# 16.317: Microprocessor Systems Design I

Fall 2014

Homework 2

Due **Wednesday, 9/24/14**

**Notes:**

- While typed solutions are preferred, handwritten solutions are acceptable (except for Question 3b).
- All solutions must be legible and contained in one file. Archive files are not acceptable.
- Electronic submissions should be e-mailed to Dr. Geiger at Michael_Geiger@uml.edu.
- This assignment is worth 100 points.

1. (45 points) Assume the state of the x86 registers and memory are:

EAX: 00000010H
EBX: 00000020H
ECX: 00000030H
EDX: 00000040H
CF: 1
ESI: 00020100H
EDI: 00020100H

| **Address** | | | | |
|---|---|---|---|---|
| 20100H | 10 | 00 | 08 | 00 |
| 20104H | 10 | 10 | FF | FF |
| 20108H | 08 | 00 | 19 | 91 |
| 2010CH | 20 | 40 | 60 | 80 |
| 20110H | 02 | 00 | AB | 0F |
| 20114H | 30 | 00 | 11 | 55 |
| 20118H | 40 | 00 | 7C | EE |
| 2011CH | FF | 00 | 42 | D2 |
| 20120H | 30 | 00 | 30 | 90 |

What is the result produced in the destination operand by each of the instructions listed below? Assume that the instructions execute in sequence.

ADD AX, 00FFh
ADC CX, AX
INC BYTE PTR [20100h]
SUB DL, BL
SBB DL, [20114h]
DEC BYTE PTR [EDI+EBX]
NEG BYTE PTR [EDI+0018h]
MUL DX
IMUL BYTE PTR [ESI+0006h]
DIV BYTE PTR [ESI+0008h]
IDIV BYTE PTR[ESI+0010h]

2. (25 points) Assume the state of the 80386DX's registers and memory are:

EAX: 00005555H
EBX: 00045010H
ECX: 00000010H
EDX: 0000AAAAH
ESI: 000000F2H
EDI: 00000200H

| Address | | | | |
|---|---|---|---|---|
| 45100H | 0F | F0 | 00 | FF |
| | … | | | |
| 45200H | 30 | 00 | 19 | 91 |
| | … | | | |
| 45210H | AA | AA | AB | 0F |
| | … | | | |
| 45220H | 55 | 55 | 7C | EE |
| | … | | | |
| 45300H | AA | 55 | 30 | 90 |

What is the result produced in the destination operand by each of the instructions listed below? Assume that the instructions execute in sequence.

AND BYTE PTR [45300H], 0FH
SAR DX, 8
OR [EBX+EDI], AX
SHL AX, 2
XOR AX, [ESI+EBX]
NOT BYTE PTR [45300H]
SHR AX, 4

3. (30 points) This exercise will give you some familiarity with debugging assembly programs in Visual Studio. You will also learn how to view the contents of the registers and memory. **The next two pages describe how to run assembly programs in the debugger while viewing registers and memory, while the actual exercises begin on page 4 of this assignment.**

Download the program `hw2_p2.c` from the course web page. Create a new Visual Studio project, and add this file to your project, either by adding it as an existing item, or by creating a new C file inside your project and copying the contents of `hw2_p2.c` into your new file.

To complete this exercise, you will need to set breakpoints in the program—points at which the debugger will stop and allow you to view the program state. Breakpoints in Visual Studio can be set either by clicking in the gray margin to the left of a given line, or moving the cursor to the desired line, then choosing "Toggle Breakpoint" from the DEBUG menu (or pressing F9).

For parts (a)-(d), you need breakpoints on the 1st, 5th, and 7th instructions; a red circle will appear next to each line with a breakpoint. Your Visual Studio window should look like Figure 1 below.

To display the contents of the registers and memory, you must start the debugger by either clicking the "Local Windows Debugger" button (circled in Figure 1) or choosing "Start Debugging" from the DEBUG menu (or pressing F5). The debugger will automatically run the program until it reaches the first breakpoint.

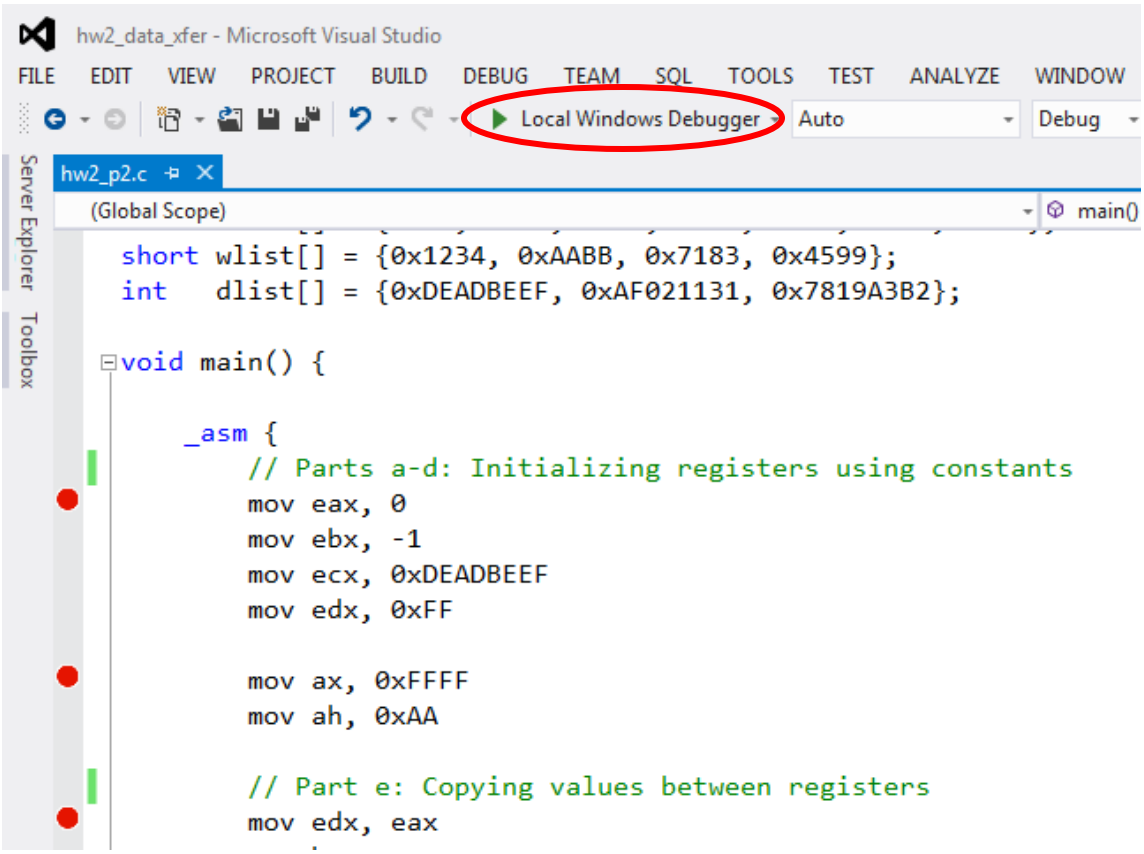


**Figure 1:** The source file `hw2_p2.c` with the first three breakpoints set. The area in which red circles appear is the "margin" in which you can click to set a breakpoint on a line. The "Local Windows Debugger" button is circled.

After starting the debugger, display the registers by opening the DEBUG menu, then choosing "Windows → Registers." You can also display the contents of memory using the same menu—choose "Windows → Memory" and then choose any of the four choices that appear. Both choices are shown in Figure 2.

Once both registers and memory are displayed, you can toggle between them by simply selecting "Registers" or "Memory 1" below the area that shows the contents of each. To display the contents of any variable or array stored in memory, as well as the data that directly follow that variable or array, type its name in the "Address" field. Both features are shown in Figure 3.

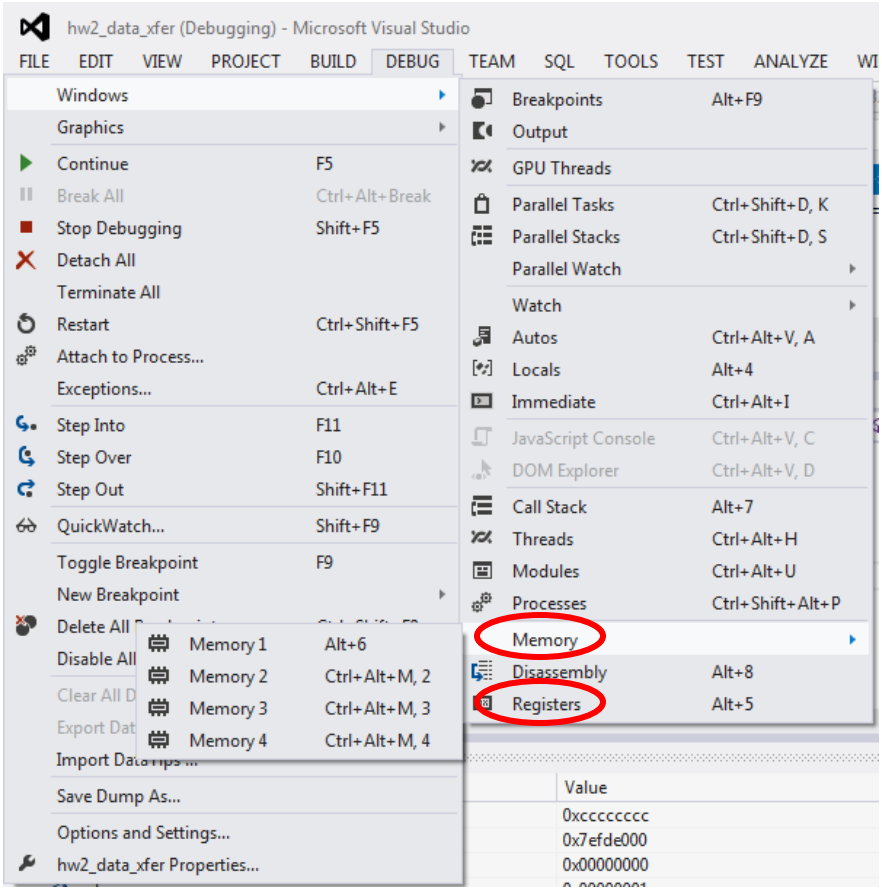


**Figure 2:** The entries in the DEBUG menu used to display both registers and memory.

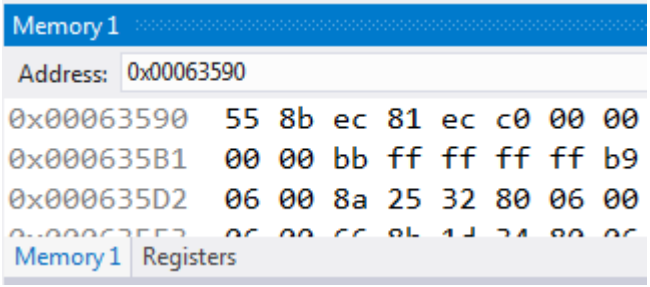


**Figure 3:** Part of the area used to show the contents of registers and memory. The tabs at the bottom of the area allow you to toggle between both views. When viewing memory, typing the name of a variable or array in the "Address:" field allows you to see its contents.

a. (3 points) Generate and submit a screenshot of the register contents at the start of the program.

b. (4 points) Type the array name "`blist`" into the address field to see the contents of memory starting at the beginning of that array. Generate and submit a screenshot of those memory contents. Describe the data shown in that screenshot and how it corresponds to any of the arrays declared at the start of the program (`blist`, `wlist`, and `dlist`).

c. (4 points) To execute the first four instructions, press the "Continue" button (which has replaced the "Local Windows Debugger" button) or use the "Continue" option from the DEBUG window. Look at the Registers window and describe how the register contents have changed after running those first four instructions.

d. (3 points) Use the "Continue" command to execute the next two instructions and run to the third breakpoint. Explain the changes these two instructions generate, and submit another screenshot of the register state at this point.

e. (4 points) Set a breakpoint on the instruction `mov al, [blist]`. Use the "Continue" command to execute the next three instructions, describe their operation, and generate a screenshot showing the register state after these instructions complete.

f. (4 points) Set a breakpoint on the instruction `mov bx, [wlist]`. Use the "Continue" command to execute the next four instructions, describe their operation, and generate a screenshot showing the register state after these instructions complete. Be sure to discuss the differences between the `movsx` and `movzx` instructions used in this step.

g. (4 points) Set a breakpoint on the instruction `mov esi, 0`. Use the "Continue" command to execute the next three instructions. How are the memory accesses performed in these instructions different than those performed in part (f)? Generate a screenshot showing the memory state that has been changed within this sequence of instructions.

h. (4 points) The last group of instructions in this program comprises a loop, which will repeatedly execute the same instructions until the end condition is reached. To see the loop operation, set one breakpoint on the instruction `mov eax, [dlist + 4 * esi]`, and another one on the curly brace `}` at the end of the program.

   Using "Continue" once will bring you to the start of the loop. Each time you use "Continue" after that will execute one iteration of the loop. When the program reaches the final breakpoint, the loop will be done.

   How many iterations does the loop execute? What happens in each loop iteration? How is the memory address accessed in each loop iteration changed?