Overflow

Now consider the following example:

Example 3

Add 0 1 1 0 1 1 1 0 and 1 1 0 1 1 1 1 0 (using 8 bits)

```
            0 1 1 0 1 1 1 0
      +
            1 1 0 1 1 1 1 0
9th bit  1) 1 1 1 1 1 1        ← carry
         1) 0 1 0 0 1 1 0 0    ← sum
```

This addition has generated a 9th bit. The 8 bits of the answer are 0 1 0 0 1 1 0 0 – this gives the denary value (64 + 8 + 4) of 76 which is incorrect because the denary value of the addition is 110 + 222 = 332.

The maximum denary value of an 8-bit binary number is 255 (which is $2^8 - 1$). The generation of a 9th bit is a clear indication that the sum has exceeded this value. This is known as an **overflow error** and in this case is an indication that a number is too big to be stored in the computer using 8 bits.

The greater the number of bits which can be used to represent a number then the larger the number that can be stored. For example, a 16-bit register would allow a maximum denary value of 65 535 (i.e. $2^{16} - 1$) to be stored, a 32-bit register would allow a maximum denary value of 4 294 967 295 (i.e. $2^{32} - 1$), and so on.

Activity 1.10

1. Convert the following pairs of denary numbers to 8-bit binary numbers and then add the binary numbers. Comment on your answers in each case:
   - **a** 89 + 175 **b** 168 + 99 **c** 88 + 215
2. Carry out the following 16-bit binary additions and comment on your answers:
   - **a** 0111 1111 1111 0001 + 0101 1111 0011 1001
   - **b** 1110 1110 0000 1011 + 1111 1101 1101 1001

1.1.5 Logical binary shifts

Computers can carry out a **logical shift** on a sequence of binary numbers. The logical shift means moving the binary number to the ***left*** or to the ***right***. Each shift ***left*** is equivalent to ***multiplying*** the binary number by 2 and each shift ***right*** is equivalent to ***dividing*** the binary number by 2.

As bits are shifted, any empty positions are replaced with a zero – see examples below. There is clearly a limit to the number of shifts which can be carried out if the binary number is stored in an 8-bit register. Eventually after a number of shifts the register would only contain zeros. For example, if we shift 01110000 (denary value 112) five places left (the equivalent to multiplying by 2^5, i.e. 32), in an 8-bit register we would end up with 00000000. This makes it seem as though 112 × 32 = 0! This would result in the generation of an error message.

Example 1

The denary number 21 is 00010101 in binary. If we put this into an 8-bit register:

128	64	32	16	8	4	2	1
0	0	0	1	0	1	0	1

The left-most bit is often referred to as the MOST SIGNIFICANT BIT

If we now shift the bits in this register one place to the left, we obtain:

128	64	32	16	8	4	2	1
0	0	1	0	1	0	1	0

The left-most bit is now lost following a left shift

Note how the empty right-most bit position is now filled with a 0

The value of the binary bits is now 21×2^1 i.e. 42. We can see this is correct if we calculate the denary value of the new binary number 101010 (i.e. 32 + 8 + 2).

Suppose we now shift the original number two places left:

128	64	32	16	8	4	2	1
0	1	0	1	0	1	0	0

The binary number 1010100 is 84 in denary – this is 21×2^2.

And now suppose we shift the original number three places left:

1	0	1	0	1	0	0	0

The binary number 10101000 is 168 in denary – this is 21×2^3.

So, let us consider what happens if we shift the original binary number 00010101 four places left:

128	64	32	16	8	4	2	1
0	1	0	1	0	0	0	0

Losing 1 bit following a shift operation will cause an error

The left-most 1-bit has been lost. In our 8-bit register the result of 21×2^4 is 80 which is clearly incorrect. This error is because we have exceeded the maximum number of left shifts possible using this register.

Example 2

The denary number 200 is 11001000 in binary. Putting this into an 8-bit register gives:

128	64	32	16	8	4	2	1
1	1	0	0	1	0	0	0

The right-most bit is often referred to as the LEAST SIGNIFICANT BIT

If we now shift the bits in this register one place to the right:

128	64	32	16	8	4	2	1
0	1	1	0	0	1	0	0

Note how the left-most bit position is now filled with a 0

The value of the binary bits is now $200 \div 2^1$ i.e. 100. We can see this is correct by converting the new binary number 01100100 to denary (64 + 32 + 4).

Suppose we now shift the original number two places to the right:

128	64	32	16	8	4	2	1
0	0	1	1	0	0	1	0

The binary number 00110010 is 50 in denary – this is $200 \div 2^2$.

And suppose we now shift the original number three places to the right:

Notice the 1-bit from the right-most bit position is now lost causing an error

128	64	32	16	8	4	2	1
0	0	0	1	1	0	0	1

The binary number 00011001 is 25 in denary – this is $200 \div 2^3$.

Now let us consider what happens if we shift four places right:

128	64	32	16	8	4	2	1
0	0	0	0	1	1	0	0

The right-most 1-bit has been lost. In our 8-bit register the result of $200 \div 2^4$ is 12, which is clearly incorrect. This error is because we have therefore exceeded the maximum number of right shifts possible using this 8-bit register.

? Example 3

a Write 24 as an 8-bit register.
b Show the result of a logical shift 2 places to the left.
c Show the result of a logical shift 3 places to the right.

a

128	64	32	16	8	4	2	1
0	0	0	1	1	0	0	0

b

128	64	32	16	8	4	2	1	
0	1	1	0	0	0	0	0	⇦ $24 \times 2^2 = 96$

c

128	64	32	16	8	4	2	1	
0	0	0	0	0	0	1	1	⇦ $24 \div 2^3 = 3$

? Example 4

a Convert 19 and 17 into binary.
b Carry out the binary addition of the two numbers.
c Shift your result from part b two places left and comment on the result.
d Shift your result from part b three places right and comment on the result.

a

128	64	32	16	8	4	2	1	
0	0	0	1	0	0	1	1	19
0	0	0	1	0	0	0	1	17