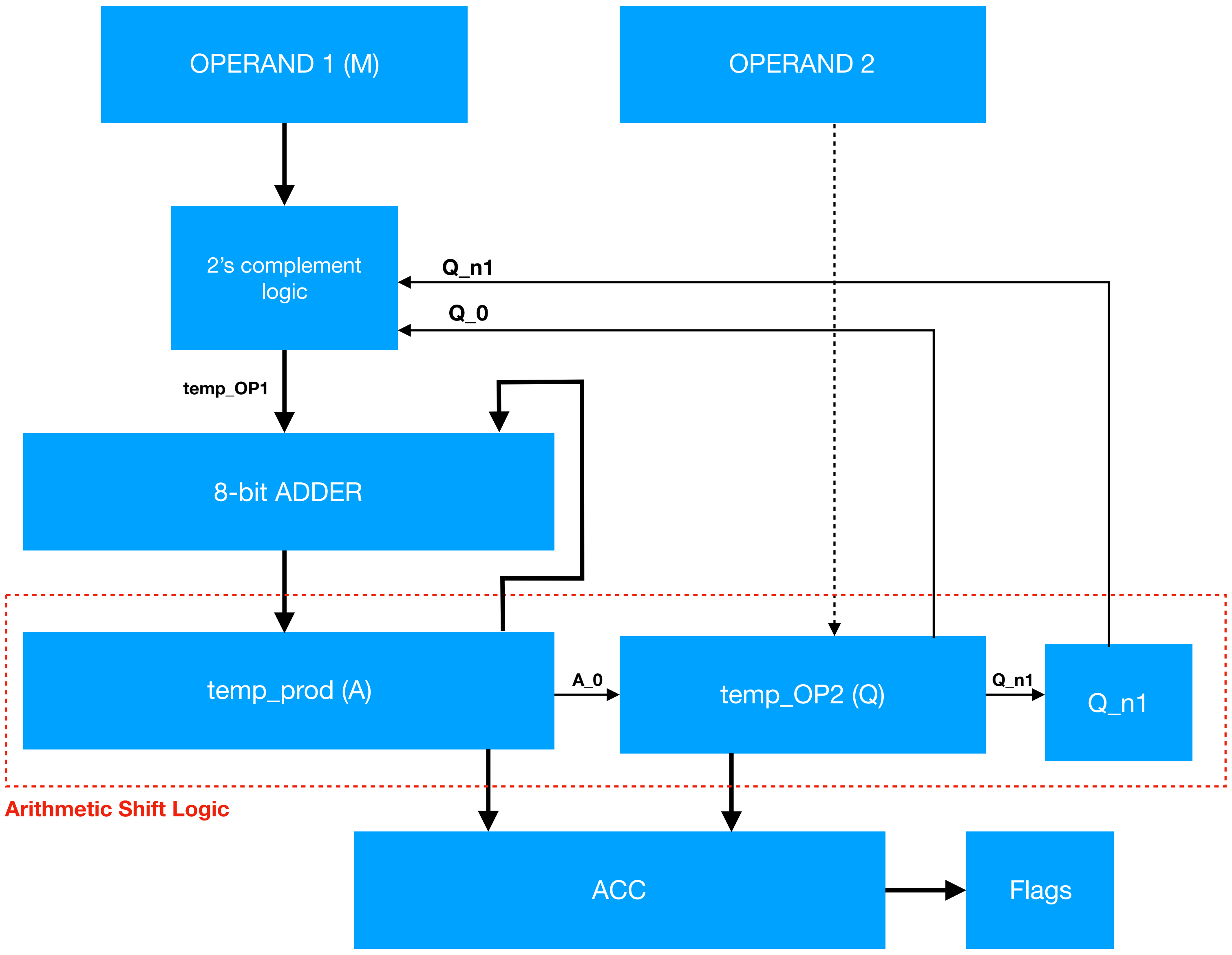
MULTIPLY (based on Booth's algorithm)



Note:

- Check out Booth's algorithm for more details.
- M (Multiplicand), Q (Multiplier), A (product)
- temp_prod (A), temp_OP2 (Q), operand1 (M), Q_0 (Q_0), Q_n1 (Q_{-1})
- The 2's complement logic applies 2's complement on operand1 depending on the value of Q_0 and Q_n1. The output is temp_OP1 connected to the first operand of the 8-bit ADDER.
- At initial state, temp_OP2 will receive the value of operand1. Take note that operand1 will not be modified, only temp_OP2.
- temp_prod will always be the second operand of 8-bit adder.
- Use the function setFlags( ) to set the zero (ZF), sign (SF), overflow (OF) and carry (CF) flags.

Solution in performing arithmetic shift right of A, Q and Q_n1 (Arithmetic Shift Logic):

- Extract and save the MSB and LSB of temp_prod (A).
- Extract and save LSB of temp_OP2 (Q).
- Shift right temp_prod (A) and temp_OP2 (Q) 1 bit to the right.
- Write the saved LSB of temp_prod (A) to the MSB of temp_OP2. The saved LSB of temp_OP2 to will be assigned to Q_n1.
- Write saved MSB of temp_prod (A) and write it as the MSB of temp_prod (A).

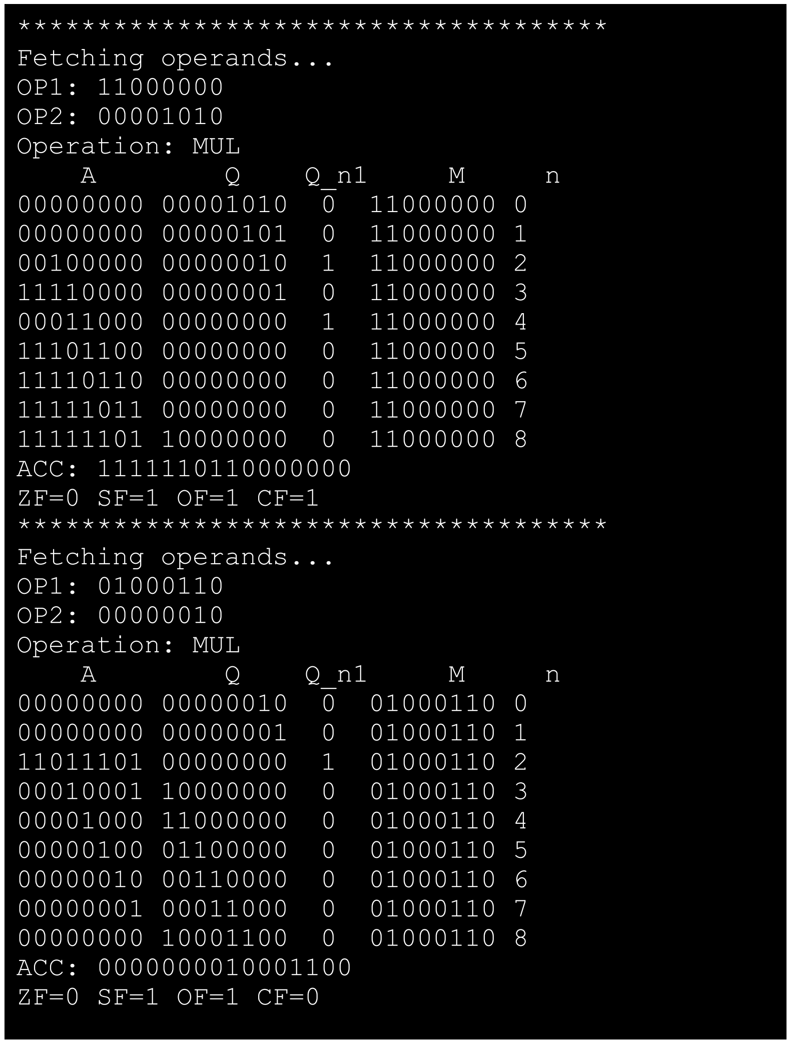
```
...
...
...
MSB_A = (temp_prod >> 8) & 0x01;        // most significant bit of temp_prod (A)
LSB_A = temp_prod & 0x01;               // least significant bit of temp_prod (A)
LSB_Q = temp_OP2 & 0x01;                // least significant bit of temp_OP2 (Q)

temp_prod = temp_prod >> 1;
temp_OP2 = temp_OP2 >> 1;

temp_OP2 = temp_OP2 | (LSB_A << 8);     // LSB of temp_prod (A) assigned to MSB of temp_OP2 (Q)
Q_n1 = LSB_Q;                           // Q_n1 receives LSB of temp_OP2
temp_prod = temp_prod | (MSB_A << 8);   // LSB of temp_prod (A) assigned to MSB of temp_prod (A) afer shifting
...
...
...
```

Sample Echo on Console:

- Display operand1 and operand2 then the operation.
- Display the temp_prod (A), temp_OP2 (Q), Q_n1, operand1 (M) and n.
- Display each cycle with update values.
- Display ACC.
- See example output below.



```
************************************
Fetching operands...
OP1: 11000000
OP2: 00001010
Operation: MUL
    A        Q    Q_n1     M     n
00000000 00001010  0  11000000 0
00000000 00000101  0  11000000 1
00100000 00000010  1  11000000 2
11110000 00000001  0  11000000 3
00011000 00000000  1  11000000 4
11101100 00000000  0  11000000 5
11110110 00000000  0  11000000 6
11111011 00000000  0  11000000 7
11111101 10000000  0  11000000 8
ACC: 1111110110000000
ZF=0 SF=1 OF=1 CF=1
************************************
Fetching operands...
OP1: 01000110
OP2: 00000010
Operation: MUL
    A        Q    Q_n1     M     n
00000000 00000010  0  01000110 0
00000000 00000001  0  01000110 1
11011101 00000000  1  01000110 2
00010001 10000000  0  01000110 3
00001000 11000000  0  01000110 4
00000100 01100000  0  01000110 5
00000010 00110000  0  01000110 6
00000001 00011000  0  01000110 7
00000000 10001100  0  01000110 8
ACC: 0000000010001100
ZF=0 SF=1 OF=1 CF=0
```

Example output of:
ALU(0xC0,0x0A,0x03)
ALU(0x46, 0x02,0x03)

Note:

- ACC is displayed as 16-bit to check the overflow and carry condition. It also allows checking
- Display the temp_prod (A), temp_OP2 (Q), Q_n1, operand1 (M) and n.
- Display each cycle with update values.
- Display ACC.
- See example output below.