# Hardware Driven Approach for Enhancing C's Operators' Library so to Increase C's Functional Capability

Abhishek Garg
Lecturer, CSA Deptt., BSACET, Mathura

Kavita Agrawal
Asst. Prof., CS Deptt., Integral University, Lucknow

## ABSTRACT
The paper published earlier entitled "An approach for swapping process through enclose of swap operator in the C language operators library" gives an idea how an operator could be added to a C operators' library [1]. The concept presented in this paper could be extended to add other operators to the library of operators that serve purpose similar to the swap operator. This time operators are being designed to serve some special mathematical operations for which we do not have an operator directly. To be specific, we are presenting basically the hardware designs for a few mathematical operators with the flowchart for explaining how they work. These operators are factorial, permutation and combination operators.

## GENERAL TERMS
Operators, C library, swap operator, parsing

## KEYWORDS
Factorialoperator, Permutationoperator, Combin ation operator, hardware, mnemonic

## 1. INTRODUCTION
We already have hardwares for addition and subtraction operations. Like we have an adder circuitry for performing addition operation [3]. We have multiplier circuitry for carrying multiply operation; different flavours for the multiplier hardware are booth multiplier and array multiplier[ 3].

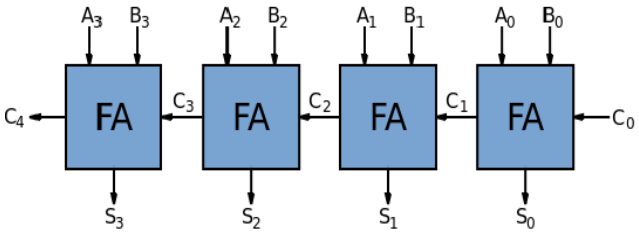


**Fig. 1. Parallel Adder for 4-bit addition[3]**

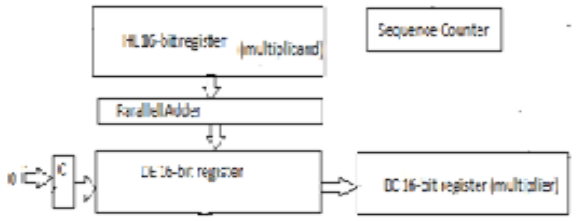


**Fig. 2. Hardware for multiply operation[2]**

## 2. OUR WORK
On the patterns for other operations, we have hardware for factorial, permutation and combination operation. Each hardware could be used for the corresponding operator through addition of a mnemonic for each added operator hardware. Following are the hardware designs for these operations.

### 2.1 Architecture for factorial operator:

**Table 1. Factorial Table (First Pass)**

| HL=1000 | F | DE | BC | SC | GC |
|---|---|---|---|---|---|
| Multiplier in BC | 0 | 0000 | 111 | 11 | 111 |
| BCn=1;add HL | | 1000 | | | |
| First Partial Product | | 1000 | | | |
| Shift right FDEBC | 0 | 0100 | 011 | 10 | |
| BCn=1 ;add HL | | 1000 | | | |
| Second Partial Product | 0 | 1100 | | | |
| Shift right FDEBC | 0 | 0110 | 001 | 01 | |
| BCn=1;add HL | | 1000 | | | |
| Third Partial Product | 0 | 1110 | | | |
| Shift right FDEBC | 0 | 0111 | 000 | 00 | |
| Final Product in DEBC=0111000 | | | | | |

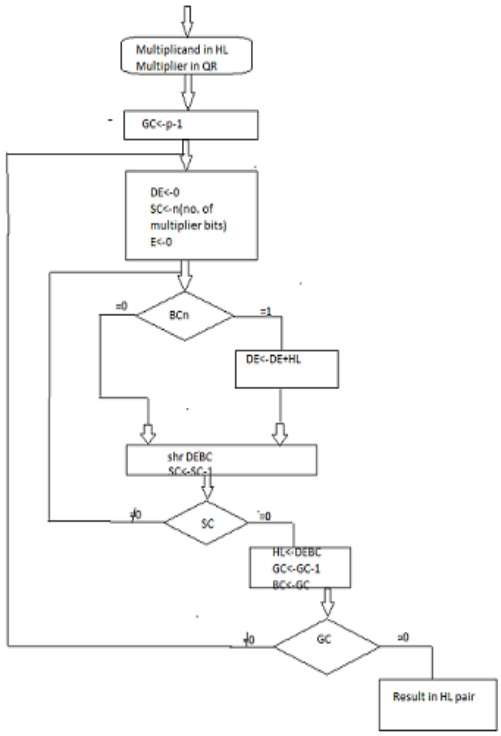


**Fig. 3. Flowchart for factorial operation**

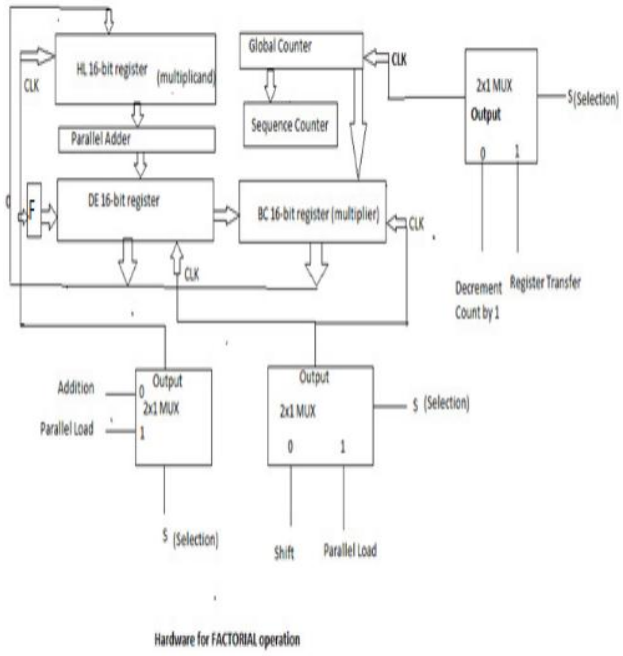


**Fig. 4. Hardware for factorial operator**

Let us demonstatre the process for calculating 8! Using above hardware. Clk=0 in Table 1. At Clk=1,DEBC is parallely loaded into HL. When Clk reaches 0 again,the above process is repeated as shown below in Table 2.1.2.

At last iteration when GC reaches 0, Final product in DEBC.This gives factorial of 8. The multiplier is stored in the BC register.The sequence counter SC is initially set to a number equal to the number of bits in the multiplier.The counter is decremented by 1 after forming each partial product.When the content of the counter reaches 0, the product is formed and the process stops. Above process is repeated no. of times the value in Global Counter GC.On each repetition of above process, multiplier BC is initialized to the value in GC.After each execution of above

**Table 2. Factorial operation (last pass)**

| Now HL=0111000 | F | DE | BC | SC | GC |
|---|---|---|---|---|---|
| Multiplier in BC | 0 | 000000 | 110 | 11 | 110 |
| BCn=0;shift right CDEBC | 0 | 000000 | 011 | 10 | |
| BCn=1;add HL | | 111000 | | | |
| First Partial Product | 0 | 111000 | | | |
| Shift right FDEBC | 0 | 011100 | 001 | 01 | |
| BCn=1;add HL | | 111000 | | | |
| Second Partial Product | 1 | 010100 | | | |
| Shift right FDEBC | 0 | 101010 | 000 | 00 | |
| Final Product in DEBC=101010000 | | | | | |

process,GC is decremented by 1.When GC reaches 0,the whole process stops.Clk=0 is used for execution of each occurrence of above process governed by value of SC.This process is repeated no. of times the value in GC.For each GC,process is repeated SC no. of times.Total no. of processes occur GC no. of times. The description of above process is as follows:-

Registers F and DE are cleared, SC is set to a number equal to the number of bits of the multiplier.GC is set to value of multiplicand in HL pair minus 1. After the initialization, loworder bit of the multiplier in BCn is tested. If it is a 1, the multiplicand in HL pair is added to the present partial product in DE pair.If it is a 0, nothing is done.Register FDEBC is then shifted once to the right to form the new partial product. The sequence counter SC is decremented by 1 and its new value checked. If it is not equal to 0, process is repeated and a new partial product is formed.The process stops when SC=0. The partial product formed in DE is shifted into BC one bit at a time and eventually replaces the multiplier.The final product is available in both DE and BC with DE holding the most significant bits and BC holding the least significant bits.Now at Clk=1[o/p from MUX], DEBC is parallel loaded into HL when selection bit of MUX is 1.Register transfer from GC to BC takes place with Clk=1. At Clk now 1 at all registers, the above process repeats now with HL holding previous product and BC holding new multiplier [which is 1 less than previous value]. At the end when GC reaches 0, this overall process of calculating factorial stops.At last iteration when GC approaches 0, DEBC will hold final factorial value.The largest that the above hardware calculates is limited only by size of registers.

## 2.2 Architecture for permutation operator:

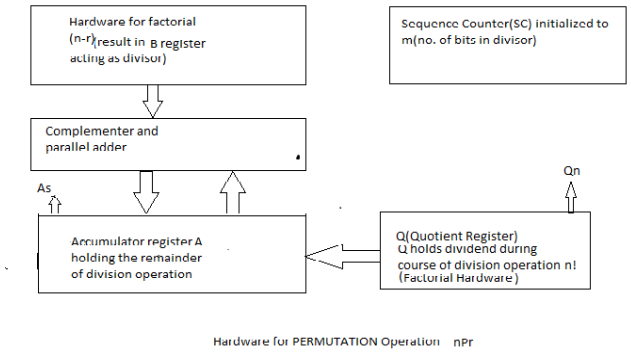


**Fig. 5. Hardware for permutation Operator**

Let us demonstrate the process used to calculate 4P2 by above hardware as in Table 3.
(n-r)!=(4-2)!=2!=2.(factorial hardware is used to calculate (n-r)!stored in B as divisor.)
Similarly, Q (dividend)=4! =24.

**Table 3. Permutation Operation**

| Operation | A | Q | B | SC |
|---|---|---|---|---|
| | 00000 | 11000 | 00010 | 101 |
| Shift left AQ | 00001 | 1000_ | | |
| A=A-B | 11111 | 1000_ | | |
| As=1<br>Put Qn=0<br>A=A+B<br>SC=SC-1 | 00001 | 10000 | | 100 |
| Shift left AQ | 00011 | 0000_ | | |
| A=A-B _<br>As=0 | 00001<br>00001 | 0000<br>00001 | | 011 |
| Put Qn=1<br>SC=SC-1 | | | | |
| Shift left AQ | 00010 | 0001_ | | |
| A=A-B | 00000 | 0001_ | | |
| As=0<br>Put Qn=1<br>SC=SC-1 | 00000 | 00011 | 010 | |
| Shift left AQ | 00000 | 0011_ | | |
| A=A-B | 11110 | 0011_ | | |
| As=1<br>Put Qn=0<br>A=A+B<br>SC=SC-1 | 00000 | 00110 | | 001 |
| Shift left AQ | 00000 | 0110_ | | |
| A=A-B | 11110 | 0110_ | | |
| As=1<br>Put Qn=0<br>A=A+B<br>SC=SC-1 | 00000 | 01100 | 000 | |

Quotient in Q=01100(result)
Remainder in A=00000

## 2.3Architecture for Combination operator:

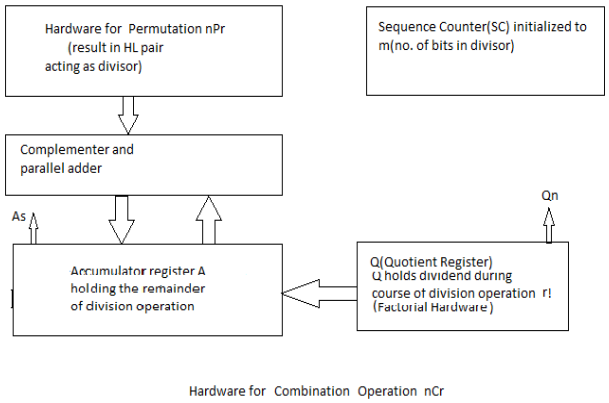


**Fig. 6. Hardware for Combination operator**

This hardware makes use of factorial and permutation hardware both to calculate nCr similarly.

## 3. CONCLUSION

Similar to the above mathematical operators' hardware design, operators' in other category of operators could be designed likewise.

## 4. REFERENCES

[1] "An approach for swapping process through enclose of a swap operator in C language operators' library" by Abhishek Garg

[2] "Computer System Architecture" by M. Morris Mano, Third Edition

[3] "Digital Logic Design" by M. Morris Mano.