**Assignment No.4**

|  |
| --- |
| **Title of Assignment:**  Write 64 bit ALP to perform multiplication of two 64 bit number using 1) Shift and add method 2) Successive addition method. |
| **Relevant Theory:**  Shift-and-add multiplication is similar to the multiplication performed by paper and pencil. This method adds the multiplicand *X* to itself *Y* times, where *Y* denotes the multiplier. To multiply two numbers by paper and pencil, the algorithm is to take the digits of the multiplier one at a time from right to left, multiplying the multiplicand by a single digit of the multiplier and placing the intermediate product in the appropriate positions to the left of the earlier results. As an example, consider the multiplication of two unsigned 4-bit numbers, 8 (1000) and 9 (1001).  Multiplicand 1000 ×  Multiplier 1001  \_\_\_\_\_\_\_\_\_\_\_\_  1000  0000  0000  1000  \_\_\_\_\_\_\_\_\_\_\_  Product 1001000  In the case of binary multiplication, since the digits are 0 and 1, each step ofthe multiplication is simple. If the multiplier digit is 1, a copy of the multiplicand (1 ×multiplicand) is placed in the proper positions; if the multiplier digit is 0, a number of0 digits (0 × multiplicand) are placed in the proper positions.Consider the multiplication of positive numbers. The first version of themultiplier circuit, which implements the shift-and-add multiplication method for two*n*-bit numbers, is shown in Figure 1.  The 2*n*-bit product register (*A*) is initialized to 0. Since the basic algorithmshifts the multiplicand register (*B*) left one position each step to align the multiplicandwith the sum being accumulated in the product register, we use a 2*n*-bit multiplicandregister with the multiplicand placed in the right half of the register and with 0 in theleft half.  The algorithmstarts by loading the multiplicand into the *B* register, loading the multiplier into the *Q* register, and initializing the *A* register to 0. The counter *N* is initialized to*n*. The least significant bit of the multiplier register (*Q*0) determines whether the multiplicand is added to the product register. Right shift A followed by Q and repeats this procedure till N becomes 0. The detail algorithm is explain in Figure 1.      **Figure 1.Flowchart for Multiplication using Shift and add Method.**  **Example 1**  Using 4-bit numbers, perform the multiplication 9 × 12 (1001 × 1100).  **Answer**  Table 1 shows the value of registers for each step of the multiplication algorithm.  **Table 1.** Multiply example using Shift and add method algorithm.   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Step N** | **A (AH)** | **Q (AL)** | **B (BL)** | **Operation** | | 4 | 0000 | 1001 | 1100 | Initialization | | 4 | 1100  0110 | 1001  0100 | 1100  1100 | A=A+B  Shift Right A\_Q | | 3 | 0011 | 0010 | 1100 | Shift Right A\_Q | | 2 | 0001 | 1001 | 1100 | Shift Right A\_Q | | 1 | 1101  **0110** | 1001  **1100** | 1100  1100 | A=A+B  Shift Right A\_Q | | **Final Answer** | **01101100** | | | |   Instruction used:  **MUL source**  Operation= multiplies source with destination  Source = memory location or any register  Destination= Accumulator for 64 bit : RAX  Result : DX: Ax pair  **Design Analysis/ Implementation Logic:**  **Algorithm**  **i) shift and add method**   1. Accept the first 64 bit number 2. Accept the second 64 bit number 3. Initialize count=64 4. Check Q0 bit if Q0==1 then perform a=a+b else next step Shift right A\_Q. 5. Decrement count. 6. Repeat from step number 4 till count becomes 0. 7. Display the 128 bit result using display procedure   **ii) successive addition method**   1. Accept the first 64 bit number 2. Accept the second 64 bit number 3. Consider initial value of partial product as 0000 4. Use multiplier as a counter 5. Add multiplicand multiplier times with partial product 6. Display final partial product as a result using display procedure   e.g. **2\* 3**  **2 + 2 +2 = 6 or 3 + 3 =6** |
| **Mathematical Modeling:**  Let num1 and num2 are two 64 bit numbers. Result is 128 bit final result storing registers.  Num2  Result= ∑ num1+num1  I=1 |
| **Testing:**  **Test Conditions:**  Check multiplication using both the methods with proper result  **Input:**  **First Number: 3 in binary (0011)**  **Second Number:2 in binary (0010)**  **Output:**   1. Using successive addition: 0006 2. Using Shift and add method: 0006 |
| **FAQs:**   1. Explain Shift and add method. 2. Explain MUL instruction. Execution of MUL instruction. 3. Give example of unconditional and conditional instructions. |
| **Conclusion:**  Successfully completion of overlapped and non-overlapped block of transfer with and without MOVSB instruction. |