/*
   This program returns the greatest common divisor of two integers. Two procedures are provided: one is implemented in C and the other is implemented in inline code

   author: eric li
*/

#include <stdio.h>
#include <math.h>

int ASMGCD(int x, int y);
int GCD(int x, int y);

int main()
{
  int x, y, out;

  printf("Please enter two integers: \n");
  scanf("%d%d", &x, &y);
  out = GCD(x,y);
  printf("C: Greatest common divisor of two numbers %d and %d is %d.\n\n", x, y, out);
  out = ASMGCD(x,y);
  printf("ASM Greatest common divisor of two numbers %d and %d is %d.\n\n", x, y, out);

  return 0;
}
/* C-implementation */
int GCD(int x, int y)
{
    x = abs(x);
y = abs(y);
do {
        int n = x % y;
x = y;
y = n;
    } while (y > 0);
return x;
}

/* inline assembly code */
int ASMGCD(int x, int y)
{
    int n;
    __asm {
        IF01:
            mov eax, x;
            cmp eax, 0;
            jg ENDIF01;
            neg eax;  // absolute value
            mov x, eax;
        ENDIF01:
        IF02:
            mov eax, y; cmp eax, 0;
            jg ENDF02;
            neg eax;  // absolute value
            mov y, eax;
        ENDF02:
        DO01:
            mov eax, x;  // numerator
            mov ebx, y;  // a reg used for denom
            sub edx, edx;  // zero dx
            idiv ebx;    // signed integer divide
            mov n, edx;  // remainder
            mov ebx, y;
            xchg ebx, x;  // x = y
            mov ebx, n;
            xchg ebx, y;  // y = n;
        DOEND01:  // while (y > 0)
            cmp y, 0;
            jg DO01;
        } return x;
    }
}

Example 2: Prime Number

Write a procedure to test whether an integer is a prime number or not. The procedure returns 0 if the integer is a prime and returns non-zero if it is not. A prime number is a natural number that has exactly two natural number divisors, which are 1 and the prime number itself.
/* The procedure tests if an integer is a prime number. It returns zero if it is a prime number; otherwise, it returns non-zero value */
int isPrime(int x)
{
    int result=0;
    __asm {
        mov eax, 1; // initialise the counter to one
        DO01:
            push eax; // save eax
            mov ecx,eax; // call ASMGCD procedure to
            push ecx // check the greatest common
            mov edx,x; // divisor of the counter and the
            push edx; // input.
call ASMGCD;
        add esp,8;
        IF01:
            cmp eax, 1;       // if the returned value is one, means
            je ELSE01;        // the input is not divisible by the counter;
            pop eax; // otherwise, the input is not a prime, then
            mov result, 1;     // terminate the loop and return one to the
            jmp ENDDO01;  // calling function
        ELSE01:
            pop eax; // to check the next integer value till
            inc eax; // the counter is equal to the input
            cmp eax, x;
            jl DO01;
       ENDIF01:ENDDO01:
    }
    return result;
}

Example 3

• To identify all prime numbers which is less than 100 using the previous two procedures.

/* C-implementation */
for(i=2; i<100; i++) {
    prime = isPrime(i);
    if(prime==0) {
        printf("Integer %d is a prime.\n", i);
    }
}

/* Assembly code */
__asm {
    mov eax, 2; // let eax=2
    FOR01:
        cmp eax, 100; // test eax <100;
        je ENDFOR01;

        mov i, eax; // let i=eax
        push eax; // save eax
        mov edx,eax; // push parameter onto the stack
        push edx;
call isPrime; // call procedure isPrime
    add esp,4;
    mov prime, eax; // returning value
    }}
Example 4: Data Encryption with RSA

- To demonstrate how to encrypt and decrypt data using RSA
- RSA encryption is a method for securing data. It has unique properties that make it especially attractive for securing data that must be transferred among many parties.
- Thus, RSA is still widely used in electronic commerce protocols.

• RSA involves a public and private key:
  - The public key can be known to everyone and is used for the encryption process.
  - The encrypted message can only be decrypted by use of the private key
• A public key consists of a product of two large prime numbers (modulus) and an integer $R$ (public exponent)
• A private key is an integer $S$ (private exponent)
• Assume:
  – Modulus = 33 (i.e. 11x3)
  – R = 7
  – S = 3

• Encryption algorithm:
• Given plain data \( x \),
  \( x = x^R; \)
  \( encrypt_x = x \% MODULUS; \)

• Decryption algorithm:
• Given encrypted data \( y \)
  \( y = y^S; \)
  \( decrypt_y = y \% MODULUS; \)

/*
To demonstrate how RSA encryption works. The
encryption method as well as decryption method is
implemented using inline assemble.

author: eric li
*/

#include <stdio.h>
#include <math.h>
#define PRIME 33
#define R 7
#define S 3

int encrypt(int x);  // encryption procedure prototype
int decrypt(int x);  // decryption procedure prototype

int main()
{
  int array[] = {3, 8, 9, 5, 2, 1};  // original data
  int enarray[6];  // encrypted data
  int dearray[6];  // decrypted data
  int i = 0;  // counter

  for(i=0; i< 6; i++) {
    enarray[i] = encrypt(array[i]);
    dearray[i] = decrypt(enarray[i]);

    printf("Orriginal value: %d	 Encypted value: %d\n", array[i], enarray[i], dearray[i]);
  }
  return 0;
}
Encryption Procedure

/*
x = (int) pow(x, R);
x = x % PRIME;
*/
int encrypt(int x)
{
  __asm {
    mov ecx, x;
    mov eax, x;
    mov bx, word ptr 0;
    Encryption Procedure
    FOR01:
    cmp bx, 6; je ENDFOR01;
    mul ecx;
    inc bx;
    jmp FOR01;
    ENDFOR01:
    mov ebx, PRIME;  // a reg used for denom
    sub edx, edx; // zero dx
    idiv ebx;      // signed integer divide
    mov x, edx;  // remainder
  }
  return x;
}

Decryption Procedure

/*
x = (int) pow(x, S);
x = x % PRIME;
*/
int decrypt(int x)
{
  __asm {
    mov ecx, x;
    mov eax, x;
    mov bx, word ptr 0;
    Decryption Procedure
    FOR01:
    cmp bx, 2; je ENDFOR01;
    mul ecx;
    inc bx;
    jmp FOR01;
    ENDFOR01:
    mov ebx, PRIME;  // a reg used for denom
    sub edx, edx; // zero dx
    idiv ebx;      // signed integer divide
    mov x, edx;  // remainder
  }
  return (x);
}
<table>
<thead>
<tr>
<th>Original value: 3</th>
<th>Encrypted value: 9</th>
<th>Decrypted value: 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original value: 8</td>
<td>Encrypted value: 2</td>
<td>Decrypted value: 8</td>
</tr>
<tr>
<td>Original value: 9</td>
<td>Encrypted value: 15</td>
<td>Decrypted value: 9</td>
</tr>
<tr>
<td>Original value: 5</td>
<td>Encrypted value: 14</td>
<td>Decrypted value: 5</td>
</tr>
<tr>
<td>Original value: 2</td>
<td>Encrypted value: 29</td>
<td>Decrypted value: 2</td>
</tr>
<tr>
<td>Original value: 1</td>
<td>Encrypted value: 1</td>
<td>Decrypted value: 1</td>
</tr>
</tbody>
</table>