Project 3

## Design:

### Overall Design:

### The overall design of the project is the utilization of a C program and two assembly modules to display a table of m mod n for a selection of k values from 2 to 25. Each module has a different method of computing m mod n and the program displays both of their results along with all other relevant data into a table. The c program is used to run the modules and build the table of data. The first module computes m mod n using division. The second program computes m mod n using AND logic, but cannot compute if n is not a power of 2. Whenever this second module cannot compute m mod n it returns -1. The end result is a table of data which includes the current k, m, n, solution using module one, and the solution using module two for k values 2 to 25

### C program:

### The design for my C program was fairly simple as it was only used to execute my modules and display their results. The program displays a small message that tells the user what the program does and also informs the user that the second module returns -1 whenever it is unable to compute the modulus for that particular set of m and n (whenever n is not a power of 2). After displaying these messages the program then runs through a loop of k from 2 to 25 and uses that k value to set the n to 4\*k and m to k. The program then sends this m and n to each of the modules and saves their return values. Then the program displays the current k, n, m, result from program 1, and result from program 2. This continues for all the values of k from 2 to 25 and creates a table of the data. The program then waits for a keypress before terminating.

### Assembly Module 1:

### The first assembly module takes the input given to it from the C program (the current m and n) and uses it to compute m mod n. The program establishes a stack frame and then loads the m and n into registers. EDX is cleared to make room for the remainder. The division function is then used on m and n and the remainder is taken from EDX and moved into EAX. The stack frame is popped and the value in EAX is returned to the waiting C program.

### Assembly Module 2:

### The second assembly module takes the input given to it from the C program (the current m and n) and uses it to compute m mod n, but instead utilizes AND logic to compute the result. Because the module uses AND logic it can only compute the modulus if n is a power of two. First, the program establishes a stack frame and loads n into a register. This n is copied to another register and then decremented in order to get (n-1). ECX is given a value of -1 for the instance in which the result is not a power of two. We now do the AND logic to give us the result of n ^ (n-1). The program checks if this value is zero, and if it is zero this means that the n is a power of two and the modulus can be computed. If the value is not zero then the program skips to the end and returns -1 to indicate to the C program and user that the n was not a power of two. However when n is a power of two the program continues and n and m are moved into registers. The n is decremented to give us (n-1) again and then we do m^(n-1). The result of m^(n-1) is then put into the ecx register to overwrite the -1. The program then moves whatever value is in ECX (a -1 if not a power of two or the result of m mod n) into EAX. The stack frame is popped and the value in EAX is returned to the waiting C program.

## Implementation:

### C program (mod.c):

//M MOD N CALCULATOR

// Made by Cody Otterbine

#include <stdio.h>

extern int modone(int, int);

extern int modtwo(int, int);

int k;

int m;

int n;

int vone;

int vtwo;

int main(void){

int vone;

int vtwo;

int i;

printf("M MOD N CALULATOR \n\n");

printf("This program calculates m mod n using two seperate assembly modules. \n");

printf("Module 2 can only compute the result if n is a power of 2. \n");

printf("A result of -1 for Module 2 indicates that the n used was not a power of two. \n\n");

printf("K | M | N | Module 1 Result | Module 2 Result \n");

for (i = 2; i < 26; i++){ //runs through k from 2 to 25

n = 4 \* i;

m = i;

vone = modone(n, m); //Module 1 result

vtwo = modtwo(n, m); // Module 2 result

printf("%d | %d | %d | %d | %d \n", i, m, n, vone, vtwo); // print all data for current set

}

getchar(); //type any character to terminate the program

return 0;

}

### Assembly Module 1 (modone.asm):

;input n, m to compute m mod n

.586

.MODEL FLAT

.CODE

; int modone(int n, int m) ;global procedure

; returns m-n\*floor(m/n)

\_modone PROC

push ebp ; save base pointer

mov ebp, esp ;establish stackframe

push ebx

mov ebx, [ebp+8] ;gets first parameter: n

mov eax, [ebp+12] ;gets second param: m

xor edx,edx ;clear EDX

mov ecx, ebx ;move ebx into ecx

div ecx ;divide m by n

mov eax,edx ;get remainder from edx and return it

pop ebx

pop ebp

ret ; return

\_modone ENDP

END

### Assembly Module 2 (modtwo.asm):

;Using AND logic take input n, m to compute m mod n

.586

.MODEL FLAT

.CODE

; int modtwo(int n, int m) ;global procedure

; returns m mod n using AND logic, but-

; if the n is not a power of two then -1 will be returned

\_modtwo PROC

push ebp ; save base pointer

mov ebp, esp ;establish stackframe

push ebx

mov eax, [ebp+8] ;gets first parameter: n

mov ebx, eax ;copying n into ebx

dec ebx ;(n-1)

mov ecx, -1 ;setting up return for if n is not a multiple of two

and eax, ebx ;n ^ (n-1)

jnz exitcode ;jump to exitcode if not equal to zero

mov eax, [ebp+8] ;gets first parameter: n

mov ebx, [ebp+12] ;gets second param: m

dec eax ;getting mask constant (n-1)

and eax, ebx ;mask constant ^ m = m mod n

mov ecx, eax ;move result into ecx

exitcode:

mov eax, ecx ;move result into eax or -1 into eax if n was not a multiple of two

pop ebx

pop ebp

ret ; return

\_modtwo ENDP

END

## Implementation:

The following table shows the result of the k values 2 to 25 being run with the program. A -1 in the column of results for module 2 indicates that the current n was not a power of two and m mod n could not be computed using this module since it uses AND logic.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Current K | M  (k) | N  (k\*4) | M mod N Result  From Module 1 | M mod N Result  From Module 2 |
| 2 | 2 | 8 | 2 | 2 |
| 3 | 3 | 12 | 3 | -1 |
| 4 | 4 | 16 | 4 | 4 |
| 5 | 5 | 20 | 5 | -1 |
| 6 | 6 | 24 | 6 | -1 |
| 7 | 7 | 28 | 7 | -1 |
| 8 | 8 | 32 | 8 | 8 |
| 9 | 9 | 36 | 9 | -1 |
| 10 | 10 | 40 | 10 | -1 |
| 11 | 11 | 44 | 11 | -1 |
| 12 | 12 | 48 | 12 | -1 |
| 13 | 13 | 52 | 13 | -1 |
| 14 | 14 | 56 | 14 | -1 |
| 15 | 15 | 60 | 15 | -1 |
| 16 | 16 | 64 | 16 | 16 |
| 17 | 17 | 68 | 17 | -1 |
| 18 | 18 | 72 | 18 | -1 |
| 19 | 19 | 76 | 19 | -1 |
| 20 | 20 | 80 | 20 | -1 |
| 21 | 21 | 84 | 21 | -1 |
| 22 | 22 | 88 | 22 | -1 |
| 23 | 23 | 92 | 23 | -1 |
| 24 | 24 | 96 | 24 | -1 |
| 25 | 25 | 100 | 25 | -1 |

## Comments on Results:

After completing this project and seeing the results it is evident that the doing m mod n with a computation that ends up being k mod k\*4 always will result in the same value that the k had. Additionally it was evident that the second module was only able to compute results when the n was a power of two and was able to compute a result for n = 8,16,32,64. Checking the results of my table with my calculator indicates that the results I got were correct and that my program was successful in its implementation and data.