# Lab Practice Week 9 and Assembly language Program

Lab Practice to be submitted as Assembler Exercise by Week 12  
Assembly Language Program to be submitted by Week 13

**Internal Students:** You need to show a working version of your solutions to program 20, 21, 22 and 23. **Your tutor will expect to see your submission on or before your lab class in week 12 as part of your Assembly Language Exercise Assessment.**

**External Students:** Please email your Program 20, 21, 22 and 23 to your tutor. Your tutor will expect to receive them by the end of week 12.

## Program 20

Determine the hexadecimal value of the destination operand after execution of each of the statements in the following table. If any instruction is illegal, write the word ILLEGAL as the answer and give the reason. var1 and var2 are word variables (16-bit), and count is a byte variable (8 bit).

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Before Execution</th>
<th>After Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>a) mov ax, bx</td>
<td>AX = 0023, BX = 00A5</td>
<td>AX =</td>
</tr>
<tr>
<td>b) mov ah, 3</td>
<td>AX = 06AF</td>
<td>AX =</td>
</tr>
<tr>
<td>c) mov dl, count</td>
<td>DX = 8F23, count = 1A</td>
<td>DL =</td>
</tr>
<tr>
<td>d) mov bl, ax</td>
<td>BX = 00A5, AX = 4000</td>
<td>BL =</td>
</tr>
<tr>
<td>e) mov var1, bx</td>
<td>var1 = 0025, BX = A000</td>
<td>var1 =</td>
</tr>
<tr>
<td>f) mov count, ax</td>
<td>count = 25, AX = 4000</td>
<td>count =</td>
</tr>
<tr>
<td>g) inc bl</td>
<td>BX = FFFF</td>
<td>BX =</td>
</tr>
<tr>
<td>h) dec cx</td>
<td>Cx = 0000</td>
<td>CX =</td>
</tr>
<tr>
<td>i) add var1, bx</td>
<td>var1 = 0025, BX = A000</td>
<td>var1 =</td>
</tr>
<tr>
<td>j) sub var1, var2</td>
<td>var1 = 15A6, var2 = 01B8</td>
<td>var1 =</td>
</tr>
</tbody>
</table>

## Program 21

As each of the instructions given in the code segment below is executed, fill in the hexadecimal value of the operand listed on the right side (in the comments section). Also specify the addressing mode.

```assembly
; ----- data segment ----- .data
array1    dw  20h, 10h
array2    dw  30h, 40h

; ----- code segment ----- .code
mov ax, @DATA ;copy the addr. of data
mov ds, ax
mov ax, array1 ;AX =
xchg array2, ax ;AX = array2 =
dec ax ;AX =
sub array2, 2 ;array2 =
mov bx, array2 ;BX =
add ah, bl ;AX =
```


Program 22

Use **Debug** to assemble and trace the following program. Write down the contents of the Zero, Carry, and Sign flags after tracing each instruction. Write a short note next to each line explaining why each of the flags changed:

```assembly
mov al, FF
inc al
sub al, 2
mov dl, al
add dx, 2
int 20
```

Now do the same for the following instructions:

```assembly
mov ax, 7FFF
inc ax
mov bx, 8000
dec bx
int 20
```

Provide an explanation of how and why the Carry and Overflow flags are affected.

Program 23

Translate the following C statements to assembly instructions.

1. `int variable1 = 2;`
2. `int variable2 = 3;`
3. `int sum;`
4. `variable1 = 2 + 7;`
5. `sum = variable2 + variable1;`

Additional Exercises on the use of Debug

Supplementary Exercise 1.

Assemble the following instructions using the MS debugger **Debug**. Type `?` for help on the available commands and use these commands to trace the program execution one instruction at a time.

Observe the values of various registers as you trace through the executions of the instructions. Copy the trace of instructions to a file for printing. On the printout, circle the registers that have changed after each instruction. What are the values in AX and BX registers after the execution of last instruction?
-a 100
  mov ax, 1020
  mov bl, ah
  mov bh, al
  add bl, 2
  add bh, 1
  add bx, 12ab
  add ax, bx

(Note that when you press Enter at the end of each line, Debug prompts for the next line of input. Each input line starts with a segment-offset address.)

To terminate input, press the Enter key on a blank line, eg,

C:\WINDOWS> debug -a 100
-1B7F:0100 mov ax, 1020
1B7F:0103 mov bl, ah
1B7F:0105 mov bh, al
1B7F:0107 add bl, 2
1B7F:010A add bh, 1
1B7F:010D add bx, 12ab
1B7F:0111 add ax, bx
1B7F:0113 MOV AH, 4C ; to terminate the program and exit to DOS
1B7F:0115 INT 21
1B7F:0117

(bold text is typed by the user)

To display the contents of registers, use the R command (i.e., type R and then press Enter).
To trace the program, use the T command (eg, T=100). Each T command executes one instruction and displays the contents of all registers.
To quit debug, use the Q command.
You should try other Debug commands as well, eg, U, D and E.

Note that you can save your Debug session using the Mark and Copy operations in the MS-DOS Prompt window. The MS-DOS window toolbar has commands that you can use to mark a section of the window, copy it to the clipboard, and paste it into some other application (such as Notepad or Word).

Supplementary Exercise 2.

Assemble and trace the following program using Debug. List the registers that have changed after each instruction. Observe the values in the AX, BX and CX registers after the execution of each instruction.

- a 100
  mov ax, 4CFF
  mov [200], ax
Assembly Language Program (to be submit in Week 13) 10%

Write a simple calculator program. The program gets two integers and the operation to be performed on those two numbers from the user. The numbers and operators are on successive lines of input from the user. There are five operations on this calculator:

- Addition
- Subtraction
- Multiplication
- Division
- Remainder

Each of these operations is to be performed by calling a separate assembler function which is passed the two numbers to be used as operands. Those functions return the result of their operation to the calling mainline.

The Program can be developed as a stand-alone Assembly Language program or as a mixed-language program combining C and Assembly Code. Note that if the program is written in Assembly language, the program has to handle data conversion between ASCII code and its numeric values. If the program is written as Mixed-language development, the mainline could be written in C to handle the I/O for the calculator and then determines which assembler language functions to be use.

The submitted program should contain the design, algorithm, program listing and examples illustrating operation of the program or modules.

Note: While you may develop your program in any platform or compiler, all programs are expected to be able to run in the Windows environment. Even you may not be able to produce the complete program in the lab session, you MUST record and show your work to your tutor, otherwise no marks will be given. You should be able to give the following information:

1. An overall design of the program (How does the program work?)
2. Algorithm of the program (How do you process the data?)
3. Code and comments (What have you developed?)
4. Results (What are the outputs from the program?)
5. Testing (How did you test it?)
6. Discussion (Does it work? If no, what will you do next? If yes, how can it be improved?)