EE 2720

Handout #4

EE 2720

- Binary Multiplication

In the multiplication you have two numbers X and Y; X is called the multiplicand, Y is called the multiplier and you are interested in computing their product P where P is $P = X \times Y$. As you already know, you can multiply two numbers in decimal by adding a list of shifted multiplicands computed according to the digits of the multiplier.

The same techique can be applied to get the product of two unsigned binary numbers. Here, obtaining the shifted multiplicands is very easy in binary because the only possible values of the multiplier bits are 0 and 1. An example follows on the next page.

Example: Perform the unsigned binary multiplication with multiplicand $X = 1011_2 = 11_{10}$ and multiplier $Y = 1101_2 = 13_{10}$

Answer: I will first show the multiplication in decimal. It is

```
  11   multiplicand
x 13   multiplier
----
  33
 11    shifted multiplicands
----
 143   product
```

I will now show it in binary

```
    1011   multiplicand
  x 1101   multiplier
  ------
    1011
   0000
  1011     shifted multiplicands
 1011
 --------
 10001111  product = 143_10
```

An easier way of doing the above is shown on the next page.

```
    1011   multiplicand
  x 1101   multiplier
  ------
    1011
+   0000
  ------
   01011
+  1011
  ------
  110111
+ 1011
  -------
 1000111   product = 143_10.
```

Note: When we multiply an n-bit unsigned number by an m-bit unsigned number, their product requires at most $n+m$ bits for its representation.

Just see that the maximum value of an n-bit unsigned number is 2^n-1 while the maximum value of an m-bit unsigned number is 2^m-1. So the maximum value of the product of an n-bit by an m-bit unsigned number is $(2^n-1)(2^m-1) =$
$= 2^{n+m} - 2^m - 2^n + 1 < 2^{n+m} - 1$.

Just observe that the maximum value of an $(n+m)$-bit unsigned number is $2^{n+m}-1$.

Note: If we want to multiply signed binary numbers we can do this using the unsigned binary multiplication as follows: Perform the unsigned multiplication of the magnitudes of the two numbers and make the product positive if multiplicand and multiplier have the same sign, otherwise make the product negative if multiplicand and multiplier have different signs. This procedure is very convenient for the signed-magnitude system. However, this procedure is not very convenient for the two's-complement system because here we might have to obtain (sometimes) the two's-complement of either the multiplicand or the multiplier or both and sometimes obtain the two's-complement of the product. These are nontrivial operations.

A direct way of multiplying binary numbers in the two's-complement system follows:

- Binary Multiplication for two's-complement numbers.

→ Go to next page.

The procedure for multiplying two binary numbers in the two's-complement system is similar to the one for multiplying unsigned numbers except:

1. Each shifted multiplicand must be sign-extended so its value is preserved; (we said that in Handout# 2; see bottom of page 16 of Handout#2).
2. The shifted multiplicand that corresponds to the sign bit of the multiplier must be negated (its two's-complement must be obtained) before it is added to the partial product. This is due to the fact that in the two's-complement system the sign bit has a negative weight; (see lemma on page 13 of Handout 2).

An example of multiplying in the two's-complement system follows

Example: Perform the signed two's-complement binary multiplication with multiplicand $X = 1010_2 = -6_{10}$ and multiplier $Y = 1011_2 = -5_{10}$.

Answer: On next page ⟶

```
    1010   multiplicand
  x 1011   multiplier
  ------
    111010   sign extended multiplicand
+ 11010      shifted and sign extended multiplicand
  ------
   1101110
+ 00000
  -------
   1101110
+ 0110        shifted and negated multiplicand
  -------
 1 00111110    product = 00111110₂ = +30₁₀
   ^^^^^^^
   product
```

ignore this carry out

Note: As you see from the above, the carry out from the last step (the last addition I mean) must be ignored.

- Codes

The BCD code

BCD stands for Binary Coded Decimal. This code is also called 8421 weighted code. The BCD code encodes the decimal digits 0–9 with their unsigned

binary representations $0000_2 - 1001_2$.

The following table shows the correspondence between the ten decimal digits and their BCD representations

Decimal digit	BCD(8421)
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Note: The code words 1010 through 1111 are not used.

Note: In the BCD code, the weights of the BCD bits are 8, 4, 2 and 1 and this is why the code is also called 8421 code

Note: You can place two BCD digits in one 8-bit byte. This is called packed-BCD re-

presentation. This way, one byte will represent the values from 0 to 99 as opposed to 0 to 255 for an unsigned 8-bit binary number.

Note: To represent a signed BCD number you need a sign digit. In the 10's complement system 0000 indicates the plus sign while 1001 indicates the minus sign.

• BCD Addition

BCD addition is similar to unsigned 4-bit binary addition, except that a correction is required if the result exceeds $1001_2 = 9_{10}$. The correct result is obtained by adding $0110_2 = 6_{10}$. Some examples follow.

Example: Perform in BCD the addition 6+8

Answer:

$$\begin{array}{r} 6 \\ +\,8 \\ \hline 14 \end{array} \qquad \begin{array}{r} 0\,1\,1\,0 \\ +\,1\,0\,0\,0 \\ \hline 1\,1\,1\,0 \\ +\,0\,1\,1\,0 \\ \hline 1\,0\,1\,0\,0 \end{array}$$

correction needed

1 4 ← result is 14

Example: Perform in BCD the addition 3+3

Answer:

$$\begin{array}{r} 3 \\ +3 \\ \hline 6 \end{array} \qquad \begin{array}{r} 0011 \\ +0011 \\ \hline 0110 \end{array} \leftarrow \text{no correction needed}$$

$$\underline{\quad 6 \quad} \leftarrow \text{result is } 6$$

Example: Perform in BCD the addition 9+9

Answer:

$$\begin{array}{r} 9 \\ +9 \\ \hline 18 \end{array} \qquad \begin{array}{r} 1001 \\ +1001 \\ \hline 10010 \end{array} \leftarrow \text{correction needed}$$

$$\begin{array}{r} +\ 0110 \\ \hline 11000 \end{array}$$

$$\underline{1} \quad \underline{8} \leftarrow \text{result is } 18$$

- **Gray Code**

A Gray code is a code (binary sequence) with the property that only one bit changes between each pair of consecutive code words.

A 3-bit Gray code is shown in the table on the next page.

Decimal Number	Binary Code	Gray Code
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

As you can see from the above table only one bit changes between each pair of consecutive code words in the Gray code. For example, the codes for 0 and 1 differ only in the third bit and the codes for 3 and 4 differ only in the first bit.

Note: A Gray code is often used when translating an analog quantity into digital form. In this case, a small change in the analog quantity will change only one bit in the code, which gives more reliable operation than if two or more bits changed at a time.

• How to construct a Gray Code

Just follow the following simple rules:

1. A 1-bit Gray code has two code words, 0 and 1.
2. The first 2^n code words of an $(n+1)$-bit Gray code equal the code words of an n-bit Gray code, written in order with a leading zero appended.
3. The last 2^n code words of an $(n+1)$-bit Gray code equal the code words of an n-bit Gray code, written in reverse order with a leading one appended.

I will now show you how to construct a 3-bit Gray code. See below.

0
1 } 1-bit Gray code

append 0 → 00 } in order
" 0 → 01 }
" 1 → 11 } in reverse order
" 1 → 10 }
} 2-bit Gray code

→ Go to next page →

append 0 → 000
" 0 → 001
" 0 → 011
" 0 → 010
" 1 → 110
" 1 → 111
" 1 → 101
" 1 → 100

(in order; in reverse order) — 3-bit Gray code

Note: The above procedure for constructing a Gray code is recursive.

- More on Codes
- The 2421 Code

For the 2421 code the weights used are 2, 4, 2 and 1 respectively. For example decimal 6 can be represented in the 2421 code as 1100. Just see that $1\times2+1\times4+0\times2+0\times1=6$. Decimal 9 can be represented in the 2421 code as 1111. Just see that $1\times2+1\times4+1\times2+1\times1=9$.

An interesting property of the 2421 code is that it is self-complementing. This means that the code word for the 9's complement of any digit can be obtained by complementing the individual bits of the digit's code word.

A table with the decimal digits 0 through 9 and their representations in the 2421 code will be provided later. The self-complementing property of the 2421 code will be demonstrated then.

• The Excess-3 Code

For the excess-3 code the code word for each decimal digit is the corresponding BCD code word plus $0011_2 = 3_{10}$. This code (the excess-3 code) is also a self-complementing code. This code finds applications in counters which you will study in EE 2730. A table with the decimal digits 0 through 9 and their representations in the excess-3 code will be provided later. The self-complementing property of this code will be demonstrated then.

• The 2-out-of-5 Code

The 2-out-of-5 code is a 5-bit code. This code has the property that exactly 2 out of the 5 bits are 1 for every valid code combination. The code has useful error-checking properties because if any one of the bits in a code combination is changed due to a

malfunction of the logic circuitry, the number of 1 bits is no longer exactly two.

The table below shows the decimal digits 0 through 9 and their representations in the 2421 code, the excess-3 code and the 2-out-of-5 code.

Decimal Digit	2421 Code	Excess-3 Code	2-out-of-5 Code
0	0000	0011	00011
1	0001	0100	00101
2	0010	0101	00110
3	0011	0110	01001
4	0100	0111	01010
5	1011	1000	01100
6	1100	1001	10001
7	1101	1010	10010
8	1110	1011	10100
9	1111	1100	11000

I will now demonstrate the self-complementing property of the 2421 code.

→ Go to next page →

Take the code word 1111 which represents the decimal digit 9. Complement the bits of 1111. You get 0000. This represents the decimal digit 0. Observe that 9's-complement of 9 = 0; (Just see that $10^1 - 1 - 9 = 9 - 9 = 0$).

Take the code word 1110 which represents the decimal digit 8. Complement the bits of 1110. You get 0001. This represents the decimal digit 1. Observe that 9's-complement of 8 = 1; (Just see that $10^1 - 1 - 8 = 9 - 8 = 1$).

Take the code word 1011 which represents the decimal digit 5. Complement the bits of 1011. You get 0100. This represents the decimal digit 4. Observe that 9's-complement of $5 = 9 - 5 = 4$.

You can do the rest by yourselfs.

I will now demonstrate the self-complementing property of the excess-3 code.

Take the code word 1100 which represents the decimal digit 9. Complement the bits of 1100. You get 0011. This represents the decimal digit 0. Observe that 9's-complement of $9 = 9 - 9 = 0$.

Take the code word 1010 which represents the decimal digit 7. Complement the bits of 1010. You get 0101. This represents the decimal digit 2. Observe that 9's-complement of $7 = 9 - 7 = 2$.

You can do the rest by yourselfs.

• Character Codes

Many applications of computers require the processing of data which contains numbers, letters, characters and other symbols. In order to transmit such alphanumeric data to and from a computer or store it internally in a computer, each symbol must be represented by a binary code. One common alphanumeric code is the ASCII code. ASCII stands for American Standard Code for Information Interchange. The ASCII code is a 7-bit code, so $2^7 = 128$ different code combinations are available to represent letters, numbers, and other symbols.

A very small portion of the ASCII code is shown on next page. I do not provide the entire code which is anyways difficult to memorize.

Character	ASCII Code
0	0110000
1	0110001
2	0110010
3	0110011
4	0110100
5	0110101
space	0100000
#	0100011
=	0111101
?	0111111
A	1000001
a	1100001
H	1001000
e	1100101
l	1101100
this is lower case O	1101111
@	1000000

The word "Hello" is represented in ASCII as follows:

1001000 1100101 1101100 1101100 1101111

H e l l o