16.317: Microprocessor Systems Design I

Spring 2014

Homework 2

Due **Friday, 2/14/14**

Notes:

- While typed solutions are preferred, handwritten solutions to Problem 1, as well as the questions in Problem 2, are acceptable. However, Problem 2 requires you to generate a number of screenshots demonstrating your ability to execute the program provided in Visual Studio.
- Electronic submissions should be e-mailed to Dr. Geiger at Michael_Geiger@uml.edu.
- All electronic submissions must be contained in a single file (Word document or PDF). Archived submissions (e.g., zipped folders sent as a .zip file) are not acceptable.
- **All submissions must be received by 12:00 PM (noon) on Sunday, 2/16. I will post the solution that day; no homework will be accepted after that point.**
- This assignment is worth 50 points.

1. (20 points) Assume the state of the x86 registers and memory are as shown below. Note that all values shown in memory are in hexadecimal:

Initial state:

EAX: 00000000h
EBX: 00000008h
ECX: 0000021Eh
EDX: 0000FFFEh
ESI: 0000F000h
EDI: 00000101h
DS: 2201h
ES: 2000h

Address	Lo			Hi
22000h	20	13	80	40
22004h	FF	AF	BC	13
22008h	99	88	77	66
2200Ch	A8	B1	F0	43
22010h	78	D6	32	33
22014h	34	35	12	16
22018h	93	03	7C	EF

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

```
MOV     AX, [BX+01H]
MOV     [000Ah], CX
MOVSX   EBX, BYTE PTR [0001H]
MOVZX   DWORD PTR ES:[SI+3000h], DX
LEA     DI, [SI+1A2BH]
LDS     EDX, ES:[2006H]
```

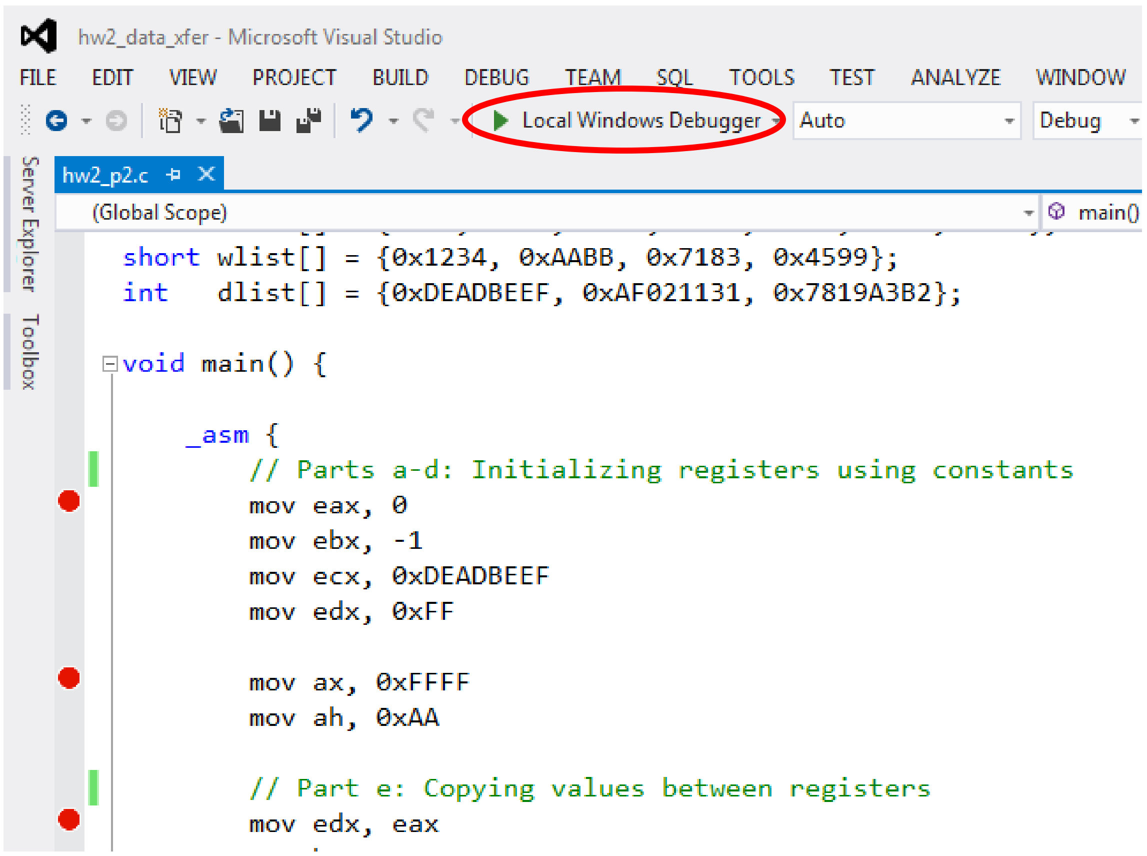
2. (30 points) This exercise will give you some familiarity with debugging assembly programs in Visual Studio. You will also learn how to view the contents of the registers and memory. **The next two pages describe how to run assembly programs in the debugger while viewing registers and memory, while 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.


hw2_data_xfer - Microsoft Visual Studio
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM SQL TOOLS TEST ANALYZE WINDOW
Local Windows Debugger
Auto
Debug
hw2_p2.c
(Global Scope)
main()
Server Explorer
Toolbox

```
short wlist[] = {0x1234, 0xAABB, 0x7183, 0x4599};
int   dlist[] = {0xDEADBEEF, 0xAF021131, 0x7819A3B2};

void main() {

    _asm {
        // Parts a-d: Initializing registers using constants
        mov eax, 0
        mov ebx, -1
        mov ecx, 0xDEADBEEF
        mov edx, 0xFF

        mov ax, 0xFFFF
        mov ah, 0xAA

        // Part e: Copying values between registers
        mov edx, eax
```

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.

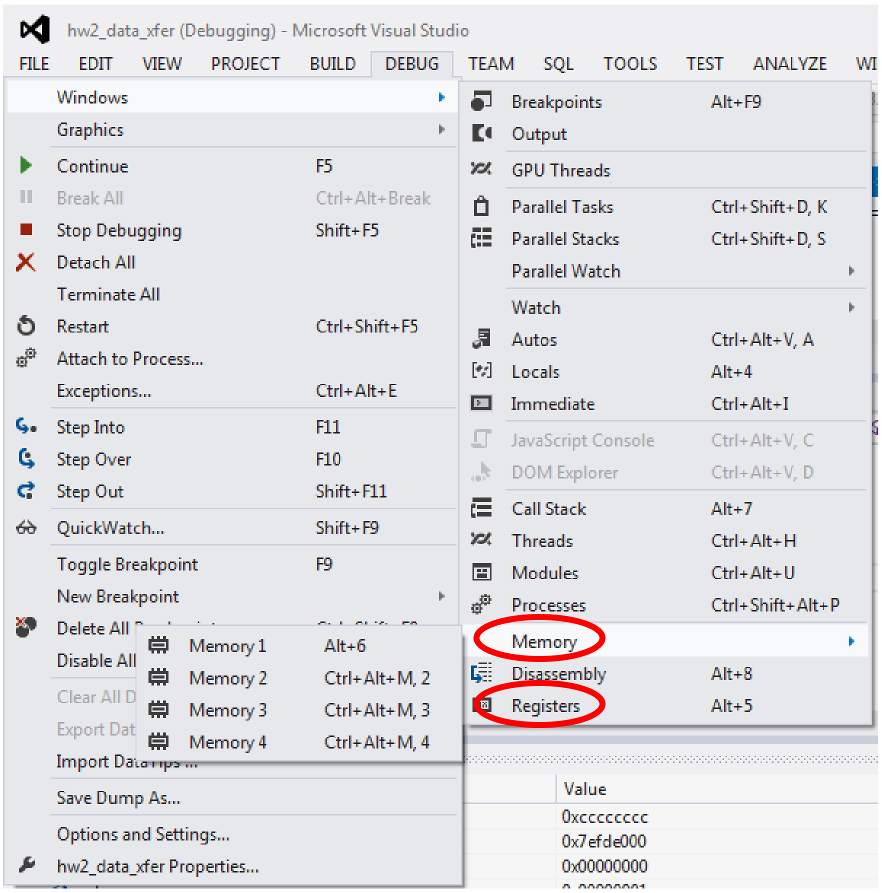
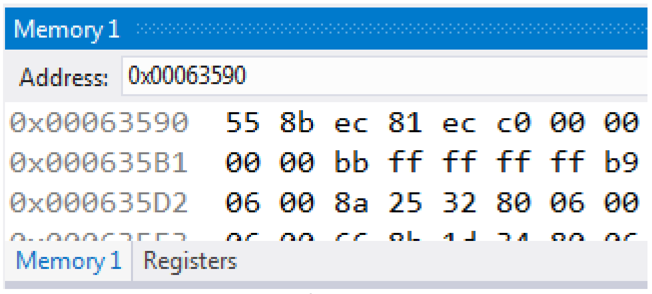


Figure 2: The entries in the DEBUG menu used to display both registers and memory.



```
Address: 0x00063590
0x00063590  55 8b ec 81 ec c0 00 00
0x000635B1  00 00 bb ff ff ff ff b9
0x000635D2  06 00 8a 25 32 80 06 00
```

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?