The following MIPS instruction adds the values inside of registers $t2 and $t3 and stores the result in $t1:

add	$t1	$t2	$t3

One thing you'll be asked to do in this class is to convert this instruction to machine code, which is what the assembler does for us. To do this you'll need the MIPS Green Sheet to decode the instructions.

The first thing you'll notice if you lookup the 'add' instruction is that it is an R-type instruction:

CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT	OPERATION (in Verilog)		OPCODE / FUNCT (Hex)
Add	add	R	R[rd] = R[rs] + R[rt]	(1)	$0 / 20_{hex}$

Looking further down the sheet you will see the instruction format for an R-type instruction:

BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct
	31 26	25 21	20 16	15 11	10 6	5 0

The first thing we'll do is look up the opcode, which 6-bits long, for the 'add' instruction: 0_{10} = 000000_2.

Next we'll decode the registers. Before we do, let me point out an important subtelty between MIPS syntax and the instruction format. Notice that the 'add' instruction has the three registers in the order: rd, rs, rt. But notice the R-type instruction format has them in a slightly different order: rs, rt, rd. Applying this to our example, rs = $t2, rt = $t3, and rd = $t1. I hope this distinction is clear! Ok, moving on! Referring to our Green Sheet again, we see the following mapping of registers to integers:

REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	No

Since \$t0 = 8, that means \$t1 = 9_{10}, \$t2 = 10_{10}, and \$t2 = 11_{10}. Converting these to 5-bit binary values (as the instruction format tells us to do), we get: \$t1 = 01001_2, \$t2 = 01010_2, and \$t3 = 01011_2. Great!

The next step is simple. In nearly all cases, the 5-bit shamt (shift amount) will be 0_{10}, which is 00000_2. Done.

Finally, we need to decode the 6-bit function code. For this we just consult the Green Sheet for the 'add' instruction again and we see that the function code is 20_{16} = 32_{10} = 100000_2.

Now that we've decode all of the fields (highlighted in yellow), let's assemble them according to the R-type instruction format:

000000	**01010**	**01011**	**01001**	**00000**	**100000**
opcode	rs	rt	rd	shamt	func

The 32-bit binary string on the top row is the machine code equivalent of the MIPS instruction!

You will sometimes be asked to give the hex representation of an instruction. Since 1 hex digit = 4 binary bits, let's break the 32-bit string up into 8 4-bit chunks and then convert to hex:

0000	0001	0100	1011	0100	1000	0010	0000
0	**1**	**4**	**B**	**4**	**8**	**2**	**0**

And the bottom row of the above table is the hex representation of the MIPS 'add' instruction!