Further assembly language

Two topics: (1) Why study assembly language?
(2) Self-modifying code

(1) Why study assembly language?

Answer: There are many reasons e.g. understand the link between high level languages and machine code.
Another reason: write correct programs. See following example:

[Demo: Counter.java (available on resources page)]

Why does the value of X in this code not return to zero after being incremented and decremented the same number of times?

It is due to interleaving of instructions.

thread 1

```
for i = 1 to 10000
    X = X + 1
```

```
load X      (A)
add one     (B)
store X     (C)
```

thread 2

```
for i = 1 to 10000
    X = X - 1
```

```
load X      (D)
subt one    (E)
store X     (F)
```

If X is initialized to zero, then we execute

A, B, C, D, E, F

X ends at zero

BUT if we execute

A B D E F C] this is a legal interleaving of the 2 threads

the effect is:

	X	AC
A	0	0
B	0	1
D	0	1
E	0	-1
F	-1	-1
C	-1	-1

- so final value of X is -1, not 0 !!

There is an easy way to fix this problem in Java. But we need to understand instruction sets and assembly language to understand the problem in the first place.

② Self-modifying code

Puzzle: what does the following code do?

```
                load J
                add Value
                store J
J,              jump 0      / this instruction gets modified before
                            / being executed
                halt
                load Three
                output
                halt
                load Nine
                output
                halt
Value,          dec 5
Three,          dec 3
Nine,           dec 9
```

example of self-modifying code.

Instruction J is modified before being executed.

Another example is the following method of simulating a 'load immediate' instruction in MARIE:

(in this program, "@" means "address of")

```
                load AddData
                subt AddZero
                store DataAddr        / DataAddr = @Data
                load DataAddr
                add Three
                store ElementAddr     / ElementAdd = @Data[3]
                loadi ElementAddr
                output                / outputs Data[3]
                halt
AddZero,        add 0
AddData,        add Data
DataAddr,       dec 0
ElementAddr,    dec 0
Data,           dec 3
                dec 5
                dec 7
                dec 9
Three,          dec 3
```