ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

HOÀNG DÂN AN – 1612001

NGUYỄN PHƯỚC AN – 1612009

NGUYỄN HOÀNG ANH – 1612023

TRƯƠNG THÀNH DANH - 1612085

HOÀNG HẢI GIANG – 1612154

ĐỒ ÁN GIỮA KỲ

BIỂU DIỄN VÀ TÍNH TOÁN SỐ HỌC TRÊN MÁY TÍNH

Lớp: Kiến trúc máy tính & Hợp ngữ 16_1

Thành phố Hồ Chí Minh – 2017

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

HOÀNG DÂN AN – 1612001

NGUYỄN PHƯỚC AN – 1612009

NGUYỄN HOÀNG ANH – 1612023

TRƯƠNG THÀNH DANH - 1612085

HOÀNG HẢI GIANG – 1612154

ĐỒ ÁN GIỮA KÌ

Đề tài

Biểu diễn và tính toán số học trên máy tính

Giảng viên hướng dẫn

Lê Viết Long

Lớp: Kiến trúc máy tính & Hợp ngữ 16_1

Thành phố Hồ Chí Minh – 2017

LỜI CẢM ƠN

MỤC LỤC

I. SƠ LƯỢC VỀ BIỂU DIỄN SỐ NGUYÊN

1. Hệ cơ số q tổng quát
   - Tổng quát số nguyên có n chữ số thuộc hệ cơ số q bất kỳ được biểu diễn:

$$x_{n-1}...x_1x_0 = x_{n-1}.q^{n-1} + ... + x_1.q^1 + x_0.q^0$$

   (mỗi chữ số xi lấy từ tập X có q phần tử)
   - Ví dụ:
     – Hệ cơ số 10: A = 123 = 100 + 20 + 3 = $1.10^2 + 2.10^1 + 3.10^0$
     – q = 2, X = {0, 1}: hệ nhị phân (binary)
     – q = 8, X = {0, 1, 2,…, 7}: hệ bát phân (octal)
     – q = 10, X = {0, 1, 2,…, 9}: hệ thập phân (decimal)
     – q = 16, X = {0, 1, 2,…,9, A, B,…, F}: hệ thập lục phân (hexadecimal)
   - Chuyển đổi: A = 123 d = 01111011 b = 173 o = 7B h
   - Hệ cơ số thường được biển diễn trong máy tính là hệ cơ số 2
2. Chuyển đổi giữa các hệ cơ số
   a) Từ hệ thập phân (10) sang hệ nhị phân (2)
      - Lấy số cơ số 10 chia cho 2 → Số dư đưa vào kết quả → Số nguyên đem chia tiếp cho 2 → Quá trình lặp lại cho đến khi số nguyên = 0
   b) Từ hệ thập phân (10) sang hệ thập lục phân (16)
      - Lấy số cơ số 10 chia cho 16 → Số dư đưa vào kết quả → Số nguyên đem chia tiếp cho 16 → Quá trình lặp lại cho đến khi số nguyên = 0
   c) Từ hệ nhị phân (2) sang hệ thập phân (10)
      - Khai triển biểu diễn và tính giá trị biểu thức

$$\boxed{x_{n-1}...x_1x_0 = x_{n-1}.2^{n-1} + ... + x_1.2^1 + x_0.2^0}$$

   d) Từ hệ nhị phân (2) sang hệ thập lục phân (16)
      - Nhóm từng bộ 4 bit trong biểu diễn nhị phân rồi chuyển sang ký số tương ứng trong hệ thập lục phân (0000 → 0,…, 1111 → F)

e) Từ hệ thập lục phân (16) sang hệ nhị phân (2)

- Sử dụng bảng dưới đây để chuyển đổi

HEX	BIN	HEX	BIN	HEX	BIN	HEX	BIN
0	0000	4	0100	8	1000	C`	1100
1	0001	5	0101	9	1001	D	1101
2	0010	6	0110	A	1010	E	1110
3	0011	7	0111	B	1011	F	1111

f) Từ hệ thập lục phân (16) sang hệ thập phân (10)

- Khai triển biểu diễn và tính giá trị biểu thức

$$x_{n-1}...x_1x_0 = x_{n-1}.16^{n-1} + ... + x_1.16^1 + x_0.16^0$$

3. Hệ nhị phân

$$x_{n-1}...x_1x_0 = x_{n-1}.2^{n-1} + ... + x_1.2^1 + x_0.2^0$$

- Được dùng nhiều trong máy tính để biểu diện các giá trị lưu trong các thanh ghi hoặc trong các ô nhớ. Thanh ghi hoặc ô nhớ có kích thước 1byte (8 bit) hoặc 1 word (16 bit).
- n được gọi là chiều dài bit của số đó
- Bit trái nhất x_n-1 là bit có giá trị (nặng) nhất MSB (Most Significant Bit)
- Bit phải nhất x_0 là bit ít giá trị (nhẹ) nhất LSB (Less Significant Bit)

4. Số nguyên có dấu

- Lưu các số dương hoặc âm (số có dấu) có 4 cách phổ biến:
  – Dấu lượng
  – Bù 1
  – Bù 2
  – Số quá (thừa) K

 - Số có dấu trong máy tính được biểu diễn ở dạng số bù 2

a) Dấu lượng

- Bit trái nhất (MSB): bit đánh dấu âm / dương
  – 0: số dương
  – 1: số âm
- Các bit còn lại: biểu diễn độ lớn của số (hay giá trị tuyệt đối của số)
- Ví dụ:
  – Một byte 8 bit: sẽ có 7 bit (trừ đi bit dấu) dùng để biểu diễn giá trị tuyệt đối cho các số có giá trị từ 0000000 (0_{10}) đến 1111111 (127_{10})
  ➔ Ta có thể biểu diễn các số từ -127_{10} đến $+127_{10}$
- -N và N chỉ khác giá trị bit MSB (bit dấu), phần độ lớn (giá trị tuyệt đối) hoàn toàn giống nhau

b) Bù 1

- Tương tự như phương pháp dấu lượng, bit MSB dùng làm bit dấu
  – 0: Số dương
  – 1: Số âm
- Các bit còn lại để biểu diễn giá trị (*)
- Số âm: thực hiện tất cả các phép đảo bit của (*)

c) Bù 2

- Biểu diễn giống như số bù 1 + ta phải cộng thêm số 1 vào kết quả (dạng nhị phân)
- Số bù 2 ra đời khi người ta gặp vấn đề với hai phương pháp dấu lượng và bù 1, đó là:
  – Có hai cách biểu diễn cho số 0 (+0 và -0) → không đồng nhất
  – Bit nhớ phát sinh sau khi đã thực hiện phép tính phải được cộng tiếp vào kết quả → dễ gây nhầm lẫn → Phương pháp số bù 2 khắc phục hoàn toàn 2 vấn đề đó

d) Số quá k

- Còn gọi là biểu diễn số dịch (biased representation)
- Chọn một số nguyên dương K cho trước làm giá trị dịch

- K từ -2^{127} đến 2^{127}
- Biểu diễn số N:

  +N (dương): có được bằng cách lấy K + N, với K được chọn sao cho tổng của K và một số âm bất kỳ trong miền giá trị luôn luôn dương

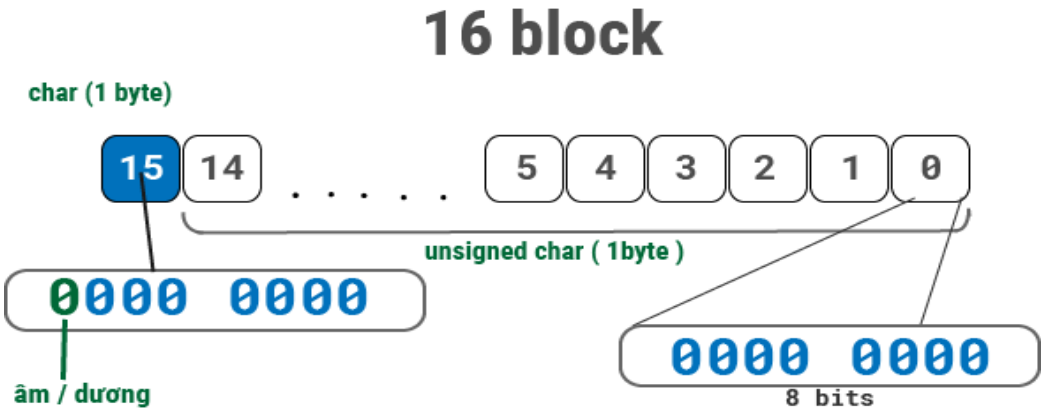
  -N (âm): có được bằng cách lấy K - N (hay lấy bù hai của số vừa xác định)

II. THUẬT TOÁN

1. Mô tả cách thức thuật toán:

a) Nguyên lý lưu thông tin số nguyên lớn của đồ án :

- Chia 128 bits dữ liệu thành 1 mảng 16 block độ lớn 1 byte (unsigned char)
- Riêng block cao nhất được định nghĩa là [char] để có thể dễ nhận dạng số âm.
- Bit cao nhất (tức là bit cao nhất của block thứ 15) dùng để đánh dấu số âm.
- Lưu số nguyên theo định nghĩa số bias bù 2.



b) Chuyển chuỗi thành số nguyên lớn (16 bytes) :

(*) Chuỗi ký tự bit

- Ví dụ: “10101011”, “00111001”, ….
- Mỗi kí tự có 2 giá trị là ‘0’ và ‘1’

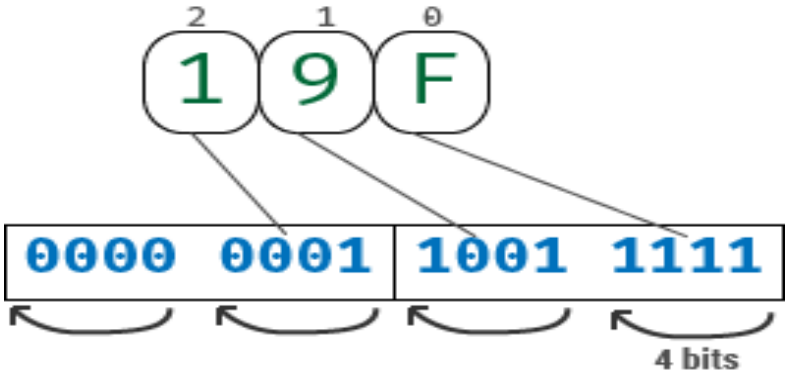
- Ý tưởng :
  - Kí tự thứ n trong chuỗi sẽ là giá trị của bit thứ n trong số nguyên
  - Do ta chia số nguyên thành 16 block nên ta cần phải tính toán vị trí bit trong số ứng
  - Sử dụng các phép logic để gán bit đó vào vị trí cụ thể của block.

```
Bit = ChuoiBin[len - Position - 1];
Vị_Tri_Block = Position / 8;
Vi_Tri_Trong_Block = Position % 8;
//Su dung phep toan ^ de gan bit vao 1 so
```

- Block[Vi_Tri_Block] = Block[Vi_Tri_Block] ^ (Bit Vi_Tri_Trong_Block);

c) Chuỗi kí tự hex :

- Ví dụ : ”00FFAA”, “13B”, …
- Mỗi kí tự là một số tương ứng trong hệ số thập lục phân (1,2,3,4,5,6,7,8,9,A,B,C,D,E,F)



- Ý tưởng giống với chuyển chuỗi kí tự bit nhưng :
  - Giá trị ở hệ số n trong chuỗi sẽ tương ứng với 4 bits thứ n trong số nguyên 16 bytes.
  - Sử dụng các phép logic để gán 4 bit vào vị trí cụm 4 bit thứ n.

```
Hex = chuoiHex[position];
Vi_tri_block = position / 2;
Vi_tri_trong_block = position % 2;
Block[vi_tri_block] = block[vi_tri_block] ^ (hex << 4*(vi_tri_trong_block));
```

a. Chuỗi thập phân :

- Ví dụ :”12345”,”93”, …
- Mỗi kí tự là số 0 – 9.
- Do sơ cấp 10 không là bội ước của 255, nên việc chia thành từng dãy bits không dễ dàng. Cách duy nhất chúng ta có thể làm được là “chia 2” chuỗi số dần để lấy số dư – là bit thứ 2n của chuỗi số đó.

- Ý tưởng :
  - Thiết lập thuật toán "chia 2" chuỗi số và xuất ra số dư
  - Chia dần chuỗi số, tìm số dư
  - Gán bit (là số dư tìm được) vào vị trí tăng dần khi chia.

```
while (chuoiSo != "0") {
    chuoiSo = chiaChuoiCho2(chuoiSo, sodu);

    //tim vi tri cua bit
    int vitri_trong_bit = pos % 8;
    int vitri_trong_mang = pos / 8;

    // su dung phep ^ de gan 1bit vao vi tri thu pos
    body[vitri_trong_mang] ^= (sodu << vitri_trong_bit);
    pos++;
}
```

Trong hình: **body** là *dãy block*, **pos** là vị trí đếm từ 0 lên.

b. Hàm chuyển đổi thập phân xang nhị phân :
   - Đem chuỗi ***chia*** 2 lấy dư
   - Số dư đưa vào kết quả
   - Qúa trình lập lại cho đến khi số nguyên bằng 0.

c. Hàm chuyển đổi nhị phân xang thập phân :
   - Chuỗi kết quả ban đầu là 0
   - Số x là vị trí bit cao nhất (khác bit dấu)
   - Đem chuỗi nhân 2 công giá trị bit thứ x
   - Giảm x xuống 1 đơn vị
   - Lập lại cho đến khi x bằng 0.

d. Hàm chuyển đổi nhị phân xang thập lục phân :
   - Chọn 4 bit từ 8 bit của 1 ô rồi so sánh với bảng đã biết

e. Toán tử operator " + " (cộng)
   - Đem block thứ i của số A cộng cho block thứ i của số B
   - Phần tràn đem cộng vào block i + 1 của kết quả
   - Cộng hết các block tương ứng với nhau

- Do ta sử dụng nguyên tắc số bias thì phần tràn lớn nhất sẽ bị bỏ đi.

f. Toán tử operator ”- ” (trừ)

- Sử dụng nguyên tắt A – B = A + Negative(B)
- Sử dụng lại toán tử cộng A với Negative(B)

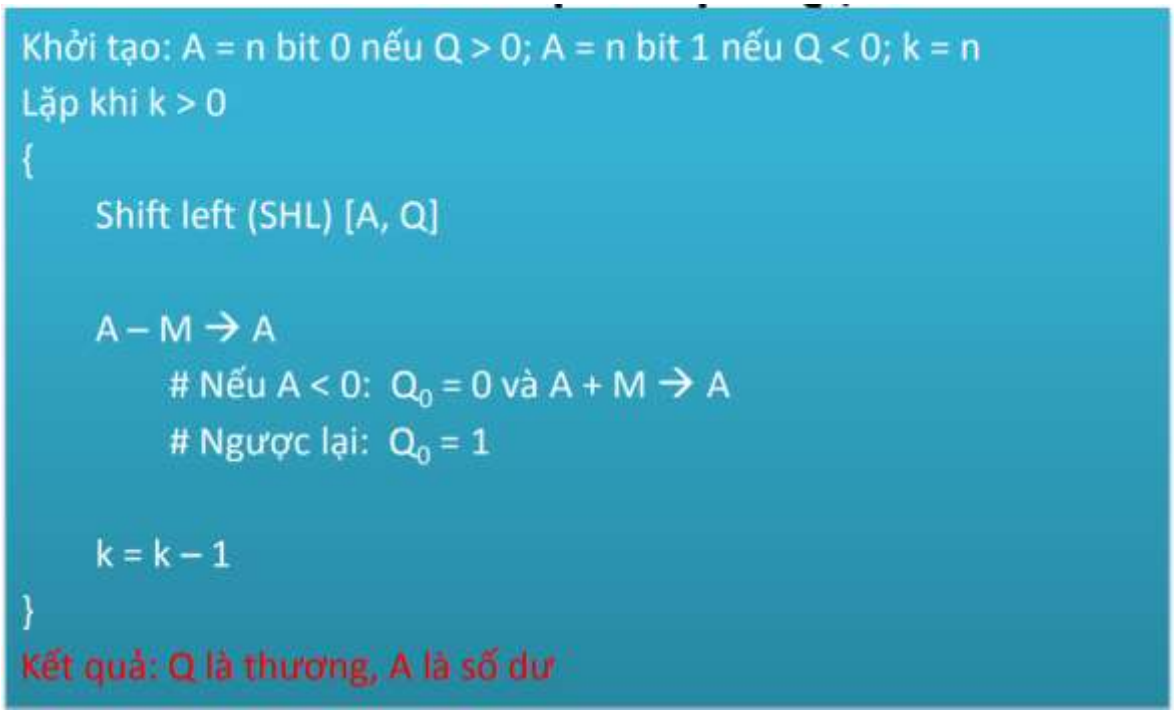
g. Toán tử operator ” * ” (phép nhân)

- Ta có $A = A_n \ldots A_1 A_0 = A_n.Q^n + \ldots + A_1.Q^1 + A_0.Q^0$
- $B = B_m \ldots B_1 B_0 = B_m.B^m + \ldots + B_1.Q^1 + B_0.Q^0$
- Đem A*B ta thấy:
  - Ứng với hệ số Q^0 ta có $A_0 * B_0$
  - Ứng với hệ số Q^1 ta có $A_1 * B_0 + A_0 * B_1$
  - Ứng với hệ số Q^2 ta có $A_2B_0 + A_1B_1 + A_0B_2$
  - …
  - Ứng với hệ số Q_{i+j} ta có $\sum A_i * B_j$
- Từ đó, ta suy ra kết quả C có
  - **C_k = Tổng(A_i*B_j)** sao cho **i + j = k**
  - Phần dư của C_k (phần tràn lên hệ số Q^{k+1}) sẽ cộng dồn lên C_{k+1} phía sau.
- Thuật toán mô tả: (Với Q = 255 - do mỗi block của ta là 1 byte)

```
BigInt BigInt::operator*(const BigInt &A)const {
    BigInt result;

    int tmp = 0,
        phan_tran   = 0;
    for (int i = 0; i < 16; i++)
    {
        tmp = 0;
        for (int j = 0; j <= i; j++)
            tmp += this->body[j] * A.body[i - j]; // A[j]*B[i-j] = C[i]

        tmp += phan_tran; //phan tran nho o lan truoc

        result.body[i] = tmp % 256;
        phan_tran = tmp / 256;
    }

    return result;
}
```

h. Toán tử operator ” / ” (phép chia)

- Sử dụng công thức dịch bit và cộng trừ có trong Tài liệu tham khảo



i. Toán tử "&", "|", "^":
   - Đem từng block thực hiện theo phép toán tương ứng

j. Toán tử NOT "~"
   - Đem tất cả các block XOR với 0xFF (1111 1111).

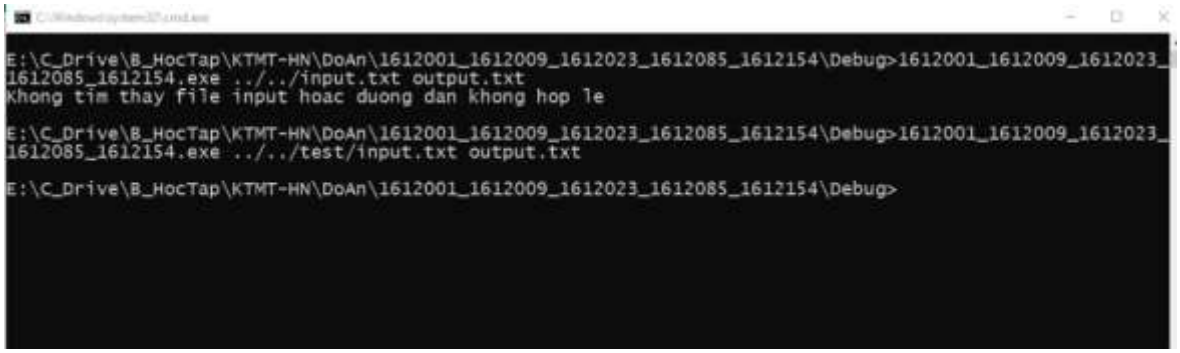
III. TESTING

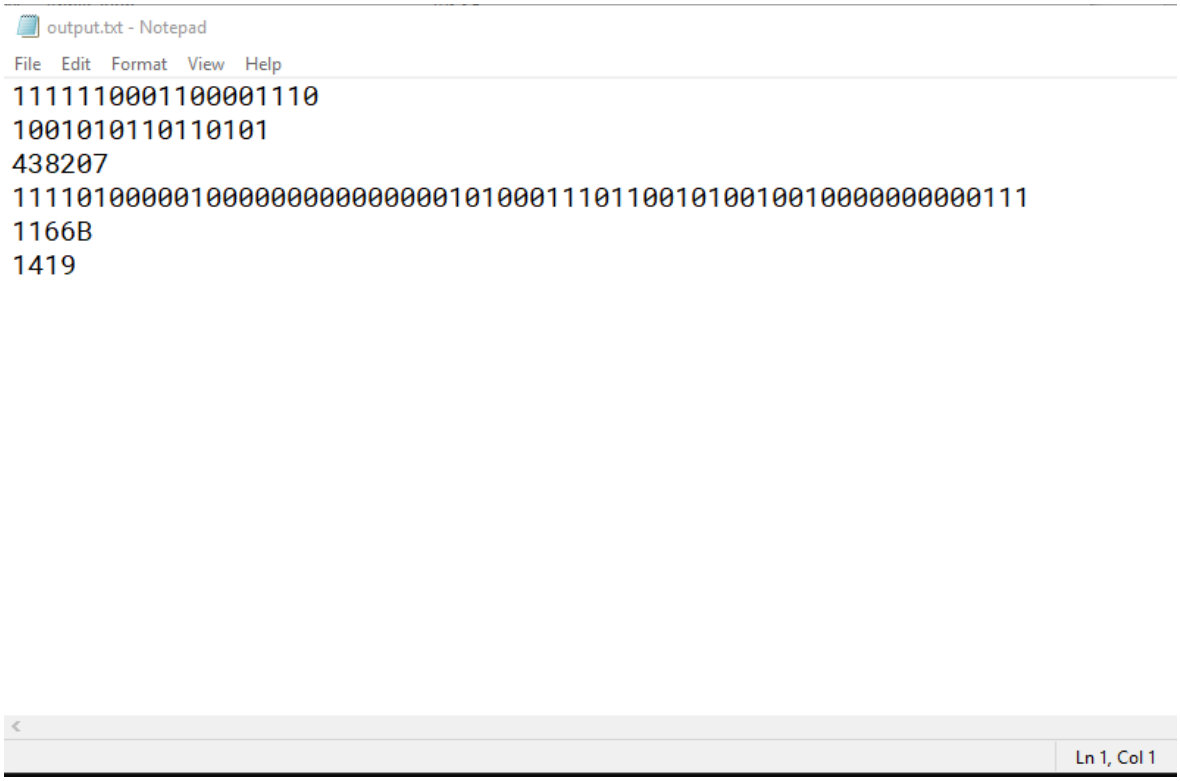
1) Test theo ví dụ đề bài :

a. Input

INPUT.TXT	OUTPUT.TXT
2 1111100011101010111 + 01101110110111	1111110001100001110
2 11011011 * 010101111	1001010110110101
2 10 01101010111110111111	438207
10 2 8793278316383117319	111 01000001000000000000000001010001110 11001010010010000000000111
16 85AF + 90BC	1166B
10 5678 >> 2	1419

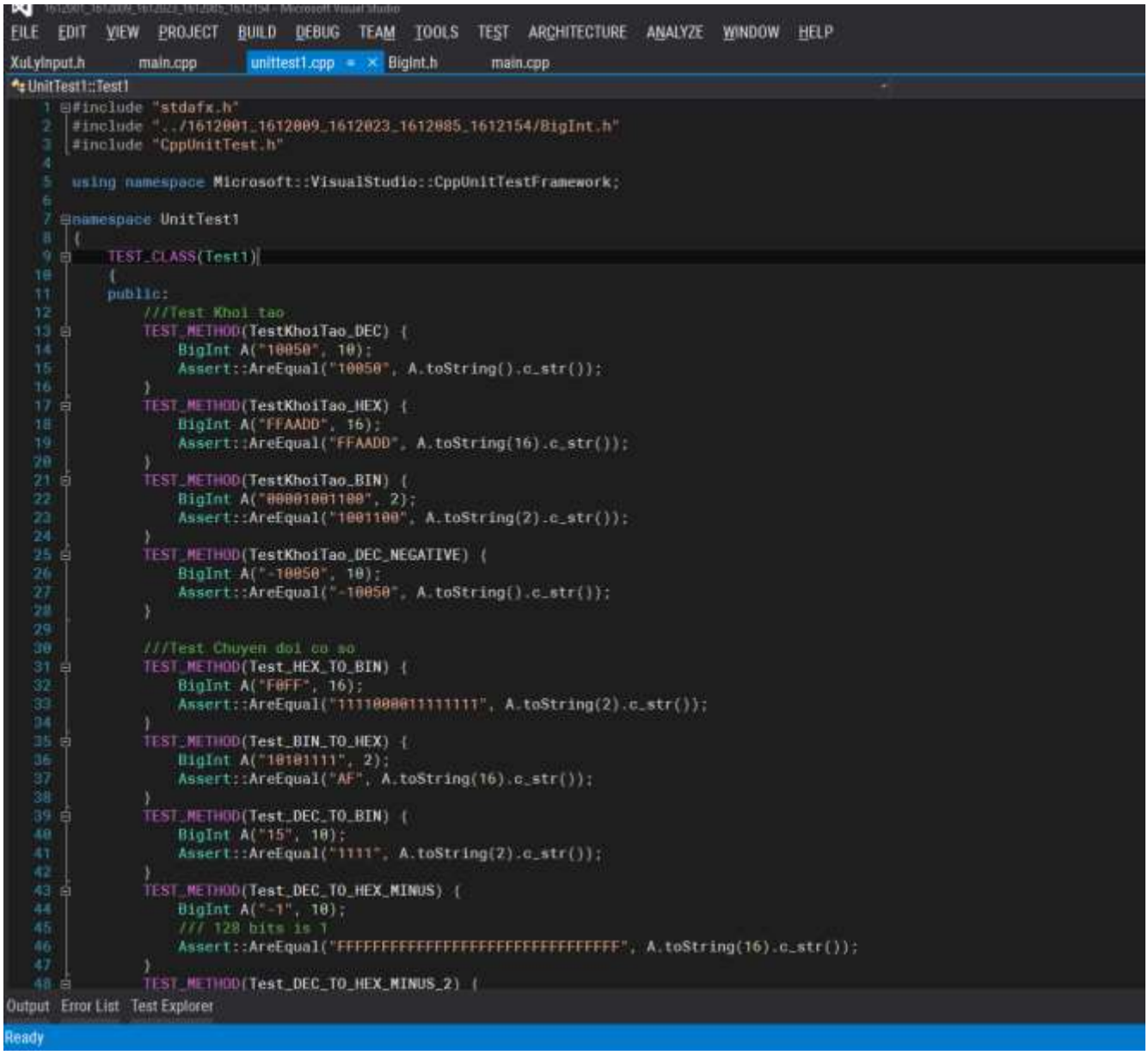
b. Output

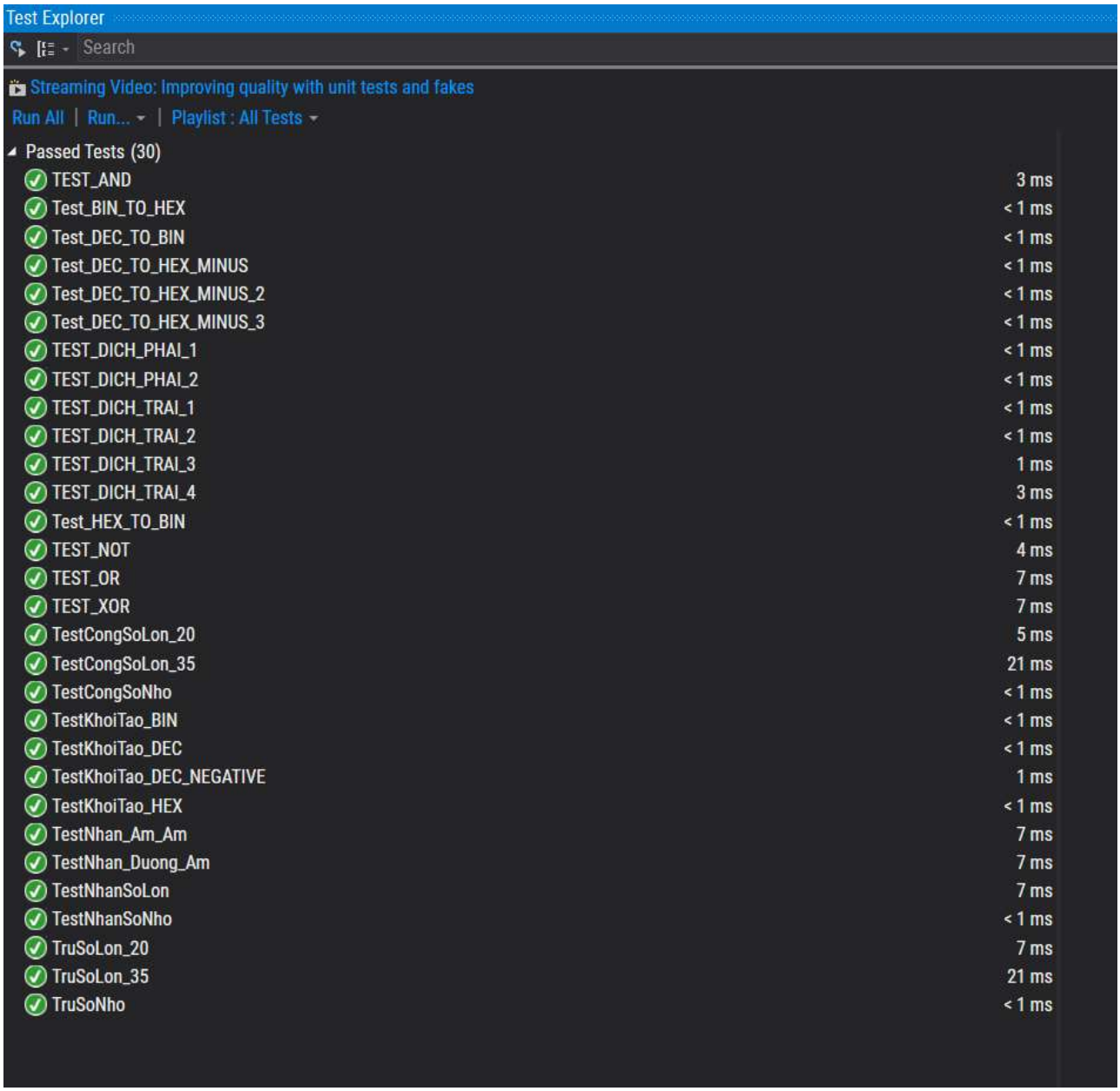




2) **Test toàn diện:**

- Sử dụng framework Unit Test có sẵn của Visual Studio để test các chức năng và phương thức của class BigInt.
- Các hình thức test:
  - Test Khởi tạo
  - Test phép toán đại số
  - Test phép toán logic
  - Test các bộ số lớn
  - Test các bộ số âm
- Nguồn kết quả tin cậy để kiểm tra
  - Các bộ test sẽ tạo dựa trên các kết quả trên trang web https://defuse.ca/big-number-calculator.htm
  - Các bộ test tự cho, lấy kết quả để tạo bộ test
- Screenshots:


FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST ARCHITECTURE ANALYZE WINDOW HELP
XuLyInput.h
main.cpp
unittest1.cpp
BigInt.h
main.cpp
UnitTest1::Test1
#include "stdafx.h"
#include "../1612001_1612009_1612023_1612085_1612154/BigInt.h"
#include "CppUnitTest.h"
using namespace Microsoft::VisualStudio::CppUnitTestFramework;
namespace UnitTest1
{
TEST_CLASS(Test1)
{
public:
///Test Khoi tao
TEST_METHOD(TestKhoiTao_DEC) {
BigInt A("10050", 10);
Assert::AreEqual("10050", A.toString().c_str());
}
TEST_METHOD(TestKhoiTao_HEX) {
BigInt A("FFAADD", 16);
Assert::AreEqual("FFAADD", A.toString(16).c_str());
}
TEST_METHOD(TestKhoiTao_BIN) {
BigInt A("00001001100", 2);
Assert::AreEqual("1001100", A.toString(2).c_str());
}
TEST_METHOD(TestKhoiTao_DEC_NEGATIVE) {
BigInt A("-10050", 10);
Assert::AreEqual("-10050", A.toString().c_str());
}
///Test Chuyen doi co so
TEST_METHOD(Test_HEX_TO_BIN) {
BigInt A("F0FF", 16);
Assert::AreEqual("1111000011111111", A.toString(2).c_str());
}
TEST_METHOD(Test_BIN_TO_HEX) {
BigInt A("10101111", 2);
Assert::AreEqual("AF", A.toString(16).c_str());
}
TEST_METHOD(Test_DEC_TO_BIN) {
BigInt A("15", 10);
Assert::AreEqual("1111", A.toString(2).c_str());
}
TEST_METHOD(Test_DEC_TO_HEX_MINUS) {
BigInt A("-1", 10);
/// 128 bits is 1
Assert::AreEqual("FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF", A.toString(16).c_str());
}
TEST_METHOD(Test_DEC_TO_HEX_MINUS_2) {
Output Error List Test Explorer
Ready


Test Explorer
Search
Streaming Video: Improving quality with unit tests and fakes
Run All | Run... | Playlist : All Tests
Passed Tests (30)
TEST_AND 3 ms
Test_BIN_TO_HEX < 1 ms
Test_DEC_TO_BIN < 1 ms
Test_DEC_TO_HEX_MINUS < 1 ms
Test_DEC_TO_HEX_MINUS_2 < 1 ms
Test_DEC_TO_HEX_MINUS_3 < 1 ms
TEST_DICH_PHAI_1 < 1 ms
TEST_DICH_PHAI_2 < 1 ms
TEST_DICH_TRAI_1 < 1 ms
TEST_DICH_TRAI_2 < 1 ms
TEST_DICH_TRAI_3 1 ms
TEST_DICH_TRAI_4 3 ms
Test_HEX_TO_BIN < 1 ms
TEST_NOT 4 ms
TEST_OR 7 ms
TEST_XOR 7 ms
TestCongSoLon_20 5 ms
TestCongSoLon_35 21 ms
TestCongSoNho < 1 ms
TestKhoiTao_BIN < 1 ms
TestKhoiTao_DEC < 1 ms
TestKhoiTao_DEC_NEGATIVE 1 ms
TestKhoiTao_HEX < 1 ms
TestNhan_Am_Am 7 ms
TestNhan_Duong_Am 7 ms
TestNhanSoLon 7 ms
TestNhanSoNho < 1 ms
TruSoLon_20 7 ms
TruSoLon_35 21 ms
TruSoNho < 1 ms

IV. ĐÁNH GIÁ

STT	Mục đánh giá	Mức độ hoàn thành
1	Lưu trữ trên đủ 16 bytes	100%
2	Nhập và xuất	90%
3	Các phép toán đại số	100%
4	Các phép toán logic	100%
5	Chuyển đổi cơ số	100%
6	Dịch trái dịch phải	100%
7	Biểu diễn được số âm	100%
8	Xử lý lỗi logic và đại số	10%
9	Xử lý đọc ghi file bằng command line	100%
10	Source code và báo cáo rõ ràng, dễ hiểu	80%
Tổng cộng		**88%**

TÀI LIỆU THAM KHẢO

- **Slide biểu diễn số nguyên**
- **Stackoverflow**