CS221 Assembly Language

Lab 01: Introduction to Assembly Language

Objectives

- Learn the basic template of a MASM assembly code.
- Use the IA-32 general-purpose registers.
- Use and understand `mov`.
- Understand how to use binary, octal, decimal, and hexadecimal values in MASM.
- Use `add`, `sub`, `inc`, and `dec`.
- Use the debugger to inspect the registers during execution.

Practice

The basic 32-bit MASM template is

```
.686
.model flat, stdcall
.stack 4096
ExitProcess proto,dwExitCode:dword
.data
	; declare variables here
.code
main proc
	; write your code here
	invoke ExitProcess, 0
main endp
	; write additional procedures here
end main
```

Write the following assembly instructions inside the main procedure `main proc`. Then, use the debugger to check the content of each register after executing each line. Note that all the destination operands for the instructions in this example are registers. Figure 1 lists the IA-32 registers.

```
	mov eax, 0
	mov eax, 43707
	mov ebx, 255

	; swapping values
	mov ecx, eax
	mov eax, ebx
	mov ebx, ecx

	; immediate hexadecimal values use the h suffix.
```

```
mov edx, 0FFh
mov ebp, 0AABBh          ; Delete the leading zero. What happens?

; immediate binary values have the suffix b or y.
mov esi, 11111111b
mov edi, 1010101010111011b

; immediate octal values have the suffix o or q.
mov eax, 377o
mov ebx, 125273o

; character constants (ASCII values).
mov eax, "5"
mov ebx, 5     ; observe the different values in eax and ebx.
mov eax, "ZA"  ; 32-bit register can hold 4 ASCII characters, try it yourself.
```

IA-32 instructions have the following format:

```
[label:]     mnemonic     [operands]     [;comment]
```

Remember, when the mnemonic requires two operands, the first is the destination and the second is the source. For example:

```
mov eax, ebx
```

In the previous instruction, the source is `ebx` register and the destination is the `eax` register.

Exercises

Use the mnemonics in Table 1 to answer the questions in this section.

Table 1

Mnemonic	# of Operands	Function
`mov`	2	Stores the value of the source in the destination.
`add`	2	Adds the value of the source to the destination. The value is stored in the destination.
`sub`	2	Subtracts the value of the source from the destination. The value is stored in the destination.
`inc`	1	Increment the operand by 1.
`dec`	1	Decrement the operand by 1.

1. Initialize all the general-purpose registers with random values.
2. Add the values of all the general-purpose registers shown in Figure 1 into the `eax` register.
3. Find the minimum and maximum values that can be stored in 32-bit registers. Examine the values using the debugger.

4. Use the general-purpose registers shown in Figure 1 to calculate the following expression:

$$X = (A - B) + (C + D - E) - 19$$
$$Where:$$
$$A = 100, B = 55, C = 15, D = 200, and\ E = 123$$

5. Printing the result: you can use the procedure `WriteDec` provided by the Irvine32 library to print decimal (unsigned) values. The `WriteDec` procedure requires the value to be printed in `eax`. Move the result of task 3 into `eax`. Then, call the procedure using the following x86 instruction:

```
call WriteDec
```

   Did you get an error after assembling the instructions? What is missing? Try adding "`include irvine32.inc`" before "`.686`" and try again. Why is there a warning after adding `include irvine32.inc`?
6. Call the procedure "`DumpRegs`". What does it do?

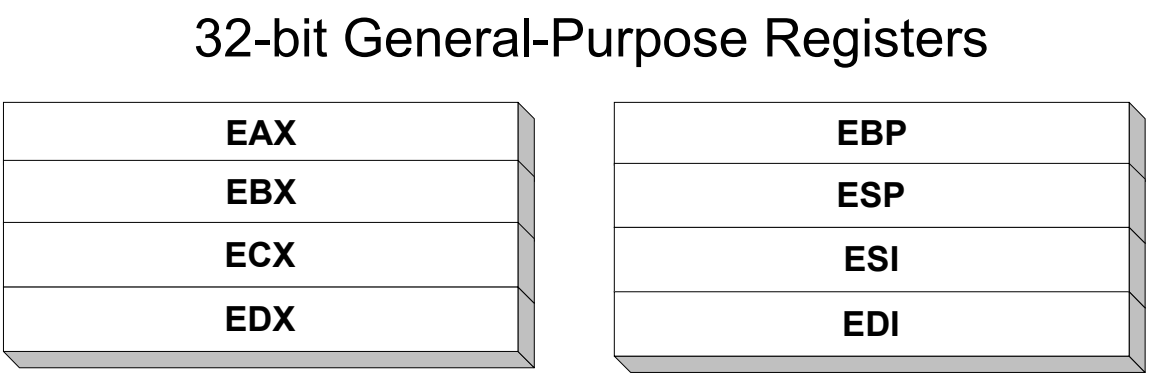

Figure 1