Integers

Leon Tabak
Basis of Software
Beihua University

18 April 2019

```c
#include <stdio.h>

int main( int argc, char** argv ) {
    printf( "%s\n", "Good day!" );
} // main( int, char** )
```

0. A first C program.

```c
#include <stdio.h>

int main( int argc, char** argv ) {
    int china = 1300000000; // population of China
    int india = 1300000000; // population of India

    int sum = china + india; // lower bound on number of Asians

    printf( "Number of Asians is at least %12d\n", sum );
} // main( int, char** )
```

1. Integer overflow.

```c
#include <stdio.h>

int main( int argc, char** argv ) {
    int a = 5;
    int b = 10;

    int sum = a | b;

    printf( "%2d + %2d = %2d\n", a, b, sum );
} // main( int, char** )
```

2. Bitwise disjunction (or).

```c
#include <stdio.h>

int main( int argc, char** argv ) {
    int a = 5;
    int b = 10;

    printf( "%2d & %2d = %2d\n", a, b, a & b );
    printf( "%2d & %2d = %2d\n", a, a, a & a );
    printf( "%2d & %2d = %2d\n", b, b, b & b );
} // main( int, char** )
```

3. Bitwise conjunction (and)

```c
#include <stdio.h>

int main( int argc, char** argv ) {
    int a = 6;
    int b = 8;

    printf( "%2d & ~%2d = %2d\n", a, a, (a & ~a) );
    printf( "%2d & ~%2d = %2d\n", b, b, (b & ~b) );

    printf( "%2d + (~%2d + 1) = %2d\n", b, a, (b + (~a + 1)) );
} // main( int, char** )
```

4. Bitwise not (logical complement)

```c
#include <stdio.h>

int main( int argc, char** argv ) {
    int a = 9;
    int b = 96;

    printf( "%2d << 1 = %2d\n", a, (a << 1) );
    printf( "%2d << 2 = %2d\n", a, (a << 2) );
    printf( "%2d << 3 = %2d\n", a, (a << 3) );

    printf( "\n" );

    printf( "%2d >> 1 = %2d\n", b, (b >> 1) );
    printf( "%2d >> 2 = %2d\n", b, (b >> 2) );
    printf( "%2d >> 3 = %2d\n", b, (b >> 3) );
} // main( int, char** )
```

5. Left and right shifts.

$$
\begin{aligned}
2019_{10} &= 2 \cdot 10^3 + 0 \cdot 10^2 + 1 \cdot 10^1 + 9 \cdot 10^0 \\
&= 2 \cdot 1000 + 0 \cdot 100 + 1 \cdot 10 + 9 \cdot 1 \\
1969_{10} &= 1 \cdot 1000 + 9 \cdot 100 + 6 \cdot 10 + 9 \cdot 1 \\
1863_{10} &= 1 \cdot 1000 + 8 \cdot 100 + 6 \cdot 10 + 3 \cdot 1 \\
1776_{10} &= 1 \cdot 1000 + 7 \cdot 100 + 7 \cdot 10 + 6 \cdot 1
\end{aligned}
$$

We can represent a number as a sum of multiples of powers of 10.

6. Base 10 place values

$$
\begin{aligned}
2019_{10} &= 11111100011_2 \\
&= 1024 + 512 + 256 + 128 + 64 + 32 + 2 + 1 \\
1969_{10} &= 11110110001_2 \\
&= 1024 + 512 + 256 + 128 + 32 + 16 + 1 \\
1863_{10} &= 11101000111_2 \\
&= 1024 + 512 + 256 + 64 + 4 + 2 + 1 \\
1776_{10} &= 11011110000_2 \\
&= 1024 + 512 + 128 + 64 + 32 + 16
\end{aligned}
$$

We can represent a number as a sum of multiples of powers of 2.

7. Base 2 place values.

2^n	8^n	16^n	Decimal	Binary
2^0	8^0	16^0	1	0000 0000 0000 000**1**
2^1			2	0000 0000 0000 00**1**0
2^2			4	0000 0000 0000 0**1**00
2^3	8^1		8	0000 0000 0000 **1**000
2^4		16^1	16	0000 0000 000**1** 0000
2^5			32	0000 0000 00**1**0 0000
2^6	8^2		64	0000 0000 0**1**00 0000
2^7			128	0000 0000 **1**000 0000
2^8		16^2	256	0000 000**1** 0000 0000
2^9	8^3		512	0000 00**1**0 0000 0000
2^{10}			1024	0000 0**1**00 0000 0000
2^{11}			2048	0000 **1**000 0000 0000
2^{12}	8^4	16^3	4096	000**1** 0000 0000 0000
2^{13}			8192	00**1**0 0000 0000 0000
2^{14}			16384	0**1**00 0000 0000 0000
2^{15}	8^5		32768	**1**000 0000 0000 0000
2^{16}		16^4	65536	**1** 0000 0000 0000 0000

8. Powers of 2.

Decimal	Binary	Decimal	Binary
0	0000	8	1000
1	0001	9	1001
2	0010	10	1010
3	0011	11	1011
4	0100	12	1100
5	0101	13	1101
6	0110	14	1110
7	0111	15	1111

9. Decimal and binary representations.

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

10. Decimal, binary, octal, and hexadecimal representations.

Dividend	**Divisor**	**Quotient**	**Remainder**			
2019	2	1009	1	$2 \times 1009 + 1$	=	2019
1009	2	504	1	$2 \times 504 + 1$	=	1009
504	2	252	0	$2 \times 252 + 0$	=	504
252	2	126	0	$2 \times 126 + 0$	=	252
126	2	63	0	$2 \times 63 + 0$	=	126
63	2	31	1	$2 \times 31 + 1$	=	63
31	2	15	1	$2 \times 15 + 1$	=	31
15	2	7	1	$2 \times 7 + 1$	=	15
7	2	3	1	$2 \times 3 + 1$	=	7
3	2	1	1	$2 \times 1 + 1$	=	3
1	2	0	1	$2 \times 0 + 1$	=	1

Read the remainders from bottom to top:

$$2019_{10} = 11111100011_2$$

11. Translation of decimal 2019 to binary representation.

$$\begin{aligned}
2019_{10} &= 11111100011_2 \\
11111100011_2 &= 1 \times 1024 \\
&+ 1 \times 512 \\
&+ 1 \times 256 \\
&+ 1 \times 128 \\
&+ 1 \times 64 \\
&+ 1 \times 32 \\
&+ 0 \times 16 \\
&+ 0 \times 8 \\
&+ 0 \times 4 \\
&+ 1 \times 2 \\
&+ 1 \times 1 \\
&= 2019_{10}
\end{aligned}$$

12. Sum of powers of 2 that equal 2019.

Dividend	**Divisor**	**Quotient**	**Remainder**			
1969	2	984	1	$2 \times 984 + 1$	=	1969
984	2	492	0	$2 \times 492 + 0$	=	984
492	2	246	0	$2 \times 246 + 0$	=	492
246	2	123	0	$2 \times 123 + 0$	=	246
123	2	61	1	$2 \times 61 + 1$	=	123
61	2	30	1	$2 \times 30 + 1$	=	61
30	2	15	0	$2 \times 15 + 0$	=	30
15	2	7	1	$2 \times 7 + 1$	=	15
7	2	3	1	$2 \times 3 + 1$	=	7
3	2	1	1	$2 \times 1 + 1$	=	3
1	2	0	1	$2 \times 0 + 1$	=	1

Read the remainders from bottom to top:

$$1969_{10} = 11110110001_2$$

13. Translation of decimal 1969 to binary representation.

$$\begin{aligned}
1969_{10} &= 11110110001_2 \\
11110110001_2 &= 1 \times 1024 \\
&+ 1 \times 512 \\
&+ 1 \times 256 \\
&+ 1 \times 128 \\
&+ 0 \times 64 \\
&+ 1 \times 32 \\
&+ 1 \times 16 \\
&+ 0 \times 8 \\
&+ 0 \times 4 \\
&+ 0 \times 2 \\
&+ 1 \times 1 \\
&= 1969_{10}
\end{aligned}$$

14. Sum of powers of 2 that equal 1969.

Dividend	**Divisor**	**Quotient**	**Remainder**			
1863	2	931	1	$2 \times 931 + 1$	=	1863
931	2	465	1	$2 \times 465 + 1$	=	931
465	2	232	1	$2 \times 232 + 1$	=	465
232	2	116	0	$2 \times 116 + 0$	=	232
116	2	58	0	$2 \times 58 + 0$	=	116
58	2	29	0	$2 \times 29 + 0$	=	58
29	2	14	1	$2 \times 14 + 1$	=	29
14	2	7	0	$2 \times 7 + 0$	=	14
7	2	3	1	$2 \times 3 + 1$	=	7
3	2	1	1	$2 \times 1 + 1$	=	3
1	2	0	1	$2 \times 0 + 1$	=	1

Read the remainders from bottom to top:

$$1863_{10} = 11101000111_2$$

15. Translation of decimal 1863 to binary representation.

$$\begin{aligned}
1863_{10} &= 11101000111_2 \\
11101000111_2 &= 1 \times 1024 \\
&+ 1 \times 512 \\
&+ 1 \times 256 \\
&+ 0 \times 128 \\
&+ 1 \times 64 \\
&+ 0 \times 32 \\
&+ 0 \times 16 \\
&+ 0 \times 8 \\
&+ 1 \times 4 \\
&+ 1 \times 2 \\
&+ 1 \times 1 \\
&= 1863_{10}
\end{aligned}$$

16. Sum of powers of 2 that equal 1863.

Dividend	**Divisor**	**Quotient**	**Remainder**			
1776	2	888	0	$2 \times 888 + 0$	=	1776
888	2	444	0	$2 \times 444 + 0$	=	888
444	2	222	0	$2 \times 222 + 0$	=	444
222	2	111	0	$2 \times 111 + 0$	=	222
111	2	55	1	$2 \times 55 + 1$	=	111
55	2	27	1	$2 \times 27 + 1$	=	55
27	2	13	1	$2 \times 13 + 1$	=	27
13	2	6	1	$2 \times 6 + 1$	=	13
6	2	3	0	$2 \times 3 + 0$	=	6
3	2	1	1	$2 \times 1 + 1$	=	3
1	2	0	1	$2 \times 0 + 1$	=	1

Read the remainders from bottom to top:

$$1776_{10} = 11011110000_2$$

17. Translation of decimal 1776 to binary representation.

$$\begin{aligned}
1776_{10} &= 11011110000_2 \\
11011110000_2 &= 1 \times 1024 \\
&+ 1 \times 512 \\
&+ 0 \times 256 \\
&+ 1 \times 128 \\
&+ 1 \times 64 \\
&+ 1 \times 32 \\
&+ 1 \times 16 \\
&+ 0 \times 8 \\
&+ 0 \times 4 \\
&+ 0 \times 2 \\
&+ 0 \times 1 \\
&= 1776_{10}
\end{aligned}$$

18. Sum of powers of 2 that equal 1776.

A	B	A & B (A AND B)
1	1	1
1	0	0
0	1	0
0	0	0

```
int a = 14; // binary representation: 1110
int b =  5; // binary representation: 0101
```

19. Bitwise and (conjunction).

A	B	A | B (A OR B)
1	1	1
1	0	1
0	1	1
0	0	0

```
int a = 14; // binary representation: 1110
int b =  5; // binary representation: 0101
```

20. Bitwise or (disjunction).

A	B	A ∧ B (A XOR B)
1	1	0
1	0	1
0	1	1
0	0	0

```
int a = 14; // binary representation: 1110
int b =  5; // binary representation: 0101
```

21. Bitwise exclusive or.

A	~A (NOT A)
1	0
0	1

```
int a = 10; // binary representation: 1010
int b = ~a; // binary representation: 0101
printf( "b = ~a = %4d\n", b ); // will print "b = ~a = 5"
```

22. Bitwise not.

$$\begin{aligned} 37 - 25 &= 37 - 25 + 100 - 100 \\ &= 37 + (100 - 25) - 100 \\ &= 37 + ((99 - 25) + 1) - 100 \\ &= 37 + 74 + 1 - 100 \\ &= 111 + 1 - 100 \\ &= 112 - 100 \\ &= 12 \end{aligned}$$

Let's agree to operate only on two digit integers whose magnitudes are ≤ 50.

$$\begin{aligned} 37 - 25 &\Rightarrow 37 + ((99 - 25) + 1) \\ &\Rightarrow 37 + 74 + 1 \\ &\Rightarrow 1\ 12 \quad \text{(Throw away digit in hundreds place.)} \\ &\Rightarrow 12 \end{aligned}$$

Although there is still a subtraction in this algorithm, it is an easy subtraction—there is no "borrowing" in a difference like $99 - 25$. That difference gives us a "complement." Once we have the complement of 25, we can find the difference $37 - 25$ using just addition.

23. Finding a difference using only addition.

Rules for addition of binary integers:

$$
\begin{aligned}
&0 + 0 = 0\\
&0 + 1 = 1\\
&1 + 0 = 1\\
&1 + 1 = 0 \text{ and "carry" 1 to the left}
\end{aligned}
$$

Example:

$$
\begin{array}{rr}
 & 0101\\
+ & 1101\\
\hline
 & 1\ 0010
\end{array}
$$

24. Addition of binary integers.

$$
\begin{aligned}
0101_2 - 0011_2 &= 0101_2 - 0011_2 + 10000_2 - 10000_2\\
&= 00101 + 10000 - 00011 - 10000\\
&= 00101 + 01111 - 00011 + 00001 - 10000\\
&= 00101 + 01100 + 00001 - 10000\\
&= 00101 + 01101 - 10000\\
&= 10010 - 10000\\
&= 00010
\end{aligned}
$$

Let's agree to operate only on four digit binary integers.
Again, finding a complement requires only an easy subtraction—no "borrowing."

$$
\begin{aligned}
0101 - 0011 &\Rightarrow 0101 + 1111 - 0011 + 1\\
&\Rightarrow 0101 + 1101\\
&\Rightarrow 1\ 0010 \quad \text{Throw away digit in sixteens place.}\\
&\Rightarrow 0010
\end{aligned}
$$

25. Finding a difference using only addition.