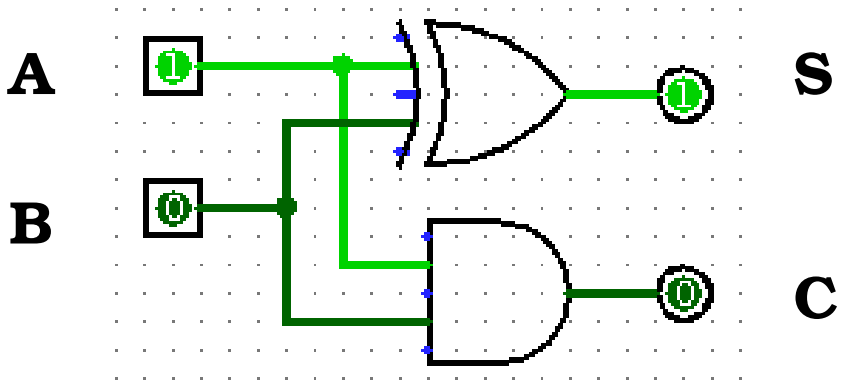
A \ B	0	1
0	0	1
1	1	10

Sum:

A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

Carry:

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



Half-adder

Addition:

$A + B$

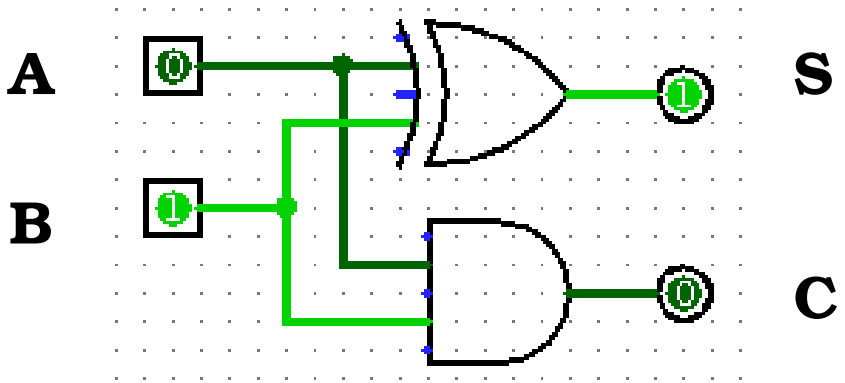
A \ B	0	1
0	0	1
1	1	10

Sum:

A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

Carry:

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



Half-adder

Addition:

$A + B$

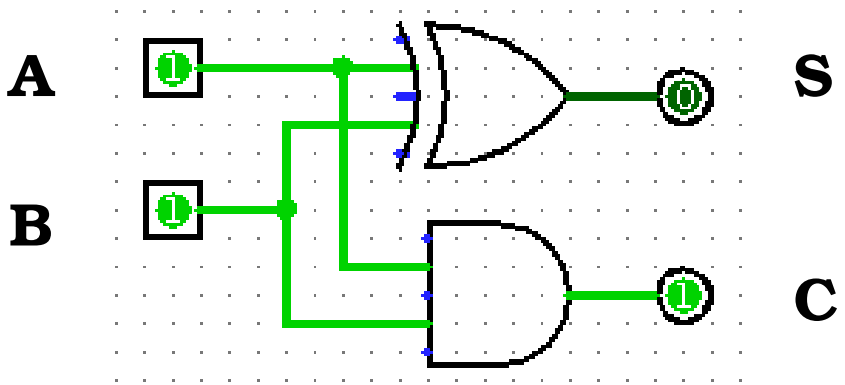
A \ +	B = 0	B = 1
0	0	1
1	1	10

Sum:

A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

Carry:

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



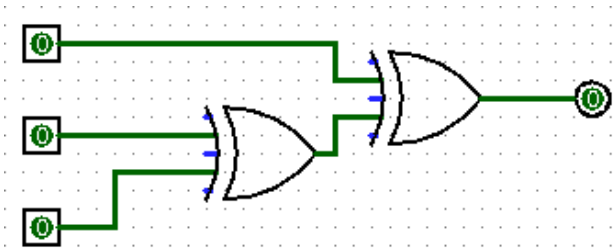
Full adder

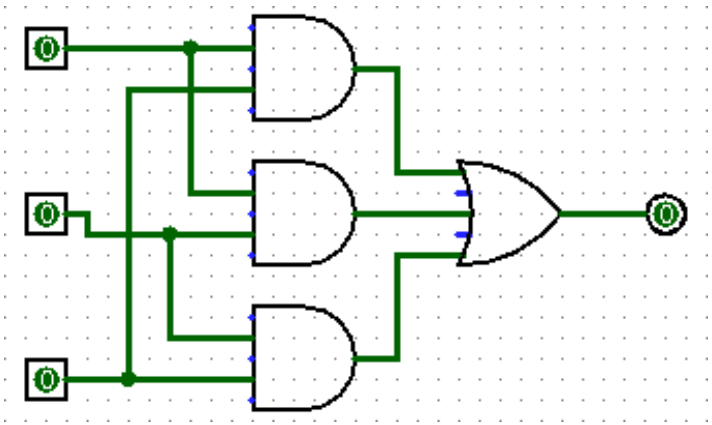
Sum:

C_{in}	A	B	S
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Carry:

C_{in}	A	B	C_{out}		
0	0	0	0		
0	0	1	0		
0	1	0	0		
0	1	1	1	$\overline{C_{in}}AB$	AB
1	0	0	0		
1	0	1	1	$C_{in}\overline{A}B$	$C_{in}B$
1	1	0	1	$C_{in}A\overline{B}$	$C_{in}A$
1	1	1	1	$C_{in}AB$	

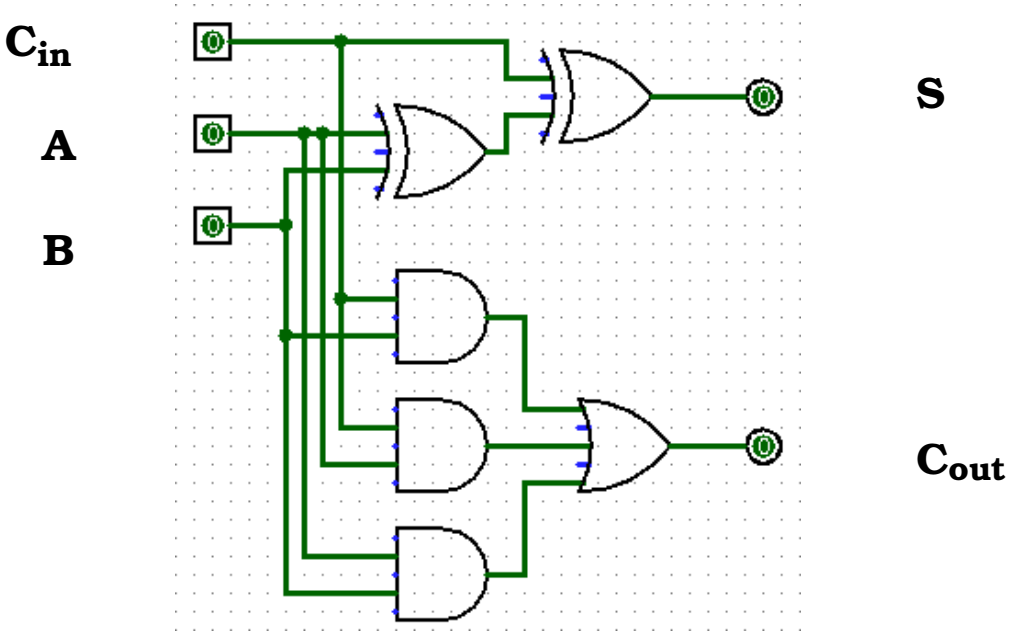




Full adder (naïve)

Sum and Carry:

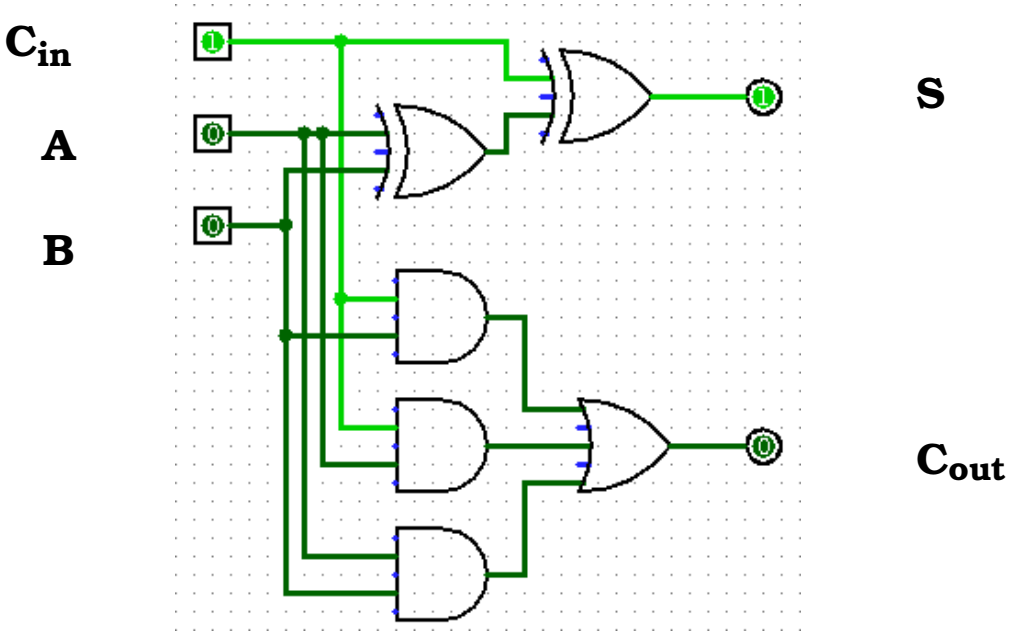
C_{in}	A	B	C_{out}	S
0	**0**	**0**	**0**	**0**
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



Full adder (naïve)

Sum and Carry:

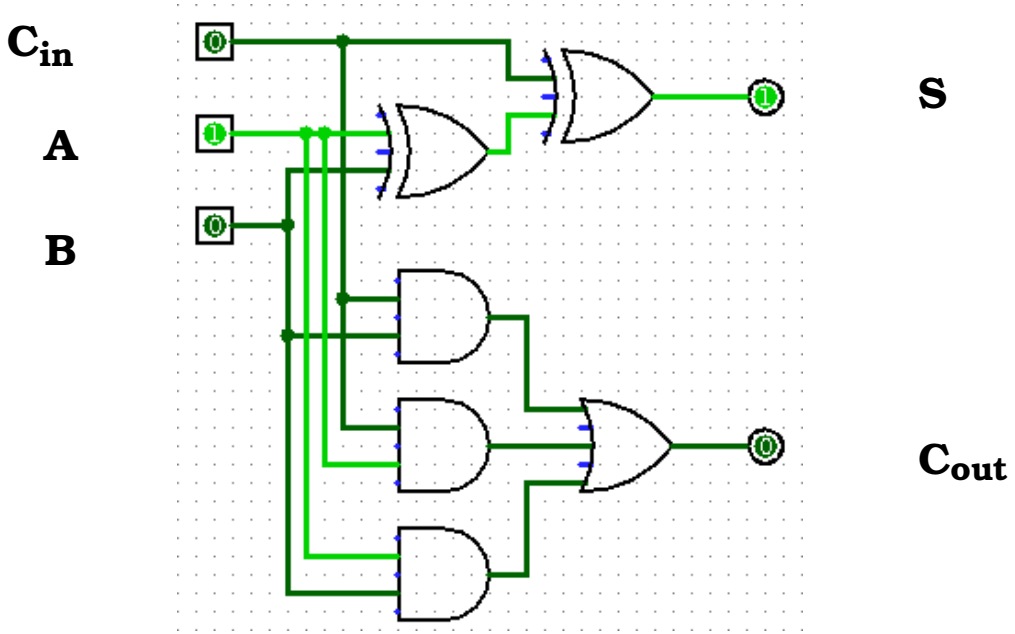
C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	**0**	**0**	**0**	**1**
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



Full adder (naïve)

Sum and Carry:

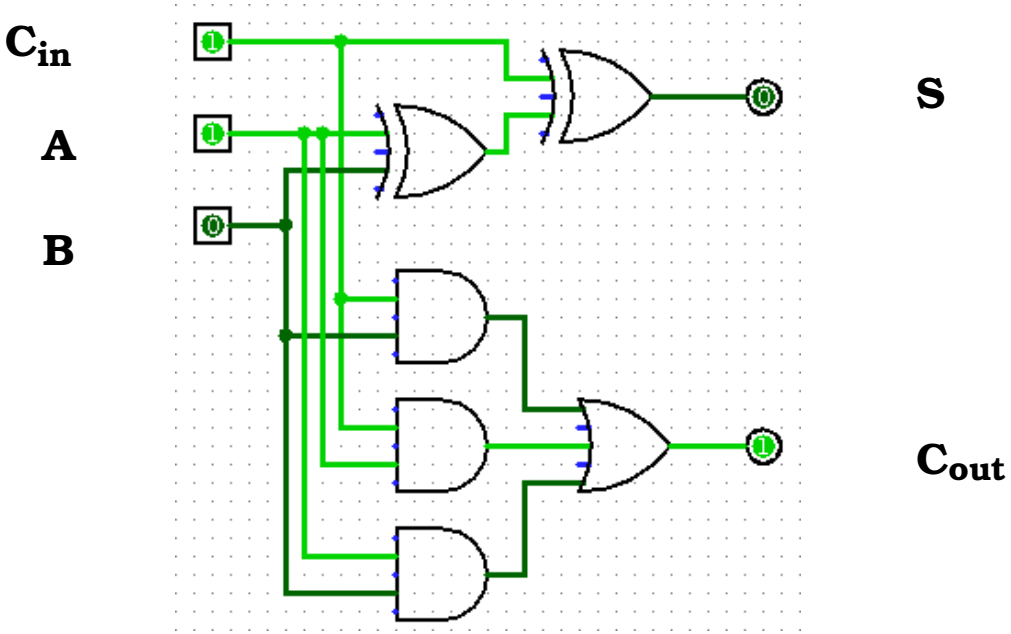
C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	**0**	**1**	**0**	**1**
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



Full adder (naïve)

Sum and Carry:

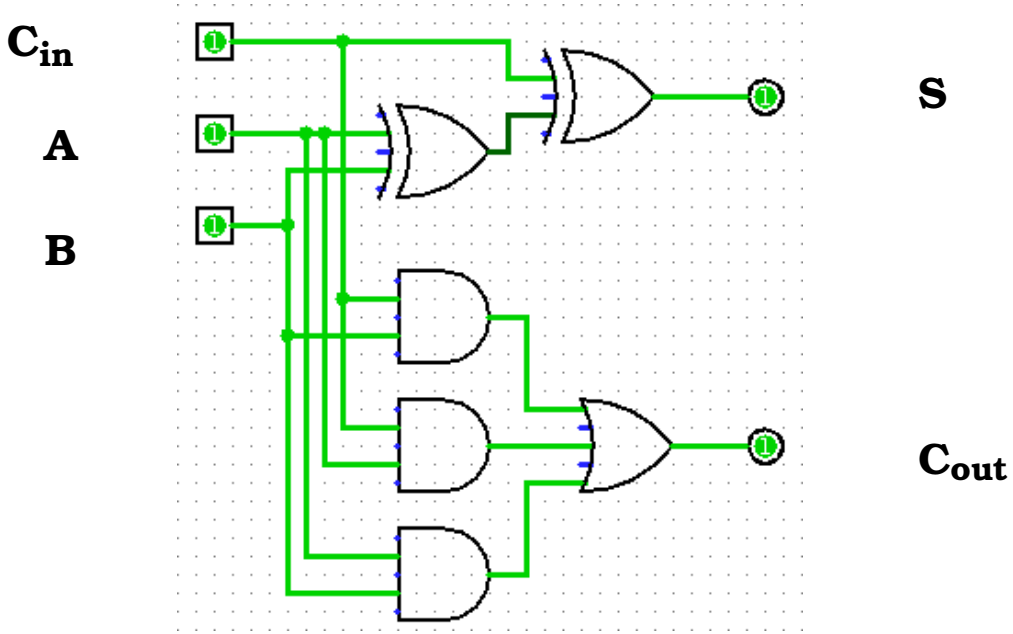
C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	**0**	**1**	**1**	**0**
1	1	0	1	0
1	1	1	1	1



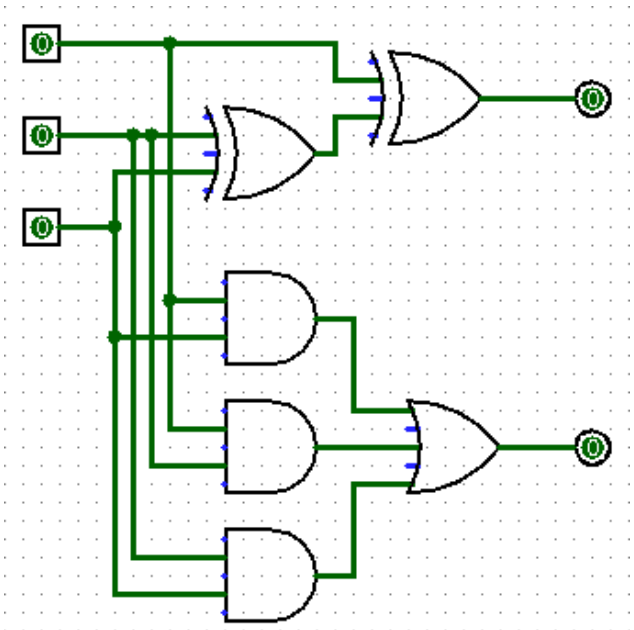
Full adder (naïve)

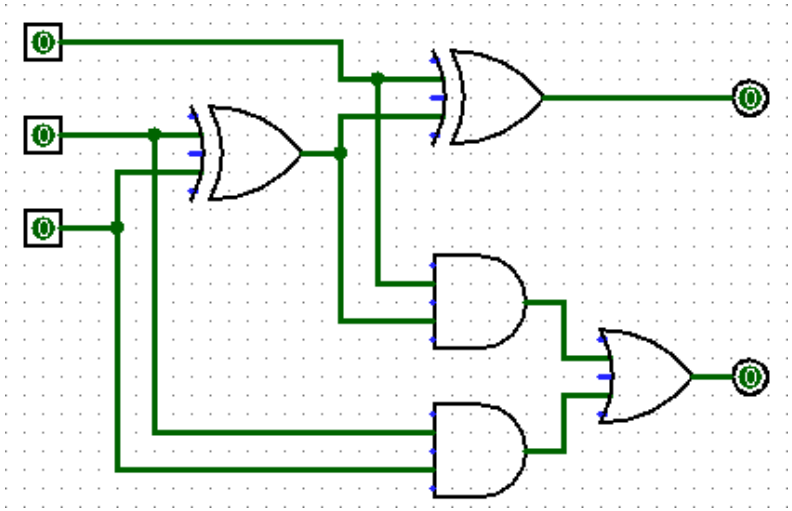
Sum and Carry:

C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	**1**	**1**	**1**	**1**

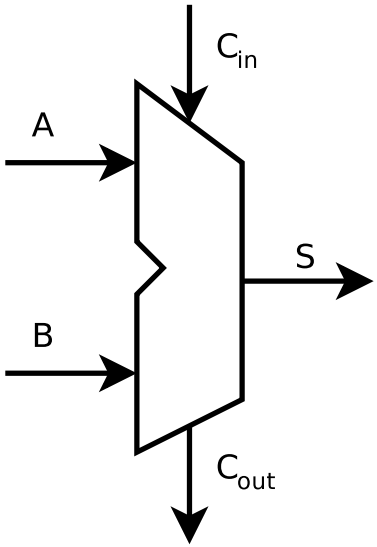


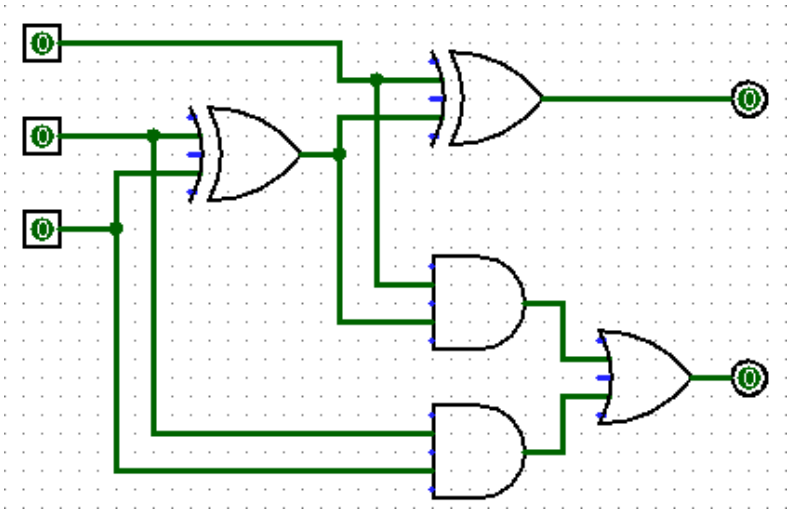
Full adder

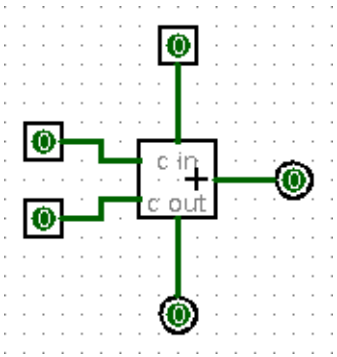




Full adder symbols

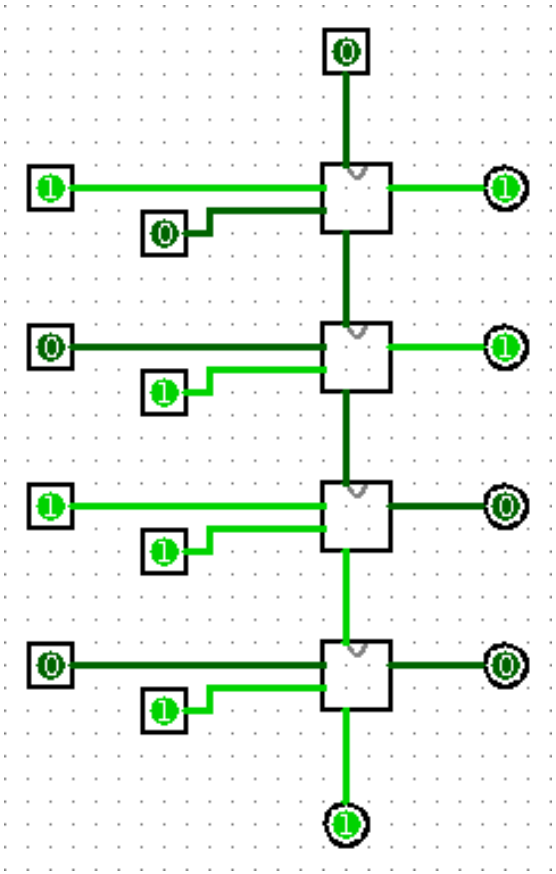


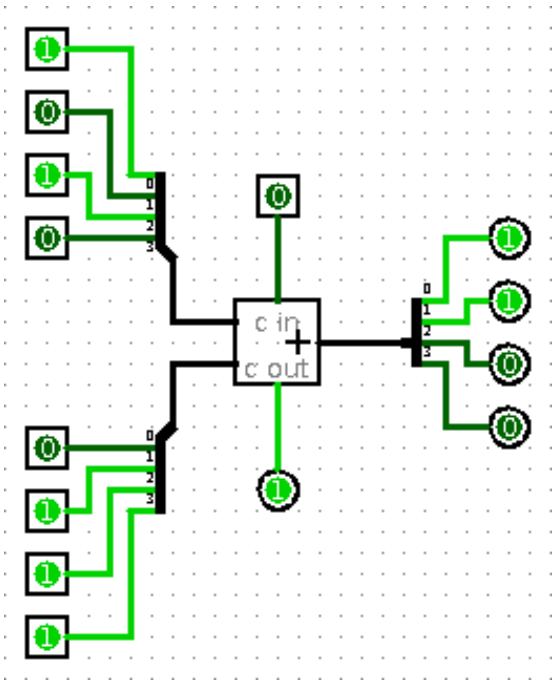




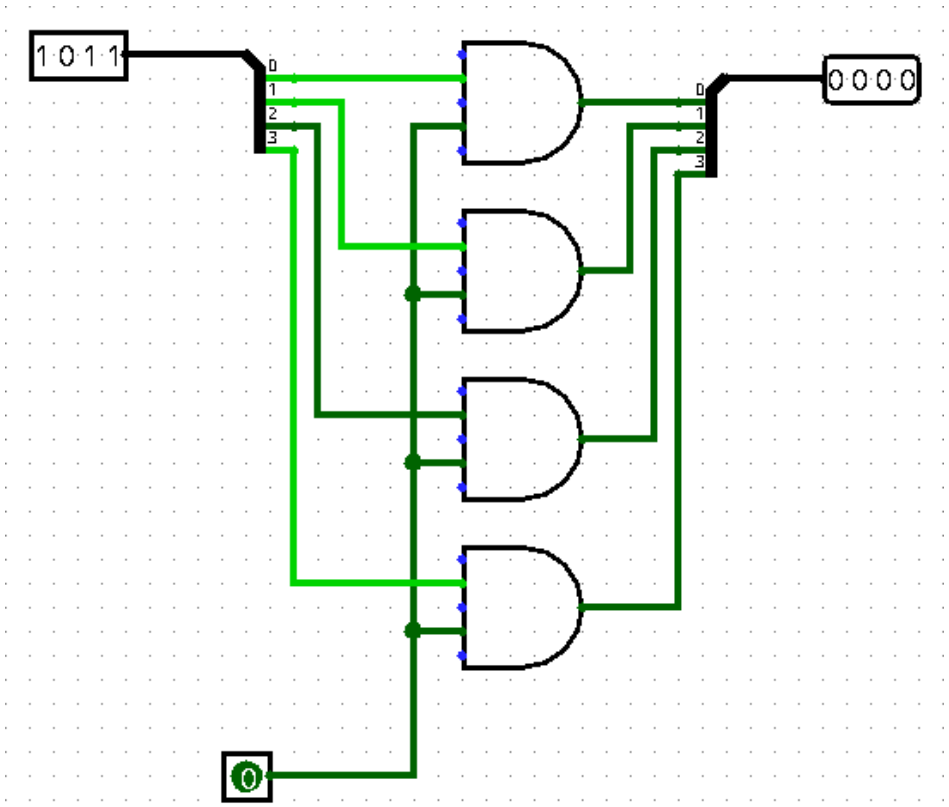
4-bit adder

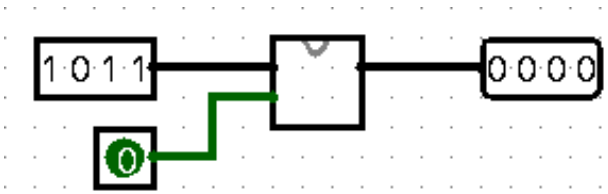
$$0101_2 + 1110_2 = 1\,0011_2 \qquad 5_{16} + \mathrm{E}_{16} = 13_{16} \qquad 5_{10} + 14_{10} = 19_{10}$$



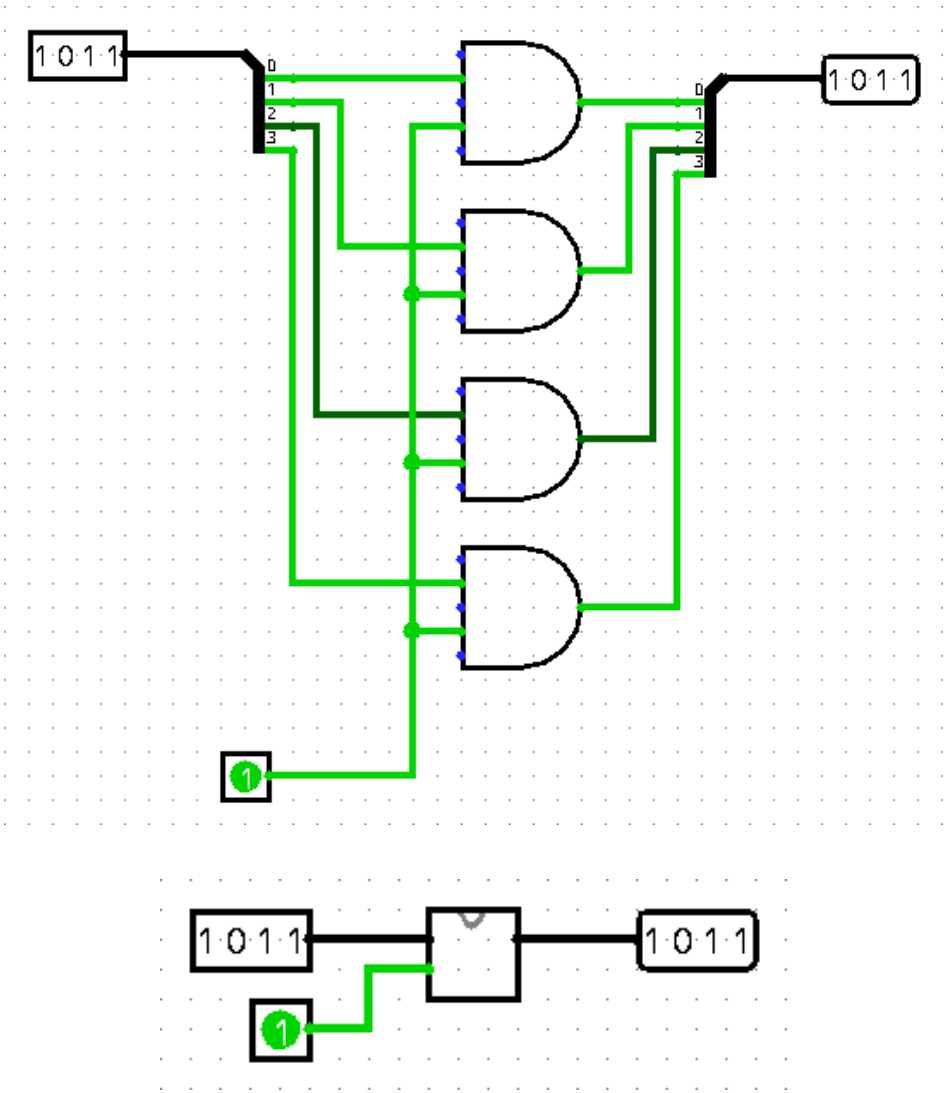


Single digit multiplier



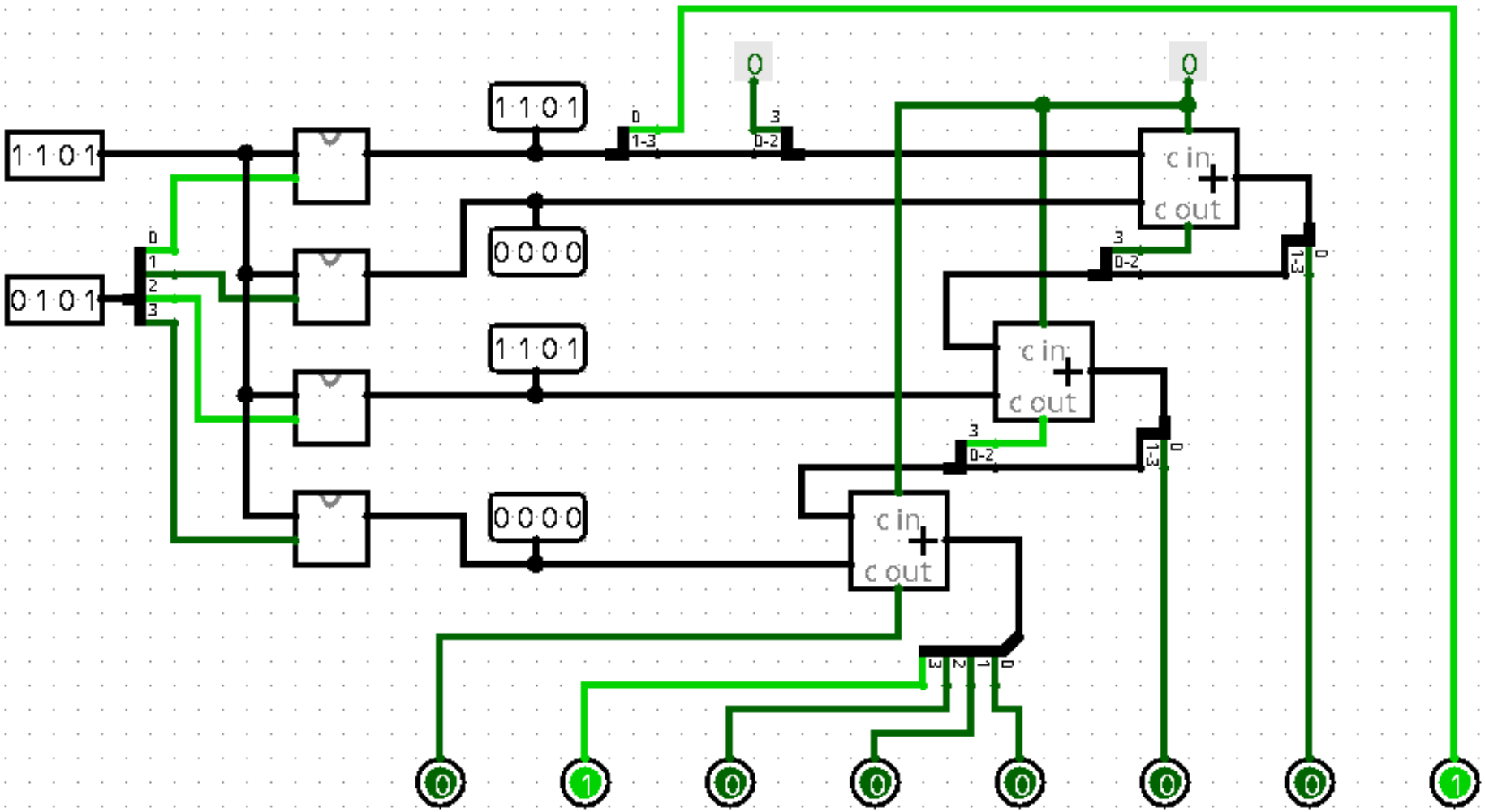


Single digit multiplier



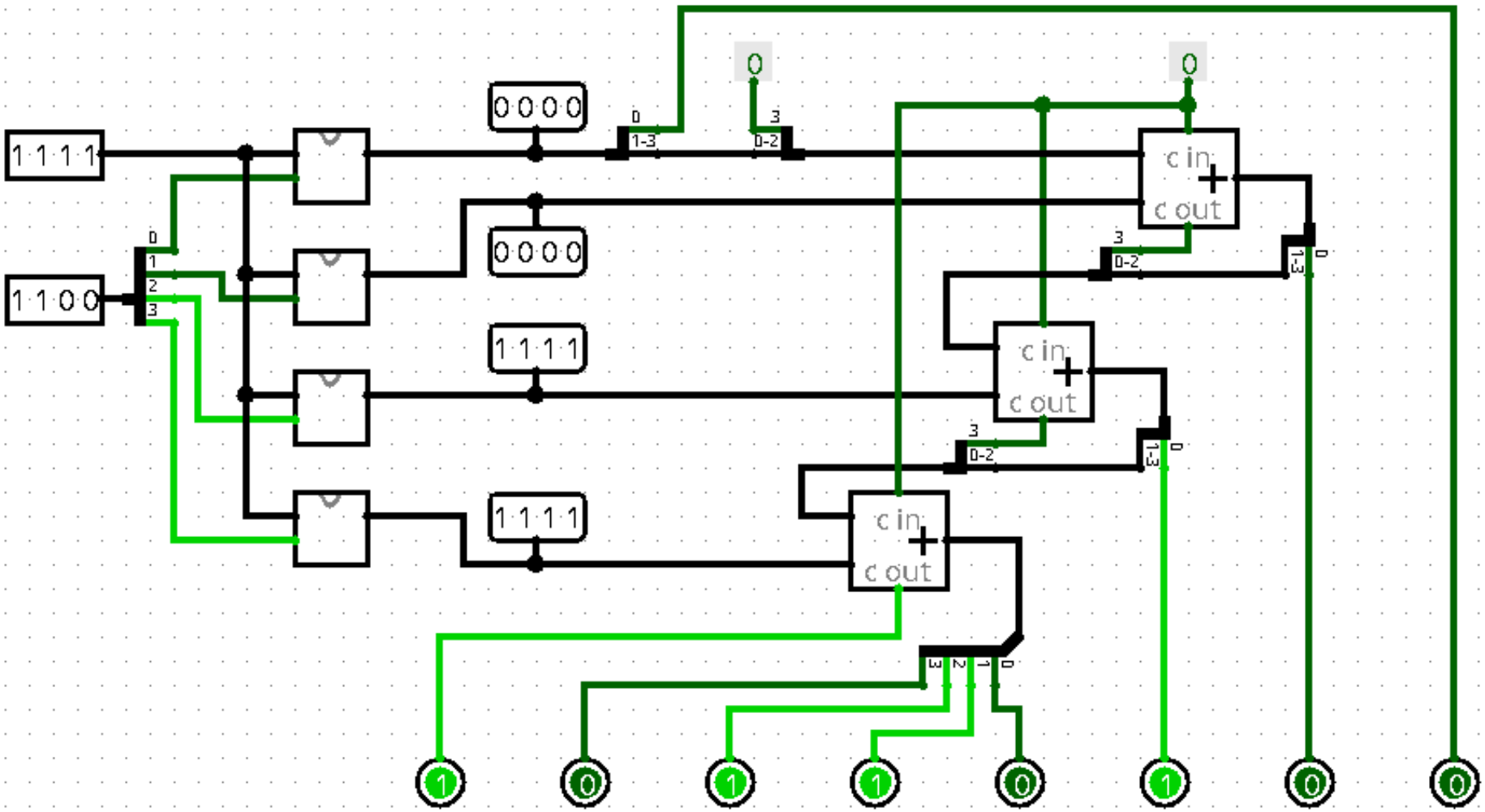
4-bit naïve multiplier

$1101_2 \times 0101_2 = 0100\,0001_2 \quad D_{16} \times 5_{16} = 41_{16} \quad 13_{10} \times 5_{10} = 65_{10}$



4-bit naïve multiplier

$$1111_2 \times 1100_2 = 1011\,0100_2 \quad \mathrm{F}_{16} \times \mathrm{C}_{16} = \mathrm{B4}_{16} \quad 15_{10} \times 12_{10} = 180_{10}$$



Subtraction

Subtracting smaller from larger is easy:

$$1101_2 - 100_2 = 1001_2$$

$$1101_2 - 11_2 = 11\overset{1}{0}1_2 - 11_2 = 1010_2$$

$$10_2 - 11_2 = ???$$

What happens if we add?

$$\begin{array}{cccccc} & & 0 & 0 & 0 & 1 \\ + & & 1 & 1 & 1 & 1 \\ \hline & 1 & 0 & 0 & 0 & 0 \end{array}$$

Negative numbers

Add large numbers so that a carry is generated:

$$\begin{array}{cccccc} & & 0 & 0 & 0 & 1 \\ + & & 1 & 1 & 1 & 1 \\ \hline & 1 & 0 & 0 & 0 & 0 \end{array}$$

i.e.:

$$1111_2 + 1_2 = 0_2$$

but we know that

$$(-1) + 1 = 0$$

Can we say that $1111_2 = -1_2$ in some sense?

Two's complement

Adding 1111 means: $1111_2 = 1\,0000_2 - 1 = 2^4 - 1$

Disregarding the carry digit means subtracting $1\,0000_2 = 2^4$

Thus

$$\begin{aligned} 1 + 1111_2 \text{ (disregarding carry)} &= 1 + (1\,0000_2 - 1) - 1\,0000_2 \\ &= 1 - 1 \\ &= 0 \end{aligned}$$

Let's see – it works for any number of N bits:

$$a + \underbrace{(2^N - b)}_{\text{2's complement of } b} - 2^N = a - b$$

Take home messages

- Binary is a natural positional number system for digital computations; simple algorithms exist to convert numbers between binary and other number systems, but systems with bases 8 and 16 are the most convenient to represent binary numbers.
- Perl one-liners can be useful for number conversions;
- Not every conversion can be made exactly!
- Not always a two-state signal carries 1 bit of information!
- We can construct digital logic circuits to perform binary arithmetic (addition, subtraction, multiplication, etc.)

References I

Mitašiūnas, Antanas (2016). *Kompiuterių architektūra*. URL: http://www.mif.vu.lt/katedros/cs/Asmen/Kompiuteriu%20architektura.pdf.