CS 271 Assignment 1

Introduction to MASM assembly language

Due Date: 07/14/19

Michael Payne

TA Signature

1 Introduction

Assignment 1 is a short program that takes two integers from the user and then adds them together, subtracts the second integer from the first integer, multiplies the two integers, and then divides the the first integer by second integer. The results of each operation are printed for the user.

This assignment was intended to get us familiar with the basics of MASM assembly language, defining variables (integers and string), using the Irvine library procedures (to make input and output easier), and doing simple integer arithmetic. The assignment was, fortunately, simple and did an effective job of showing me how to do basic things with assembly. It also helped me improve at debugging and using the visual studio debugger to find out exactly what my program was doing (finding out what information each register was holding, checking addresses, etc.).

2 Program Initialization, Internal Register, Definitions, and Constants

Several strings are declared under .data (obviously each one has a byte per character). number_1, number_2, remainderResult, mathResult are all 4 bytes and contain uninitialized values. 'Intro_1', 'intro_2', 'question_1', 'question_2', 'addition', 'subtraction', 'division', 'multi', 'equals', 'remainder', and 'farewell' are 'strings' that have a byte (as stated before) for each character.

3 Main Program

The principal parts and functional logic of Assignment1 are represented in the diagram in Figure 1. The program starts with Print Intro Message. This calls intro_subroutine, which prints a greeting to the user and gives them instructions about entering two integers. Get Two Numbers From User prints question_1 and question_2, which each prompt the user to enter an integer. User input for the first integer is stored in number_1, and user input for the second integer is stored in number_2. In the Addition section, number_1 and number_2 are summed together, and the result is stored in mathResult. Number_1, addition, number_2, equals, and mathResult are all printed (in that order) to show the operation and its result. In the Subtraction section, number_2 is subtracted from number_1, and the result is stored in mathResult. Number_1, subtraction, number_2, equals, and mathResult are all printed (in that order) to show the operation and its result. In the Multiplication section, number_1 is multiplied by number_2, and the result is stored in mathResult. Number_1, multi, number_2, equals, and mathResult are all printed (in that order) to show the operation and its result. In the Division section, number_1 is divided by number_2, and the quotient is stored in mathResult. The remainder is stored in remainderResult. Number_1, division, number_2, equals, mathResult, remainder, and remainderResult are all printed (in that order) to show the operation and its result. The program then moves to Saying Goodbye where farewell is printed to say goodbye to the user. The program then ends.

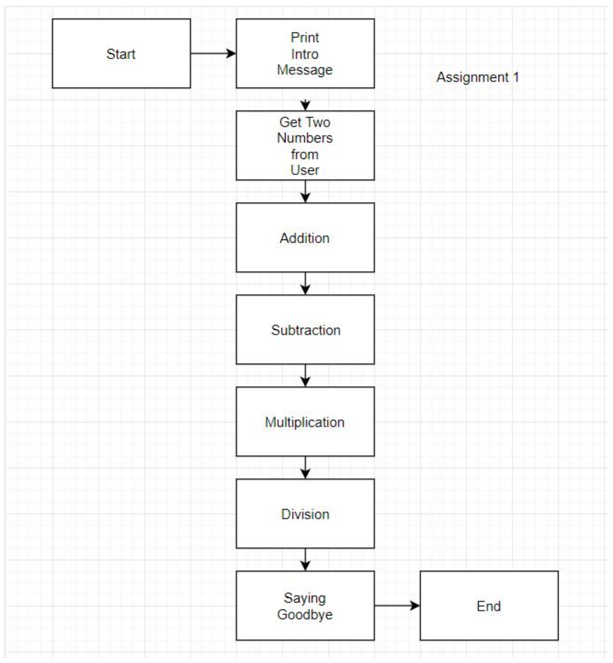

Figure 1: Block Diagram for Assignment1

4 A Subroutine

1. intro_subroutine

   ”intro_subroutine” is the first thing that runs when the program starts. It prints out the strings ’intro_1’ and ’intro_2, which displays ”Hey, my name is Michael Payne. Let’s do some simple arithmetic.” ”Enter 2 numbers, and I’ll show you the sum, difference, product, quotient, and remainder.” on separate lines.

5 Stored Program Data

1. ”Intro_subroutine” stage

   (a) intro_1 = ”Hey, my name is Michael Payne. Let’s do some simple arithmetic.”

   (b) intro_2 = ”Enter 2 numbers, and I’ll show you the sum, difference, product, quotient, and remainder.”

   (c) question_1 = ”Number 1:”

   (d) question_2 = ”Number 2:”

   (e) addition = ” + ”

   (f) subtraction = ” - ”

   (g) division = ” / ”

   (h) multi = ” * ”

   (i) equals = ” = ”

   (j) remainder = ” remainder ”

    (k) farewell = "Goodbye!"
    (l) number_1 = uninitialized value
    (m) number_2 = uninitialized value
    (n) mathResult = uninitialized value
    (o) remainderResult = uninitialized value

2. "Get two numbers from user" stage
    (a) number_1 = 10
    (b) number_2 = 5
3. "Addition" stage
    (a) mathResult = 15
4. "Subtraction" stage
    (a) mathResult = 5
5. "Multiplication" stage
    (a) mathResult = 50
6. "Division" stage
    (a) mathResult = 2
    (b) remainderResult = 0
7. "Saying Goodbye" stage
    (a) Nothing is changed/added

6 Difficulties

The largest issue was this was my first time programming in assembly, so although this assignment wasn't terribly difficult, I still had to familiarize myself with the language.

I suppose the biggest issue was using the Irvine library functions to take in user input and to print out strings and numbers (it's a multi-step process to do each of those things). Also, allocating the correct amount of data for each of my pieces of data was a challenge. Again, fortunately, this assignment was rather simple, so it made learning these things not terrifically difficult.

There was also the issue of learning to debug efficiently (while using MASM), which was somewhat painful at first, but again, the simplicity of the assignment made that less difficult than it could have potentially been.

7 Conclusion

I didn't have many issues with this assignment. It did take me a little bit of time to determine how much space to allocate for each piece of data and which registers to use when using Irvine library procedures (figuring out some basic latex stuff for this report was more difficult than the actual assembly for the assignment itself). I suppose the biggest issue was that I knew the code for this assignment could be easily cleaned up if I knew more assembly (functions that allowed me to pass in arguments would help cut down on redundant code tremendously).

Still, this assignment was very useful, because it allowed me to learn the very basics of assembly (like allocating the correct amount of memory for the various components of the program and using the Irvine library stuff to do things like take in user inputs and print stuff on the screen).

8 Extra Credit

This program has modules/segments that are titled: Print Intro Message (1 instruction), Get Two Numbers from User (8 instructions), Addition (14 instructions), Subtraction (15 instructions), Multiplication (16 instructions), Division (21 instructions), Saying Goodbye (3 instructions), Intro_Subroutine (7 instructions (I'm assuming you count the return command as an instruction)). The total number of instructions is 85.

9 Source Code

```
TITLE Assignment1    (Assignment1.asm)

; Author: Michael Payne
; Last Modified: 07/09/2019
; OSU email address: paynemi
; Course number/section: CS271
; Project Number: project1            Due Date:07/14/2019
; Description: This program will ask for two integers from the user.  It will then take the
; subtract (subtract the second integer from the first one), multiply, and divide (divide s
; It then prints the results of each operation and says goodbye to the user.

INCLUDE Irvine32.inc

; (insert constant definitions here)

.data

intro_1 BYTE "Hey, my name is Michael Payne.  Let's do some simple arithmetic.", 0
;
intro_2 BYTE "Enter 2 numbers, and I'll show you the sum, difference, product, quotient, ar
question_1 BYTE "Number 1:",0
question_2 BYTE "Number 2:",0
addition BYTE " + ",0
```

```
subtraction BYTE " - ",0
division BYTE " / ",0
multi BYTE " * ",0
equals BYTE " = ",0
remainder BYTE " remainder ",0
farewell BYTE "Goodbye!",0

number_1 DWORD ? ; first number that user inputs
number_2 DWORD ? ; second number that user inputs
mathResult DWORD ? ; result of whatever operation is done to two integers
remainderResult DWORD ? ; remainder when first integer is divided by second integer


.code

main PROC

; (insert executable instructions here)
; (print intro message)

call intro_subroutine ; I used an intro subroutine like the one in your example program

; (get two numbers from user)

mov edx, offset question_1 ; offset of question_1 to edx
call WriteString ; print question_1
call ReadInt ; read integer in eax
mov number_1, eax ; move user input to number_1
mov edx, offset question_2 ; offset of question_2 to edx
call WriteString ; print question_2
call ReadInt ; read integer in eax
mov number_2, eax ; move user input to number_2

; (addition)

mov eax, number_1 ; number_1 to eax so addition can be done
add eax, number_2 ; add number_2 to eax (which has value of number_1)
mov mathResult, eax ; eax to mathResult
mov eax, number_1 ; number_1 to eax
call WriteDec ; print number_1
mov edx, offset addition ; offset of addition to edx
call WriteString ; print addition
mov eax, number_2 ; number_2 to eax
call WriteDec ; print number_2
mov edx, offset equals ; offset of equals to edx
call WriteString ; print equals
```

```
mov eax, mathResult ; mathResult to eax
CALL WriteDec ; print mathResult
call CrLf ; add line or break point

;(subtraction)

mov eax, number_1 ; number_1 to eax
sub eax, number_2 ; subtract number_2 from number_1
mov mathResult, eax ; result of subtraction to mathResult
mov eax, number_1 ; all of this is identical to how results of addition were printed
call WriteDec
mov edx, offset subtraction
call WriteString
mov eax, number_2
call WriteDec
mov edx, offset equals
call WriteString
mov eax, mathResult
CALL WriteDec
call CrLf

;(multiplication)

mov eax, number_1 ; number_1 to eax
mov ebx, number_2 ; number_2 to ebx
xor edx,edx ; clearing edx since overflow of multiplication goes to edx
mul ebx ; multiplying number_1 by number_2
mov mathResult, eax ; result of multiplication to mathResult
mov eax, number_1 ; All of this is identical to how addition and subtraction were printed
call WriteDec
mov edx, offset multi
call WriteString
mov eax, number_2
call WriteDec
mov edx, offset equals
call WriteString
mov eax, mathResult
CALL WriteDec
call CrLf

;(division)

mov eax, number_1 ; number_1 to eax
mov ebx, number_2 ; number_2 to ebx
xor edx,edx ; clearing edx since remainder of division is stored there
div ebx ; dividing number_1 by number_2
```

```
mov mathResult, eax ; result of division to mathResult
mov remainderResult, edx ; remainder of division to remainderResult
mov eax, number_1 ; I will mention differences of printing division where appropriate
call WriteDec
mov edx, offset division
call WriteString
mov eax, number_2
call WriteDec
mov edx, offset equals
call WriteString
mov eax, mathResult
CALL WriteDec
mov edx, offset remainder ; offset of remainder to edx
call WriteString ; printing remainder (the word)
mov eax, remainderResult ; remainderResult to eax
call WriteDec ; printing remainderResult (the actual remainder generated from division)
call CrLf

;(saying goodbye)
mov edx, offset farewell ; offset of farewell to edx
call WriteString ; printing farewell
call CrLf ; break point space whatever

exit ; exit to operating system

main ENDP

; (insert additional procedures here)

; This subroutine prints a welcome message for the user
intro_subroutine        PROC
mov edx, OFFSET intro_1 ; offset of intro_1 to edx
call WriteString ; print intro_1
call CrLf ; add new line
mov edx, OFFSET intro_2 ; offset of intro_2 to edx
call WriteString ; print intro_2
call CrLf ; add new line

ret ; return to main
intro_subroutine ENDP

END main
```