Programming Practice: Logical Operations and Assembly Code

Lecture Objectives:

1) Define instruction format
2) List the MIPS instruction fields and explain their purpose
3) Explain the MIPS logical operations
4) Explain how MIPS implements a bitwise not operation.
5) List the MIPS conditional branch statements.
6) Draw a flowchart showing conditional branches executing.
7) Construct while, do-while, and for loops in MIPS assembly language

Sample Assembly Code

D:\DropBox\ClassPreps\Winter20122013\CS2710\demos\Lecture8Demo1.asm - MARS 4.2

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Edit Execute

Lecture8Demo1.asm

```
1  #This program demonstrates a simple Add operation
2
3
4
5  .code
6  # A label for main, where the program starts.
7  main:
8
9          li $t0, 0x5A     # Load the value ox5A into $t0
10         li $t1, 0xA5     # Load the value 0xA5 into $t1
11         add $t2, $t0, $t1       # Add the 2 values together.
12
13         li $v0, 10       # Exit the system.
14         syscall
```

Line: 13 Column: 31 ☑ Show Line Numbers

Mars Messages | Run I/O

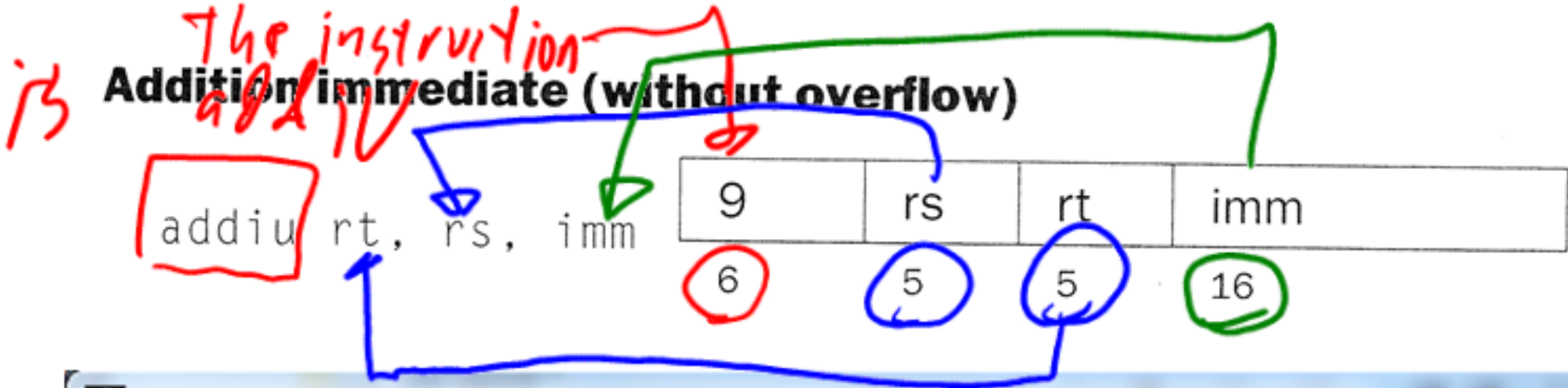
Clear

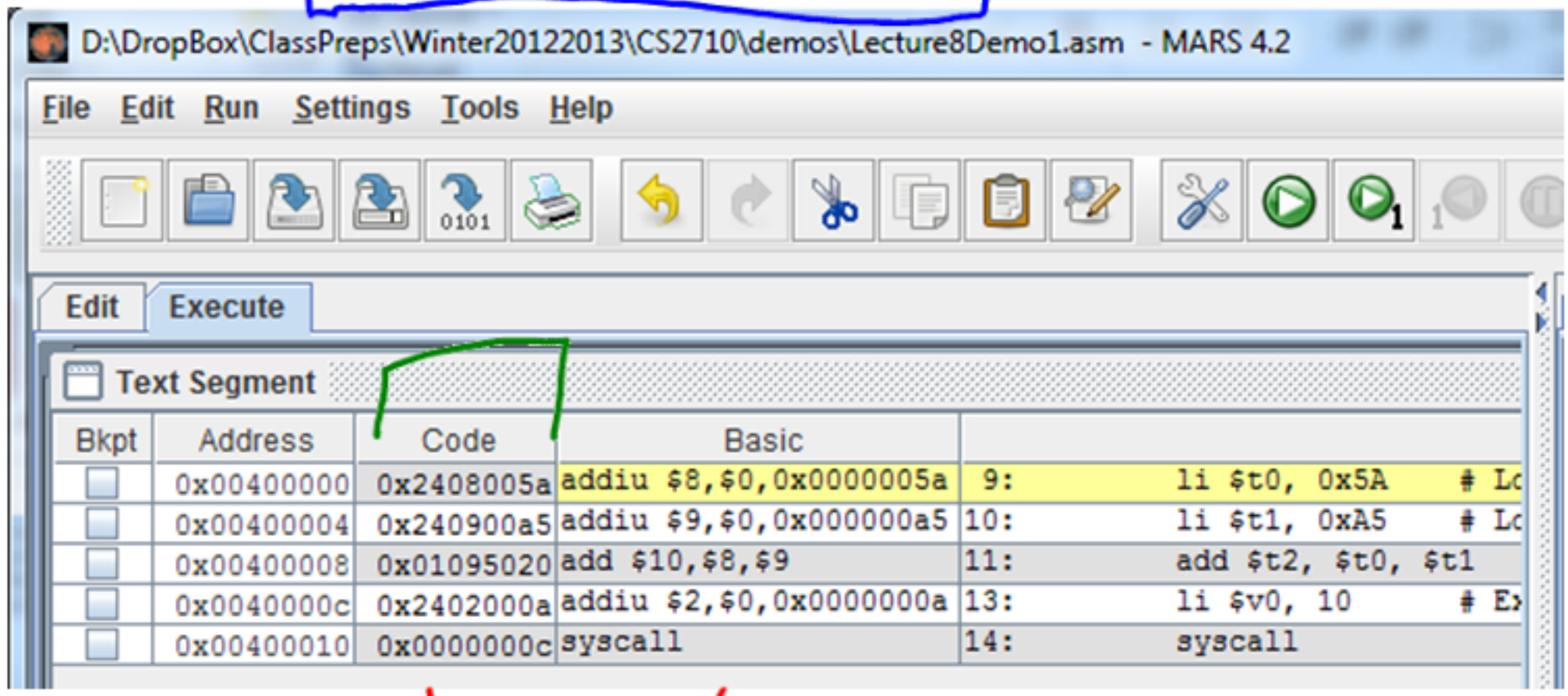
Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x0000005a
$t1	9	0x000000a5
$t2	10	0x000000ff
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400014
hi		0x00000000
lo		0x00000000

The Code Segment

Addition immediate (without overflow)

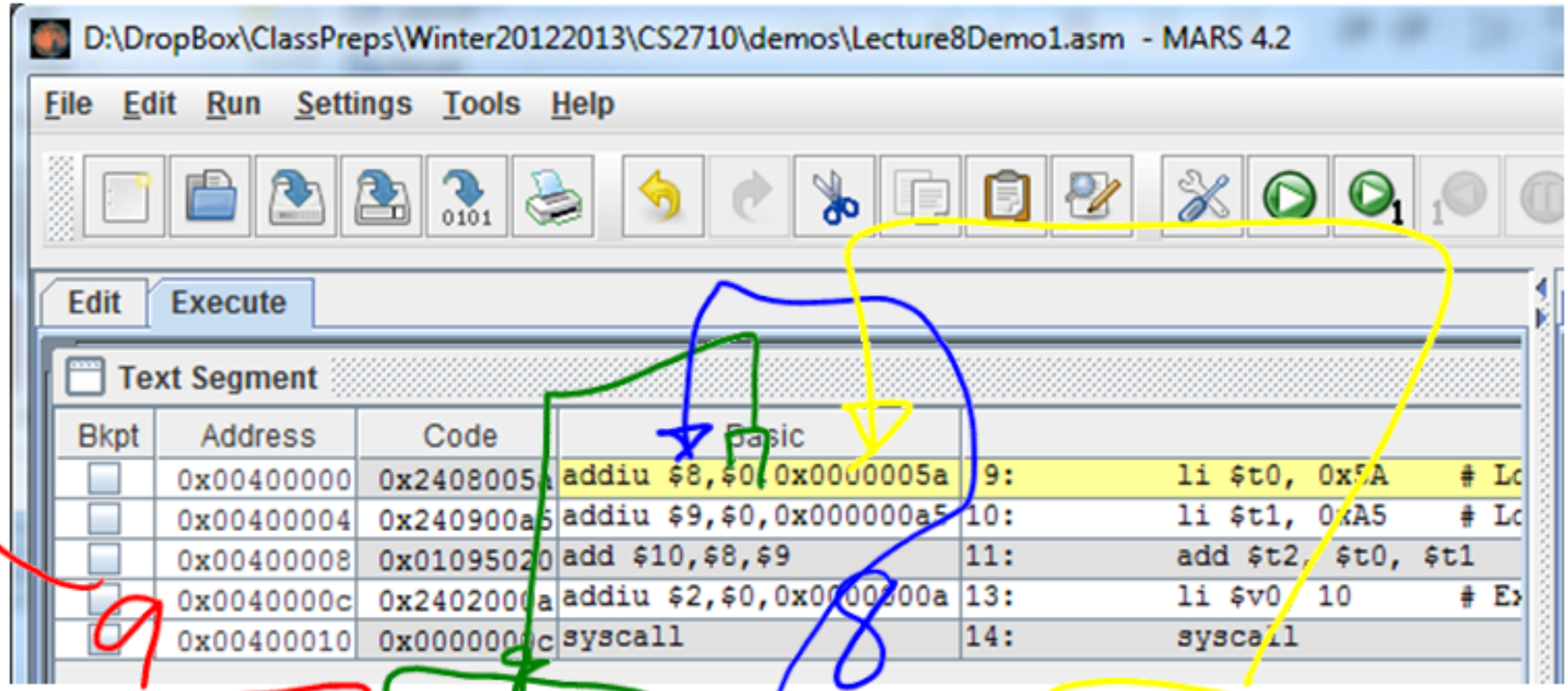




Addition immediate (without overflow)

```
addiu rt, rs, imm
```

9	rs	rt	imm
6	5	5	16



The Code Segment

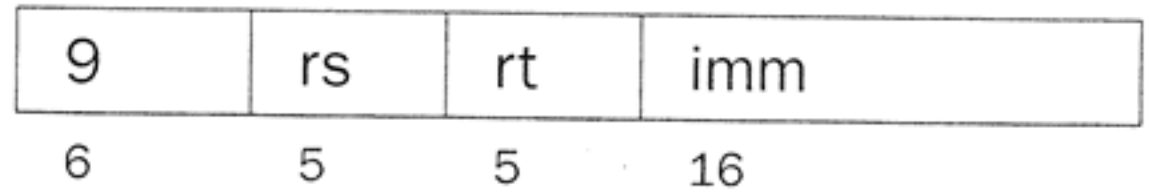
0010 0100 0000 1000 0000 0000

0x 5 A

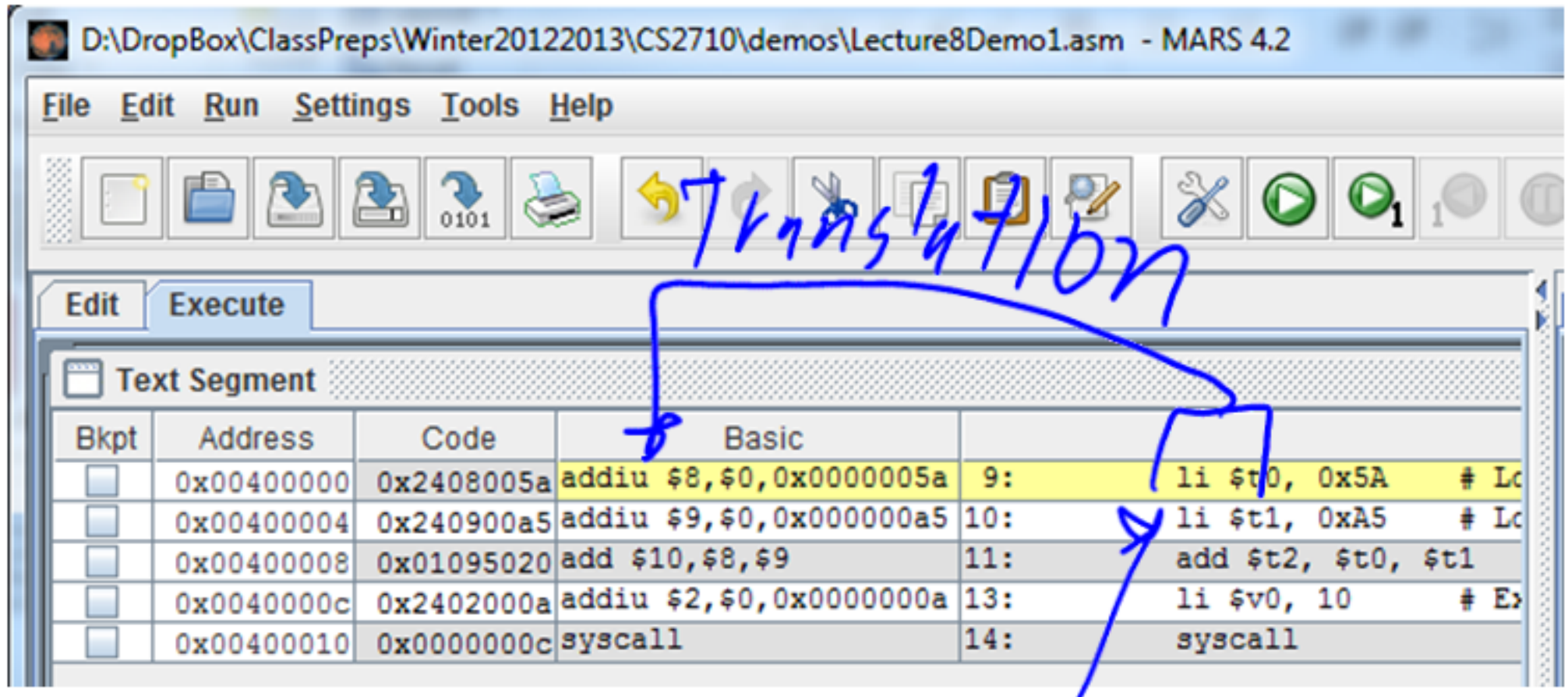
0101 1010

Addition immediate (without overflow)

`addiu rt, rs, imm`



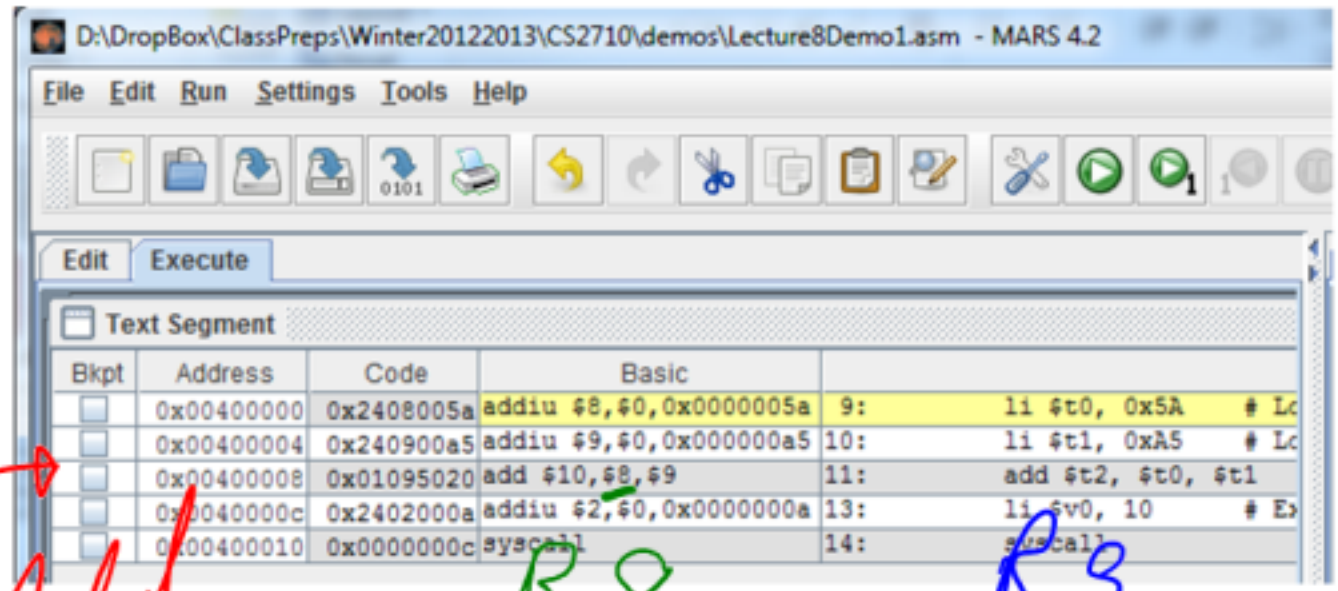
The Code Segment



Addition (with overflow)

`add rd, rs, rt`

0	rs	rt	rd	0	0x20
6	5	5	5	5	6



The Code Segment

Add

R8

R9

R10

0000 0001 0000 1001 0101 0000

0010 0000

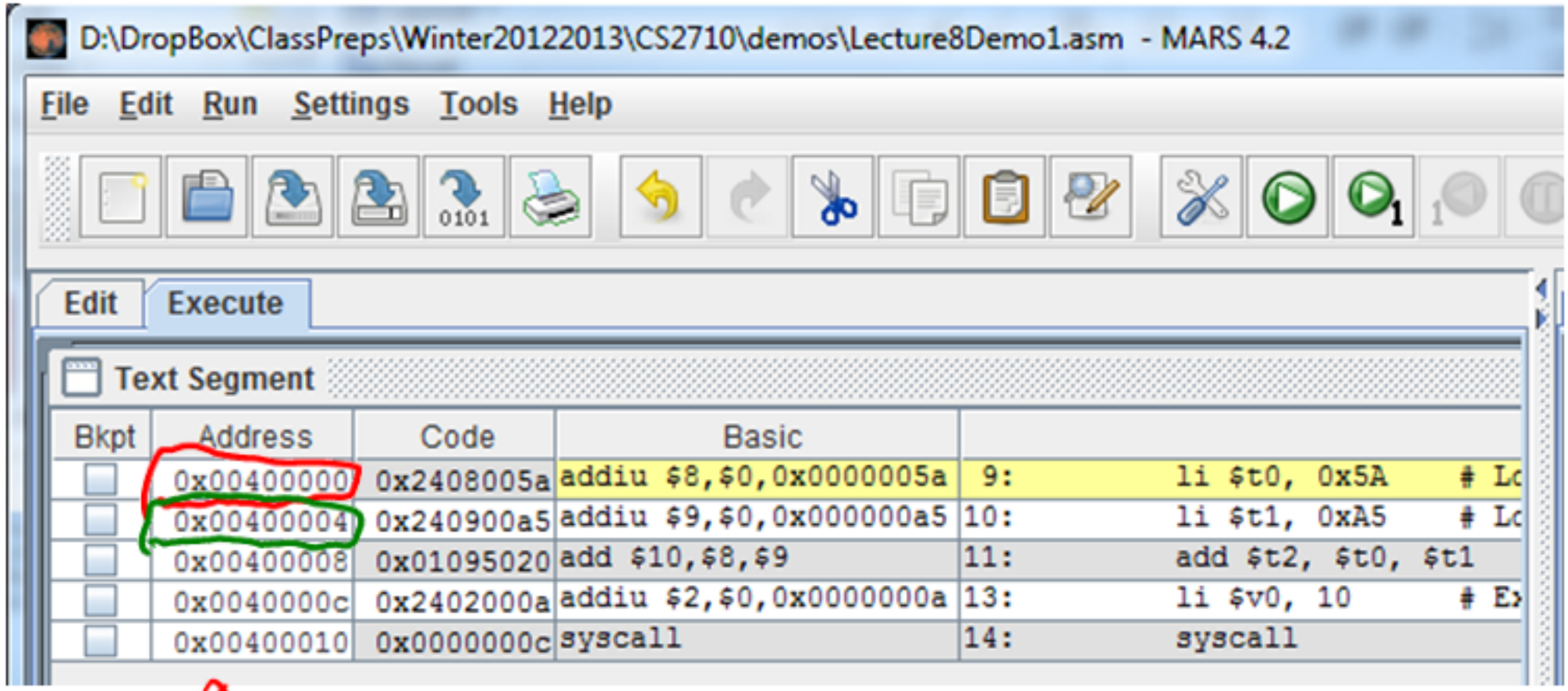
0x20

Addition (with overflow)

```
add rd, rs, rt
```

0	rs	rt	rd	0	0x20
6	5	5	5	5	6

The Code Segment



↑ where the code resides in memory

How do we compliment the bits?

- How could we write code to represent a negative number?
  - Example I want to determine the 2's compliment of a number.
  - Lets see this at work.

Not

1. Complement The Number.

2. Add One to The Number.

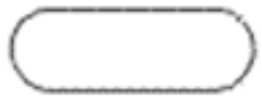
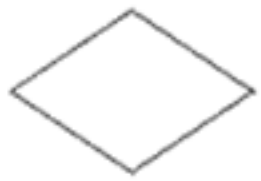
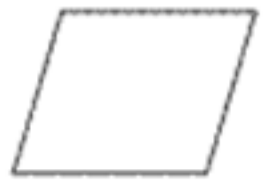
ADD7

In Class Exercise

- As a class, we are going to write an assembly language program. The program will perform the following:
  - Prompt the user to enter a number
  - Determine the value for $\sum_{1}^{n} n$
  - Determine the solution for 2^n

- We will start by drawing a flowchart for this program…

Start/End

	**Terminator**: Used at the beginning and end of a program or subroutine.
	**Process**: Such as performing arithmetic. ← Do things
	**Decision**: Used for statements that cause the computer to choose between two or more paths. The IF statement is an example of this. ← Go to
	**Input/output operations**. } Prompts/ different Input's
	**Subroutine call**: The computer temporarily runs the instructions in the subroutine and then returns to the instruction immediately following the subroutine call. Flowcharts for subroutines can be on different pages than the flowchart for the main program.

Design

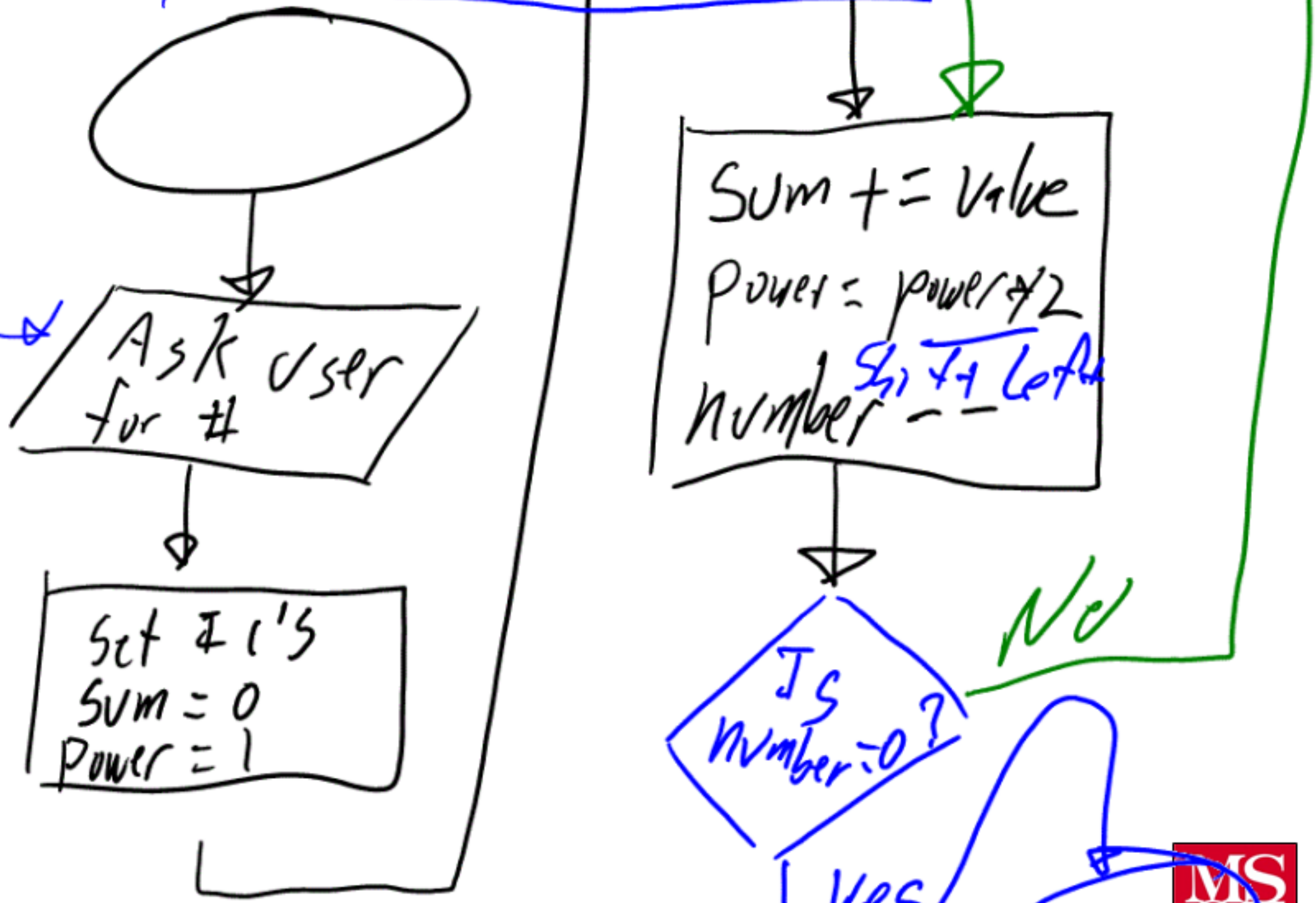
Method invocations

We need this later

MSOE

Starting implementation

- Once the flowchart is done, transfer the flowchart to code ~~as comments~~.



Implement the code

What did we learn?

```
Loop:   add $t0, $s0, $s1 # Add a couple of things together

        addi $s1, -1      #decrement s1

        bgtz $s1, loop    # loop if greater than 0.
```