

# Lecture 13: Arithmetic

## Addition

The addition of two binary numbers is carried out in a bitwise fashion, just as normal addition. We add any two bits according to the following truth table:

A	B	SUM	CARRY
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$\text{SUM} = A' \cdot B + A \cdot B'$$

$$\text{CARRY} = A \cdot B$$

In our design method for combinational circuits we do not automatically use the exclusive-or gate. The truth table for this gate is the same as that for SUM above. At the transistor level it is possible to make exclusive-or and exclusive-nor gates not much bigger than single NAND and NOR gates, and therefore we should always look for exclusive-or and exclusive-nor simplification. The minterm formulation of XOR and XNOR simplifications is:

$$\text{XOR} : \quad A' \cdot B + A \cdot B' = A \oplus B$$

$$\text{XNOR} : \quad (A \cdot B + A' \cdot B') = (A \oplus B)'$$

The symbol  $\oplus$  is commonly used for the "exclusive or" function. As noted above we can apply a simplification to the equation for the SUM:  $\text{SUM} = A \oplus B$

The above equations make up what is called a *half adder*. When adding numbers of more than one bit in length we need also to add the carry from the previous stage as well as the two digits. To do this we use a *full adder* which has a carry in, Cin, and is represented by the following truth table.

A	B	Cin	SUM	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

From the truth table we obtain the canonical equations for SUM and Cout and simplify them using Boolean Algebra.

$$\begin{aligned} \text{SUM} &= A' \cdot B' \cdot C + A' \cdot B \cdot C' + A \cdot B' \cdot C' + A \cdot B \cdot C \\ &= A' \cdot (B' \cdot C + B \cdot C') + A \cdot (B' \cdot C' + B \cdot C) \\ &= A' \cdot (B \oplus C) + A \cdot (B \oplus C)' \\ &= A \oplus B \oplus C \\ \text{Cout} &= A' \cdot B \cdot C + A \cdot B' \cdot C + A \cdot B \cdot C' + A \cdot B \cdot C \\ &= C \cdot (A' \cdot B + A \cdot B') + A \cdot B \\ &= C \cdot (A \oplus B) + A \cdot B \end{aligned}$$

The resulting circuit is shown in Figure 2. Binary addition can be implemented to any precision by means of a half adder for the bottom bit followed by a set of full adders for all the other bits connected as shown in Figure 3. Notice that the more bits that are required, the longer it will take for the adder to complete the sum.

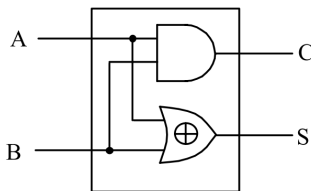


Figure 1: One bit half adder

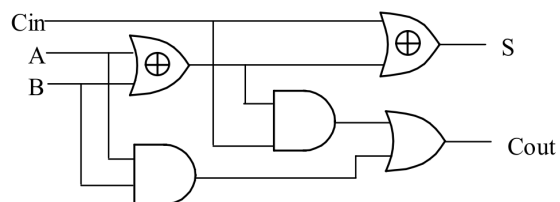


Figure 2: One bit full adder

It is possible also to add two streams of serial data using a one-bit full adder and a flip flop to delay the carry from one stage to the next as shown in Figure 4. Notice that the serial data streams must be ordered with the least significant bit first. Processing serial data in this way is only used in specialist applications where serial data is present but there is no need to convert it to parallel form. General purpose computation would normally employ parallel adders which are faster, but use more gates.

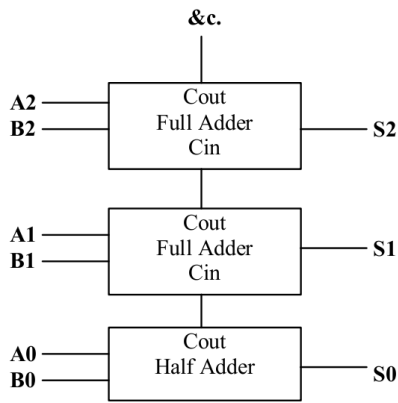


Figure 3: N-bit full adder

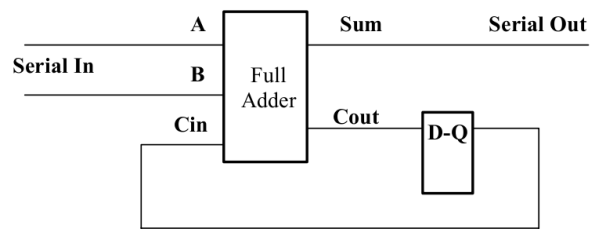


Figure 4: Serial bitstream adder

## Subtraction

Subtraction circuitry can be designed in an analogous manner, but this time we need to consider borrowing from the next most significant bit. The truth table for a full subtractor which takes B from A with a pay back P from the previous stage is:

A	B	P	DIFFERENCE	BORROW
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

As before we can derive the canonical equations for DIFFERENCE and BORROW and simplify them using Boolean Algebra. The final results are

$$\begin{aligned} \text{DIFFERENCE} &= A \oplus B \oplus P \\ \text{BORROW} &= A' \cdot (B + P) + B \cdot P \end{aligned}$$

Since  $B \cdot P$  covers the case  $B=P=1$  we can also write:  $\text{BORROW} = A' \cdot (B \oplus P) + B \cdot P$ . This reduces the number of gates by one since we already need to provide hardware for  $B \oplus P$  in the circuit for DIFFERENCE.

The full subtractor for one bit can be implemented as shown in Figure 5, and circuit for n bits is connected in an equivalent manner to Figure 3. In practice, subtraction is often achieved by two's complement addition. The two's complement of a binary digit is found by complementing the individual bits of a number and adding one to the bottom digit, losing the carry. The method can be used to implement a subtractor as shown in Figure 6.

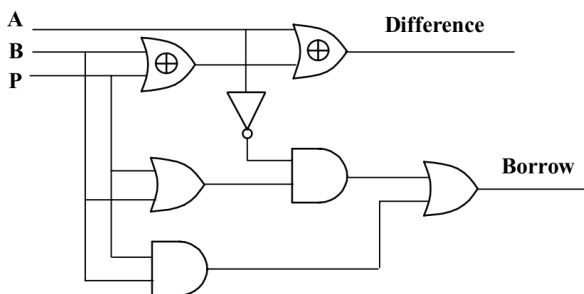


Figure 5: One bit full subtractor

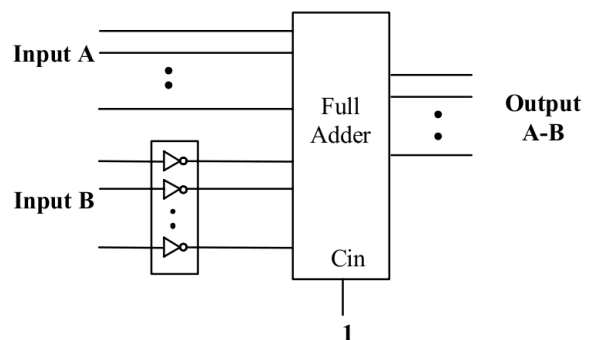


Figure 6: Two's Complement Subtractor

## Multiplication

Multiplication of two numbers is carried out by a process of multiplying all combinations of the individual digits, and adding them up in the appropriate positions. For example, we can multiply 13 by 42 by taking the

four individual products  $4 \times 1$ ,  $4 \times 3$ ,  $2 \times 1$ ,  $2 \times 3$ , and adding them raised to the appropriate power of 10 to form:  $4 \times 1 \times 10^2 + 4 \times 3 \times 10 + 2 \times 1 \times 10 + 2 \times 3$ . This is the way that long multiplication is normally carried out. We can apply the same principle to binary arithmetic. We will follow the functional approach discussed in the last lecture to come up with a design that we can scale to any size. In the simplest case let us consider multiplying two two-digit numbers:

$$A_1A_0 \times B_1B_0 = A_1 \times B_1 \times 2^2 + A_1 \times B_0 \times 2^1 + A_0 \times B_1 \times 2^1 + A_0 \times B_0$$

Now, since  $A_1$ ,  $A_0$ ,  $B_1$  and  $B_0$  are all binary digits we can replace the multiplies by ANDs

$$A_1A_0 \times B_1B_0 = A_1 \cdot B_1 \times 2^2 + A_1 \cdot B_0 \times 2^1 + A_0 \cdot B_1 \times 2^1 + A_0 \cdot B_0$$

And, since multiply by two is equivalent to shift right, we can replace these by shifts which we hard wire to obtain the multiplier of Figure 7.

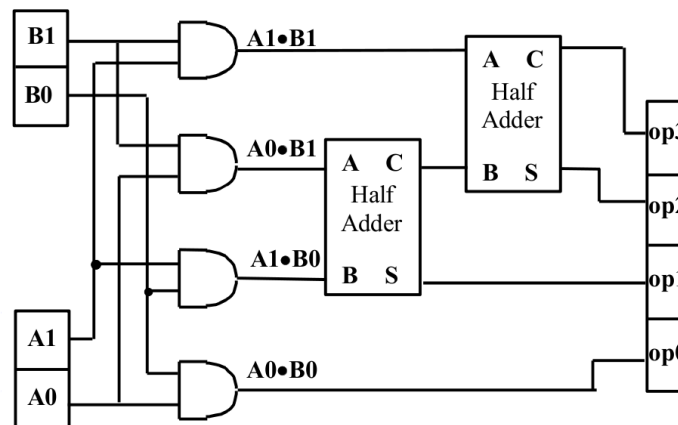


Figure 7: Two bit combinational multiplier

The next step is to apply the same reasoning recursively, and design a four bit multiplier using the two bit multiplier of figure 7. Suppose that we have two four bit numbers  $PQ$  and  $RS$  where  $P$  and  $R$  are the two most significant bits, and  $Q$  and  $S$  are the two least significant bits respectively. Now we can write:

$$PQ \times RS = P \times R \times 2^4 + P \times S \times 2^2 + Q \times R \times 2^2 + Q \times S$$

Since  $P, Q, R$  and  $S$  are two digit numbers, the products,  $P \times R$ ,  $P \times S$ ,  $Q \times R$  and  $Q \times S$  can be computed using the two bit multiplier that we just designed. We are left with the problem of how to implement the powers of two. We can solve this by adding up the results of the individual products with the shifts hard wired. The circuit for our four bit multiplier is shown in Figure 8. Clearly we can extend this idea to design a multiplier for a bit length of any power of two.

The above design only works for unsigned binary digits. In order to extend it to signed numbers we need to add further circuitry to detect if either of the input numbers is negative. This can be done quite simply by looking at the top bit, which in twos compliment arithmetic is 1 for negative numbers. This top bit can be used to control a multiplexer which either selects the number or its twos complement. The twos complement can be implemented by an inverting each bit then incrementing with an adder. The output sign can be determined by the exclusive or of the input signs, and a twos complement circuit with a multiplexer is also required to set it correctly.

## Division

It is possible to build a combinational circuit to carry out division, but in practice it is often done procedurally, with a sequential circuit, or in the machine code using shifts and subtracts.

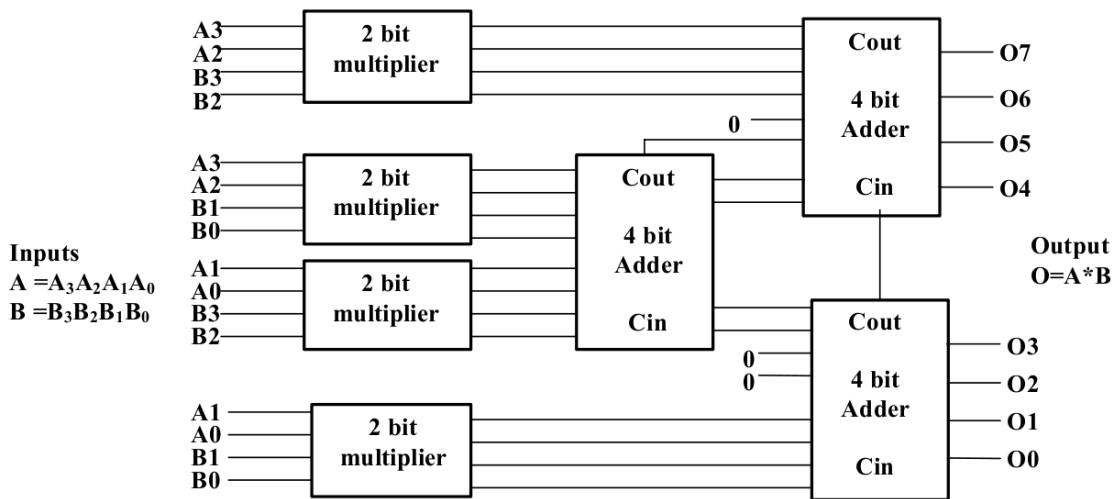


Figure 8: Four bit combinatorial multiplier

### Combining different arithmetic functions

Since we know that designing functionally is a good idea, it makes sense to put together our arithmetic functions in a single design that we can scale to any size. The most common combined arithmetic circuit that is in use is called the Arithmetic and Logic Unit, or ALU for short. It has control inputs, which select the function, and two data inputs. We will look at its design in more detail next time. For the moment we will illustrate the principle by designing an Add/Subtract unit which has four bit precision. It is shown in figure 9. The function is selected by setting the Add/Subtract input (1=add, 0=subtract). This selection input goes to five multiplexers, one for each data bit and one for the carry. The box marked 4-way mpx is just four identical multiplexers, one for each data bit. To increase the precision to 8 bits we simply replicate the circuit and take the carry in of the most significant four bits from the carry out of the least significant four bits. We can thus scale the design up easily to any precision that we want.

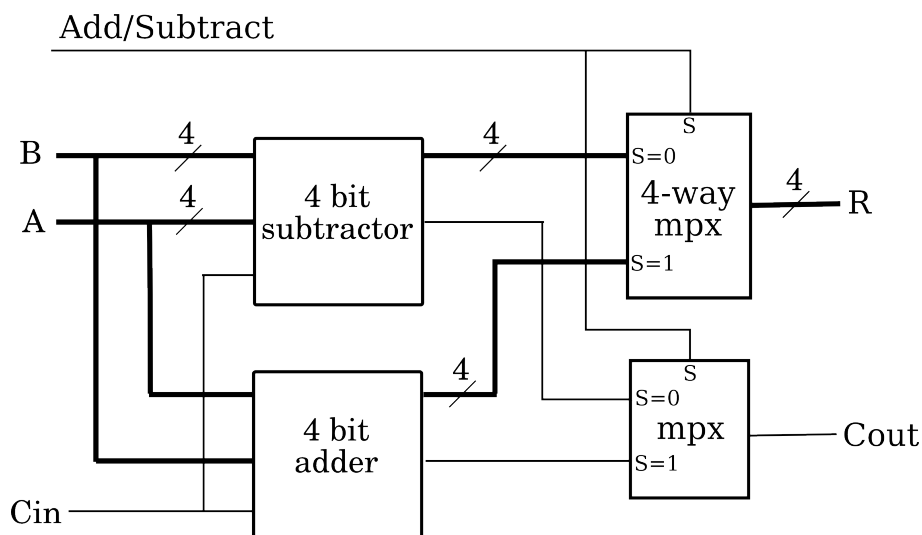


Figure 9: Combined Add and Subtract Circuit