```cpp
/*
 Erick Castro
 CSCI 113 - Simulation Program

 Algorithm
 1. if MQ[0] == 1
 AC = AC + MD
 2. AC/MQ >> 1 - Shift right 1 bit

 How to Use:
 To enter values for MD and MQ, there are two vectors in main.
 Slot 0 in vector is 0th spot in 16 bit number.
 Enter number from left to right.

 Notes:
 16Bit ALU is done by calling 1bit alu 16 times without a loop, which might not be an efficient way to do it.
 i tried to use a for loop, but i wasn't able to get it working for some reason.


*/

#include <iostream>
#include <array>
#include <vector>
#include <algorithm>
using namespace std;


//AND Gate
bool AND(bool x, bool y){
    if(x == 1 && y ==1)
         return 1;
    else
         return 0;
}

//OR Gate
bool OR(bool x, bool y){
    if(x == 1 || y == 1)
         return 1;
    else
        return 0;
}

//XOR Gate
bool XOR(bool x, bool y){
    if(x == 1 && y != 1)
        return 1;
    else if(x != 1 && y == 1)
        return 1;
    else
```

```
52          return 0;
53  }
54
55  //4x1 MUX
56  bool MUX_4x1(bool x1, bool x2, bool x3, bool x4, int op){
57      if (op == 0)
58          return x1;
59      else if(op == 1)
60          return x2;
61      else if(op == 2)
62          return x3;
63      else
64          return x4;
65
66  }
67  // 1-bit Full Adder
68  bool full_adder_1bit(bool a, bool b, bool c_in, bool &c_out){
69      c_out = OR(AND(a,b), AND(c_in,XOR(a,b)));
70      return XOR(XOR(a,b), c_in);
71  }
72
73  /*
74   1-bit ALU
75
76   3 Operations:
77      AND: OP = 0
78      OR:  OP = 1
79      ADD: OP = 2
80  */
81  bool ALU_1bit(bool a, bool b, bool c_in, int op, bool  &c_out){
82
83      bool t1 = AND(a,b);
84      bool t2 = OR(a,b);
85      bool t3 = full_adder_1bit(a, b, c_in, c_out);
86      bool t4 = 0;
87
88      return MUX_4x1(t1, t2, t3, t4, op);
89  }
90
91  /*
92   1-bit ALU with Overflow check
93   Same as 1-bit ALU with 3 operations
94   Additional Overflow check:
95      Operand sign != Result Sign -> OF
96  */
97  bool ALU_1bit_OF(bool a, bool b, bool c_in, int op, bool &of){
98
99      bool c_out;
100     bool t1 = AND(a,b);
101     bool t2 = OR(a,b);
102     int t3 = full_adder_1bit(a, b, c_in, c_out);
103     bool t4 = 0;
```

```
104
105     of = XOR(c_in, c_out);
106
107     return MUX_4x1(t1, t2, t3, t4, op);
108 }
109
110 /*
111 16Bit Alu
112 Does any operation for two 16bit numbers.
113 */
114 void ALU_16bit(vector<bool> a, vector<bool> b, int op, vector<bool> &result, bool ↵
       &of){
115     bool t[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,0 };
116
117     result[0] = ALU_1bit(a[0], b[0], 0, op, t[0]);
118     result[1] = ALU_1bit(a[1], b[1], t[0], op, t[1]);
119     result[2] = ALU_1bit(a[2], b[2], t[1], op, t[2]);
120     result[3] = ALU_1bit(a[3], b[3], t[2], op, t[3]);
121     result[4] = ALU_1bit(a[4], b[4], t[3], op, t[4]);
122     result[5] = ALU_1bit(a[5], b[5], t[4], op, t[5]);
123     result[6] = ALU_1bit(a[6], b[6], t[5], op, t[6]);
124     result[7] = ALU_1bit(a[7], b[7], t[6], op, t[7]);
125     result[8] = ALU_1bit(a[8], b[8], t[7], op, t[8]);
126     result[9] = ALU_1bit(a[9], b[9], t[8], op, t[9]);
127     result[10] = ALU_1bit(a[10], b[10], t[9], op, t[10]);
128     result[11] = ALU_1bit(a[11], b[11], t[10], op, t[11]);
129     result[12] = ALU_1bit(a[12], b[12], t[11], op, t[12]);
130     result[13] = ALU_1bit(a[13], b[13], t[12], op, t[13]);
131     result[14] = ALU_1bit(a[14], b[14], t[13], op, t[14]);
132     result[15] = ALU_1bit_OF(a[15], b[15], t[14], op, of);
133 }
134
135 /*
136 Cycle Counter
137
138 Counts down by 1 each cycle using ADD operation from 1 bit alu.
139 Subtraction is done by using 1's complement of 1 and a CIN of 1.
140 */
141 vector<bool> cycleCounter(vector<bool> &count)
142 {
143     bool t[] = { 0, 0, 0, 0, 0 };
144     vector<bool> result(5);
145
146     result[0] = ALU_1bit(count[0], 0, 1, 2, t[0]);
147     result[1] = ALU_1bit(count[1], 1, t[0], 2, t[1]);
148     result[2] = ALU_1bit(count[2], 1, t[1], 2, t[2]);
149     result[3] = ALU_1bit(count[3], 1, t[2], 2, t[3]);
150     result[4] = ALU_1bit(count[4], 1, t[3], 2, t[4]);
151
152     return result;
153 }
154
```

```
155 /*
156  16-Bit Multiplier
157
158  Uses 16-Bit ALU to do 16-bit multiplication.
159  Inputs:
160     MD
161     MQ
162  Outputs:
163     PR - Product
164 */
165 void Mult_16bit(vector<bool> MD, vector<bool> MQ, vector<bool> &PR){
166     vector<bool> AC = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
167     vector<bool> count = { 0,0,0,0,1 };
168
169     //Prints out values before any steps are done
170     cout << "Counter     " << "MD               " << "AC               " << "MQ"
          << endl;
171     for (long i = count.size() - 1; i >= 0; i--)
172         cout << count[i];
173     cout << "       ";
174     for (long i = MD.size() - 1; i >= 0; i--)
175         cout << MD[i];
176     cout << " ";
177     for (long i = AC.size() - 1; i >= 0; i--)
178         cout << AC[i];
179     cout << " ";
180     for (long i = MQ.size() - 1; i >= 0; i--)
181         cout << MQ[i];
182     cout << endl;
183
184     bool of;
185
186     //Loop that does main operation, 16 times, or size of MD, which should be 16.
187     for(int i = 0; i < MD.size(); i++){
188         if(MQ[0] == 1)
189             ALU_16bit(AC, MD, 2, AC, of);         //OP set to 2 for ADD
                  operation.
190
191         if(i > 0)
192             count = cycleCounter(count);
193
194         //Prints out current values after step 1.
195         for (long i = count.size() - 1; i >= 0; i--)
196             cout << count[i];
197         cout << "       ";
198         for (long i = MD.size() - 1; i >= 0; i--)
199             cout << MD[i];
200         cout << " ";
201         for (long i = AC.size() - 1; i >= 0; i--)
202             cout << AC[i];
203         cout << " ";
204         for (long i = MQ.size() - 1; i >= 0; i--)
```

```
205             cout << MQ[i];
206         cout << " --Step 1" << endl;
207
208         //Shifts vector to the right by 1 bit
209         MQ.insert(MQ.end(), AC[0]);
210         MQ.erase(MQ.begin());
211         //Shifts vector to the right by 1 bit
212         AC.insert(AC.end(), 0);
213         AC.erase(AC.begin());
214
215         //Prints out current values after step 2.
216         for (long i = count.size() - 1; i >= 0; i--)
217             cout << count[i];
218         cout << "       ";
219         for (long i = MD.size() - 1; i >= 0; i--)
220             cout << MD[i];
221         cout << " ";
222         for (long i = AC.size() - 1; i >= 0; i--)
223             cout << AC[i];
224         cout << " ";
225         for (long i = MQ.size() - 1; i >= 0; i--)
226             cout << MQ[i];
227         cout <<  " --Step 2" << endl << endl;
228
229     }
230
231     //Concatenates MQ and AC into one vector, PR, the product vector.
232     PR.insert(PR.begin(), MQ.begin(), MQ.end());
233     PR.insert(PR.end(), AC.begin(), AC.end());
234 }
235
236
237
238 int main()
239 {
240     vector<bool> MD = {1,0,0,1,0,0,0,0,1,0,1,0,0,0,0,0};    //MD
241     vector<bool> MQ = {0,1,1,0,0,1,1,0,0,1,1,0,0,0,0,1};    //MQ
242     vector<bool> PR;                                        //Product
243
244     Mult_16bit(MD, MQ, PR);                                 //Calls the main 16
        bit Multiplier to do operation on MD and MQ, resulting in PR
245
246     cout << "Product: ";
247     for(long i = PR.size()-1; i >= 0; i--)                  //Prints out Product
248         cout << PR[i];
249     cout << endl;
250
251
252     system("pause");
253     return 0;
254 }
255
```