2.4 HIGH PERFROMANCE ARTHMETIC

The performance improvement in arithmetic operations like addition, multiplication and division will increase the overall computational speed of the machine.

High performance adders

The high performance adders takes an extra input namely the transit time.

The transmit time of a logical unit is used as a time base in comparing the operating speeds of different methods, and the number of individual logical units required is used in the comparison of costs.

The two multi-bit numbers being added together will be designated as A and B, with individual bits being A1, A2, B1, etc. The third input will be C. Outputs will be S (sum) R (carry), and T (transmit). The two multi bit numbers being added together will be designated asA and B, with individual bits being A1, A2, B1, etc. The third input will be C. Outputs will be S (sum) R (carry), and T (transmit).

The time required to perform an addition in conventional adder is dependent on the time required for a carry originating in the first stage to ripple through all intervening stages to the S or R output of the final stage. Using the transit time of a logical block as a unit of time, this amounts to two levels to generate the carry in the first stage, plus two levels per stage for transit through each intervening stage, plus two levels to form the sum in the final stage, which gives a total of two times the number of stages.

$C_n = R_{n-1}$

$C_n = D_{n-1} \| T_{n-1} R_{n-2}$

$C_n = D_{n-1} \| T_{n-1} D_{n-2} \| T_{n-1} T_{n2} R_{n-3}$

By allowing n to have successive values starting with one and omitting all terms containing a a resulting negative subscript, it may be seen that each stage of the adder will require one OR stage with n inputs and n AND circuits having one through n inputs, where N is the position number of the particular stage under consideration.

High performance Multiplication Multiplication using variable length shift

- The multiplier and the partial product will always be shifted the same amount and at
  the same time.
- The multiplier is shifted in relation to the decoder, and the partial product with relation to the multiplicand.
- Operation is assumed starting at the low-order end of the multiplier, which means that shifting is to the right.
- If the lowest-order bit of the multiplier is a one, it is treated as though it had been approached by shifting across zeros.

Rules:

- When shifting across zeros (from low order end of multiplier), stop at the first one.

  a) If this one is followed immediately by a zero, add the multiplicand, then shift across all following zeros.

  b) If this one is followed immediately by a second one, subtract the multiplicand, then shift across all following ones.

2. When shifting across ones (from low order end of multiplier), stop at the first zero.

   a) If this zero is followed immediately by a one, subtract the multiplicand, then shift across all following ones.

   b) If this zero is followed immediately by a second zero, add the multiplicand, then shift across all following zeros.

- A shift counter or some equivalent device must be provided to keep track of the number of shifts and to recognize the completion of the multiplication.

- If the high-order bit of the multiplier is a one and is approached by shifting across ones, that shift will be to the first zero beyond the end of the multiplier, and that zero along with the bit in the next higher order position of the register will be decoded to determine whether to add or subtract.
- For this reason, if the multiplier is initially located in the part of the register in which the product is to be developed, it should be so placed that there will be at least two blank positions between the locations of the low-order bit of the partial product and the high-order bit of the multiplier.
- Otherwise the low-order bit of the product will be decoded as part of the multiplier.

Multiplication Using Uniform Shifts

- Multiplication which uses shifts of uniform size and permits predicting the number of cycles that will be required from the size of the multiplier is preferable to a method that requires varying sizes of shifts.
- The most important use of this method is in the application of carry-save adders to multiplication although it can also be used for other applications.

Uniform shifts of two

- Assume that the multiplier is divided into two-bit groups, an extra zero being added to the high-order end, if necessary, to produce an even number of bits.
- Only one addition or subtraction will be made for each group, and, using the position of the low-order bit in the group as a reference, this addition or subtraction will consist of
  either two times or four times the multiplicand.

- These multiples may be obtained by shifting the position of entry of the multiplicand into the adder one or two positions left from the reference position.
- The last cycle of the multiplication may require special handling.
- Following any addition or subtraction, the resulting partial product will be either correct or larger than it should be by an amount equal to one times the multiplicand.
- Thus, if the high-order pair of bits of the multiplier is 00 or 10, the multiplicand would be multiplied by zero or two and added, which gives a correct partial product.

  If the high-order pair of bits is 01 or 11, the multiplicand is multiplied by two or four, not one or three, and added. This gives a partial product that is larger than it should be, and the next add cycle must correct for this.
- Following the addition the partial product is shifted left- two positions. This multiplies it by four, which means that it is now larger than it should be by four times the multiplicand.
- This may be corrected during the next addition by subtracting the difference between four and the desired multiplicand multiple.
- Thus, if a pair ends in zero, the resulting partial product will be correct and the following operation will be an addition.
- If a pair ends in a one, the resulting partial product will be too large, and the following operation will be a subtraction.
- It can now be seen that the operation to be performed for any pair of bits of the multiplier may be determined by examining that pair of bits plus the low-order bit of

  the next higher-order pair.
- If the bit of the higher-order pair is a zero, an addition will result; if it is one, a

subtraction will result. If the low-order bit of a pair is considered to have a value of one and the high-order bit a value of two, then the multiple called for by a pair is the numerical value of the pair if that value is even and one greater if it is odd.

- If the operation is an addition, this multiple of the multiplicand is used. If the operation
  is a subtraction (the low-order bit of the next higher order pair a one), this value is combined with minus four to determine the correct multiple to use.
- The result will be zero or negative, with a negative result meaning subtract instead of add.

Multiplication Using Carry-Save Adders

- When successive additions are required before the final answer is obtained, it is possible to delay the carry propagation beyond one stage until the completion of all of the additions, and then let one carry-propagate cycle suffice for all the additions. Adders used in this manner are called **carry-save adders.**
- A carry-save adder consists of a number of stages, each similar to the full adder. It differs from the ripple-carry adder in that the carry (R) output is not connected directly

Multiplication Using Carry-Save Adders

- When successive additions are required before the final answer is obtained, it is possible to delay the carry propagation beyond one stage until the completion of all of the additions, and then let one carry-propagate cycle suffice for all the additions. Adders used in this manner are called **carry-save adders.**
- A carry-save adder consists of a number of stages, each similar to the full adder. It differs from the ripple-carry adder in that the

carry (R) output is not connected directly form of SIMD(Single Instruction Multiple Data) processing. It is possible to apply sub word parallelism to noncontiguous sub words of different sizes within a word.

- In practical implementation is simple if sub words are same size and they are contiguous within a word. The data parallel programs that benefit from sub word parallelism tend to process data that are of the same size.

Example: If word size is 64bits and sub words sizes are 8,16 and 32 bits. Hence an instruction operates on eight 8bit sub words, four 16bit sub words, two 32bit sub words or one 64bit sub word in parallel.