Regis University CC&IS
CS370 Assembly Language

Programming Assignment 1

Due by: midnight Sunday of Week 1

Introduction

The programs for this week assume that you have already installed Visual C++ (VC++), and the author's libraries and example programs. They also assume that you have reviewed how to set and unset breakpoints, and start the debugger.
This week's programs will consist of:

- Setting up a Visual C++ project template.
- Executing a modified program.
- Documenting the results of executing the modified program.

Before you start the assignment's programs, you will want to perform the steps in the **Pre-Task**. The **Pre-Task** will help familiarize you with executing programs in Visual C++.

Pre-Task

To prepare for this assignment and subsequent assignments, you will want to create a VC++ project template that you can use as the beginning structure for all. To accomplish this, do the following:

1. On the Author's website, click the "Creating a Project from Scratch" link. Follow the instructions found there to create a project template in a directory of your choice.
2. Download the **AddSub.asm** file, from the online content.
3. If the Solution Explorer window is not visible, bring it up through the View menu.
4. In the Solution Explorer window, under Project, double-click **main.asm** to open it for editing.
5. Delete the code in the **main.asm** file.
6. Copy the contents of the **AddSub.asm** and paste the contents in the **main.asm** file.
7. Build the project and run it (Select "Debug", "Start without debugging").

   This should result in something like this:

```
EAX=00030000  EBX=7FFDE000  ECX=0012FFB0  EDX=7C90EB94
ESI=01C748D4  EDI=3F7517F0  EBP=0012FFF0  ESP=0012FFC4
EIP=00401024  EFL=00000206  CF=0  SF=0  ZF=0  OF=0  AF=0  PF=1
Press any key to continue . . .
```

Some concepts about the AddSub program that you should review are:

- The AddSub program demonstrates the use of registers to perform add and subtract operations.

- The mDumpMem command is one of the author's macros. Its purpose is to make it easy to display a range of memory locations in a flexible format. I highly encourage you to review this command in the textbook. Only register EAX is used by the program so your EAX register

should match the example above. The other registers will not necessarily match.

- All of the data being manipulated is 32-bit, two's complement integer data.
- Starting with the value 10000h in EAX, the program first adds 40000h then subtracts 20000h. The result is 30000h.

(see next pages for Program 1 and 2 requirements)

Program #1

Modification

- Modify the **AddSub** program from the Pre-Task as follows:

```
TITLE Move and add          (AddSub.asm)

; This program adds and subtracts 32-bit integers.

INCLUDE Irvine32.inc
INCLUDE macros.inc      ; for memory dump

.data
x       DWORD 0FFF4A348h  ; summands
y       DWORD 00076540h
z       DWORD ? ; sum

.code
main PROC

        mov     ebx,x
        mov     eax,ebx
        mov     ecx,y
        add     eax,ecx
        mov     z,eax
        call    DumpRegs
        mDumpMem offset x,12,1 ; Display data area

        exit
main ENDP
END main
```

Execution

- Build and execute the program.

Documentation

- For each value in the EAX, EBX and ECX registers:
    - Record the name of the register (e.g., EAX)
    - Record the 2s complement value in the register (the value seen in the memory dump)
    - Convert the 2s complement value to its Base 10 equivalent. You MUST show all of your conversion work.
- For each value in the x, y and z variables:
    - Record the name of the variable (e.g., x)
    - Describe how and where the variable's value is represented in the memory dump.
- For each command in the main procedure, describe what the command on that line is doing.
- Finally, describe what mathematical task the program performed and what the result of that task was.

Program #2

Modification

- Create a new copy of your project template in a directory that is associated with your Week 1 program.
- Download the **Moves.asm** program from the online content.
- Copy and paste the **Moves.asm** content to your **main.asm** file.
- Modify the **Moves** program as follows:

Change this…	From this…	To this…
val1	1000h	1234h
val2	2000h	5678h
arrayB	10h, 20h, 30h, 40h, 50h	12h, 34h, 56h, 78h, 9Ah
First line after MOVZX	mov bx,0A69Bh	mov bx,085C2h
First line after MOVSX	mov bx,0A69Bh	mov bx,085C2h

Execution

- Build the program.
- Start the debugger (e.g., press F10)

Documentation

- Add a section header to your document that says "**MOVZX Section**". For this section:
  - Step over each line in the program until you have executed the command directly after the MOVZX comment (i.e., execute the command mov bx 085c2h).
  - Record the value in the EBX register along with its Base 10 equivalent.
  - Execute the next three commands.
  - Record the values in the EAX, EDX and CX registers along with their respective Base 10 equivalents.
  - For each of the four commands that you just executed, describe what the command on that line is doing.

- Add a section header to your document that says "**MOVSX Section**". For this section:
  - Execute the five commands under the MOVSX label.
  - Record the value in the BX, BH and BL registers along with their respective Base 10 equivalents.
  - For each of the four commands that you just executed, describe what the command on that line is doing.

- Add a section header to your document that says "**Memory-to-Memory Section**". For this section:
  - Display the **Memory 1** window of VC++ and type **&val1** (the address of your first variable in the data area) into the Address textbox of the **Memory 1** window.
  - Right-click inside the **Memory 1** Window. In the resulting pop-up menu, select "2-byte Integer" instead of the default (1-byte Integer) as the data display mode. Note that the display mode we have chosen doesn't show memory content in the way it really exists (i.e. it doesn't show the little-endian representation of multi-byte data in memory).

    - Execute the three commands under the Memory-to-memory exchange label one at a time. For each line:
        - Record the line number, the value in the val1 variable, the value in the val2 variable and the value in the AX register.
        - Describe what the command on that line is doing.

- Add a section header to your document that says "**Direct-offset Addressing (Bytes)**". For this section:
    - Change the memory display mode back to "1-byte Integer".
    - Type **&arrayB** in the Address text box of the Memory 1 window to realign the display to start at the address of **arrayB.**
    - Execute the three commands under the Direct-Offset Addressing (byte array) label one at a time. For each line:
    - Record the step number and the value in the AL register and describe what the command on that line is doing.

- Add a section header to your document that says "**Direct-offset Addressing (Words)**". For this section:
    - Type **&arrayW** in the Address text box of the Memory 1 window to realign the display to start at the address of **arrayW.**
    - Change the memory display mode to "2-byte Integer".
    - Record the three word values in the three arrayW indexes along with their respective Base 10 equivalents.

Program Submission

Submit each of program documents to the **Programming Assn 1** Dropbox.

Before submitting your program documents, you MUST name them as follows:

Lastname-Assn1-Program1.xxx
Lastname-Assn1-Program2.xxx

Where **xxx** is one of the following document types .doc, .docx, or .txt. As noted before, a Word document is preferred.

For example:

Jones-Assn1-Program1.docx
Jones-Assn1-Program1.docx

Grading

The following rubric will be used to grade your program.

Rating Category	Exemplary	Partially Proficient	Basic (needs work)	Not Demonstrated
Documentation	Documentation is well written, clearly explains what the code is accomplishing. Includes complete and accurate recorded values.	Documentation is well written, clearly explains what the code is accomplishing. Includes a few incorrect recorded values.	Documentation explains what the code is accomplishing, but is entirely clear. Includes missing or completely incorrect recorded values.	Documentation is incomplete or not attempted.
Delivery	Submitted on time	Submitted 1-3 days late (3% deducted per day late)	Submitted 4-7 days late (3% deducted per day late)	Not submitted within 1 week of due date