A **word** on the ARM Cortex M will have 32 bits. Consider an unsigned number with 32 bits, where each bit $b_{31},...,b_0$ is binary and has the value 1 or 0. If a 32-bit number is used to represent an unsigned integer, then the value of the number is

$N = 2^{31} \cdot b_{31} + 2^{30} \cdot b_{30} + ... + 2 \cdot b_1 + b_0 = \text{sum}(2^i \cdot b_i)$ for i=0 to 31

There are 2^{32} different unsigned 32-bit numbers. The smallest unsigned 32-bit number is 0 and the largest is 2^{32}-1. This range is 0 to about 4 billion.

A **halfword** or **double byte** contains 16 bits, where each bit $b_{15},...,b_0$ is binary and has the value 1 or 0, as shown in Figure 2.4.

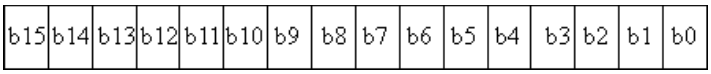

Figure 2.4

Similar to the unsigned algorithm, we can use the basis to convert a decimal number into signed binary. We will work through the algorithm with the example of converting –100 to 8-bit binary: We start with the most significant bit (in this case –128) and decide do we need to include it to make –100? Yes (without –128, we would be unable to add the other basis elements together to get any negative result), so we set bit 7 and subtract the basis element from our value. Our new value equals –100 minus –128, which is 28. We go the next largest basis element, 64 and ask, “do we need it?” We do not need 64 to generate our 28, so bit 6 is zero. Next we go the next basis element, 32 and ask, “do we need it?” We do not need 32 to generate our 28, so bit 5 is zero. Now we need the basis element 16, so we set bit 4, and subtract 16 from our number 28 (28-16=12). Continuing along, we need basis elements 8 and 4 but not 2, 1. Putting it together we get 10011100_2 (which means -128+16+8+4).

Number	Basis	Need it	bit	Operation
-100	-128	yes	bit 7=1	subtract -100 - -128
28	64	no	bit 6=0	none
28	32	no	bit 5=0	none

28	16	yes	bit 4=1	subtract 28-16
12	8	yes	bit 3=1	subtract 12-8
4	4	yes	bit 2=1	subtract 4-4
0	2	no	bit 1=0	none
0	1	no	bit 0=0	none

Table 2.6. Example conversion from decimal to signed 8-bit binary.

A second way to convert negative numbers into binary is to first convert them into unsigned binary, then do a two's complement negate. For example, we earlier found that +100 is 01100100_2. The two's complement negate is a two-step process. First we do a logic complement (flip all bits) to get 10011011_2. Then add one to the result to get 10011100_2.

A third way to convert negative numbers into binary uses the number wheel. Let *n* be the number of bits in the binary representation. We specify **precision**, *M*=2^*n*, as the number of distinct values that can be represented. To convert negative numbers into binary is to first add *M* to the number, then convert the unsigned result to binary using the unsigned method. This works because binary numbers with a finite *n* are like the minute-hand on a clock. If we add 60 minutes, the minute-hand is in the same position. Similarly if we add *M* to or subtract *M* from an *n*-bit number, we go around the number wheel and arrive at the same place. This is one of the beautiful properties of 2's complement: unsigned and signed addition/subtraction are same operation. In this example we have an 8-bit number so the precision is 256. So, first we add 256 to the number, then convert the unsigned result to binary using the unsigned method. For example, to find –100, we add 256 plus –100 to get 156. Then we convert 156 to binary resulting in 10011100_2. This method works because in 8-bit binary math adding 256 to number does not change the value. E.g., 256-100 has the same 8-bit binary value as –100.

When dealing with numbers on the computer, it will be convenient to memorize some **Powers of 2** as shown in Table 2.7.

exponent	decimal
2^0	1
2^1	2
2^2	4
2^3	8
2^4	16
2^5	32
2^6	64
2^7	128

2^8	256
2^9	512
2^{10}	1024 about a thousand
2^{11}	2048
2^{12}	4096
2^{13}	8192
2^{14}	16384
2^{15}	32768
2^{16}	65536
2^{20}	about a million
2^{30}	about a billion
2^{40}	about a trillion

Table 2.7. Some powers of two that will be useful to memorize.

CHECKPOINT 2.16

Give the representations of -54 in 8-bit binary and hexadecimal.

Hide Answer

Combine signed binary basis elements to create the desired value. -54 = -128+64+8+2 = 11001010_2 = 0xCA.

CHECKPOINT 2.17

Why can't you represent the number 150 using 8-bit signed binary?

Hide Answer

Because the range of 8-bit signed numbers is -128 to +127.

CHECKPOINT 2.18

Use Table 2.7 to determine the approximate value of 2^{32}?

Hide Answer

2^{32} is 4 times 2^{30}, and 2^{30} is about a billion, so 2^{32} is about 4 billion.

CHECKPOINT 2.19

Convert the 16-bit binary number 0010000001101010_2 to unsigned decimal.

Hide Answer

8192+64+32+8+2=8298.

CHECKPOINT 2.20

Convert the 16-bit hex number 0x1234 to unsigned decimal.

Hide Answer

1*4096+2*256+3*16+4=4660.