MIPS Homework 01.

Translating machine language to assembly language and vice versa.

Example:

Translate the following high-level code into assembly language. Assume variables **a–c** are held in registers \$s0–\$s2 and **f–j** are in \$s3–\$s7.

a = b - c;

f = (g + h) - (i + j);

Solution:

The program uses four assembly language instructions.

```
# MIPS assembly code
# $s0 =a, $s1 = b, $s2 = c, $s3 = f, $s4 = g, $s5 = h, $s6 = i, $s7 = j
sub $s0, $s1, $s2 # a = b - c
add $t0, $s4, $s5 # $t0 = g + h
add $t1, $s6, $s7 # $t1 = i + j
sub $s3, $t0, $t1 # f = (g + h) - (i + j)
```

Exercise:

Translate the following high level code into assembly language. Assume variables a-c are held in register \$s0-s2. Test resulting program in MIPS emulator.

c = c + c

b = (c + a) – (a + a)

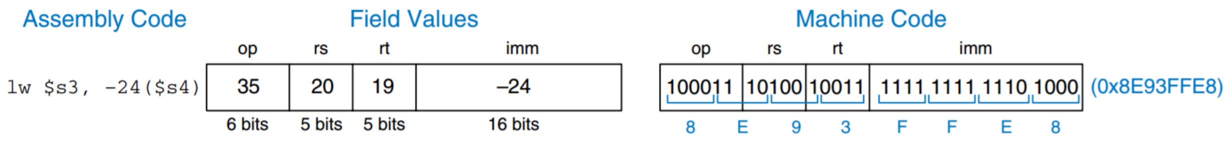
Example:

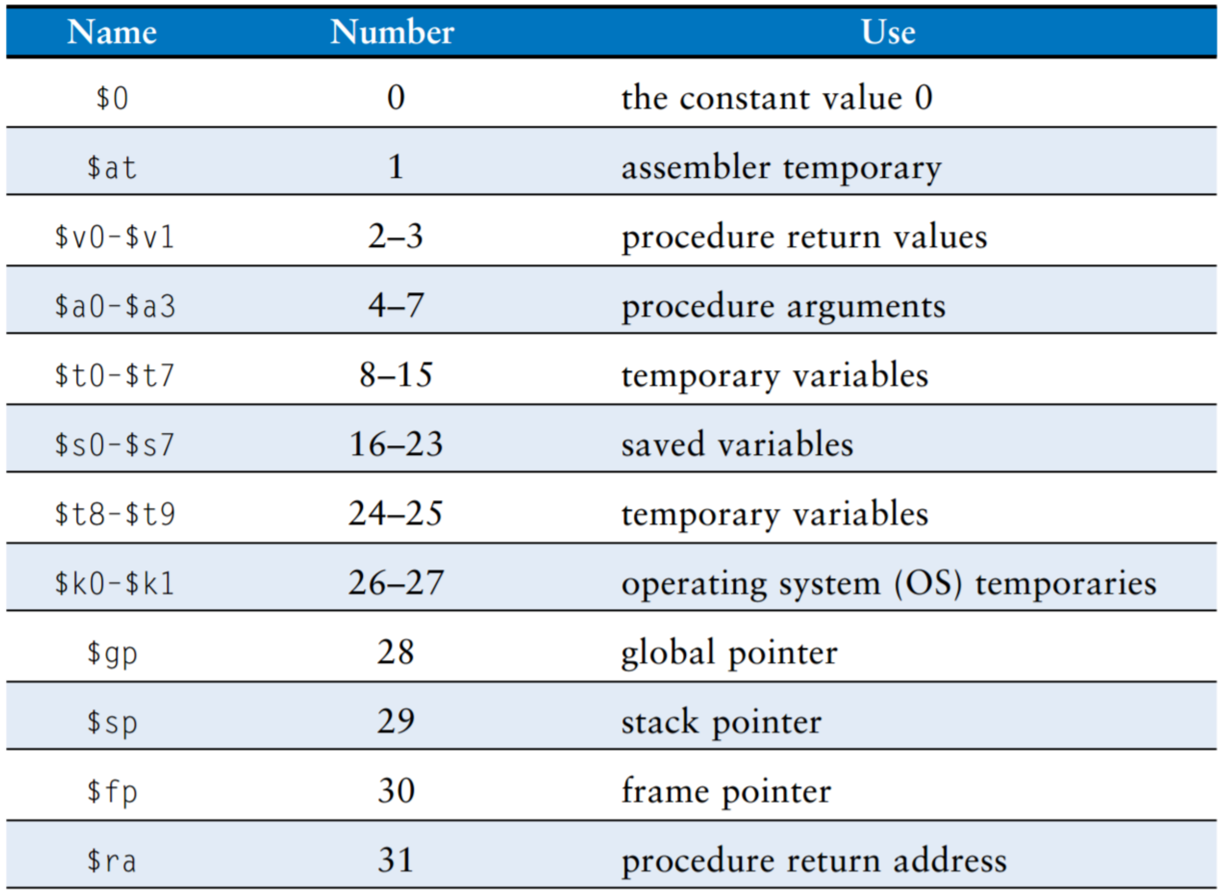
Translate the following assembly language statement into machine language.

add $t0, $s4, $s5

Solution:

According to table of register numbers, $t0, $s4, and $s5 are registers 8, 20, and 21. According to table of MIPS instructions, add has an opcode of 0 and a funct code of 32. Thus, the fields and machine code are given in figure below. The easiest way to write the machine language in hexadecimal is to first write it in binary, then look at consecutive groups of four bits, which correspond to hexadecimal digits (indicated in blue). Hence, the machine language instruction is 0x02954020





Name	Number	Use
$0	0	the constant value 0
$at	1	assembler temporary
$v0-$v1	2–3	procedure return values
$a0-$a3	4–7	procedure arguments
$t0-$t7	8–15	temporary variables
$s0-$s7	16–23	saved variables
$t8-$t9	24–25	temporary variables
$k0-$k1	26–27	operating system (OS) temporaries
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	procedure return address

Exercise:

Translate the following MIPS assembly into machine language. Write the instructions in hexadecimal. Compare results with the output of MIPS compiler.

```
add $t0, $s0, $s1

lw $t0, 0x20($t7)

addi $s0, $0, -10
```

Example:

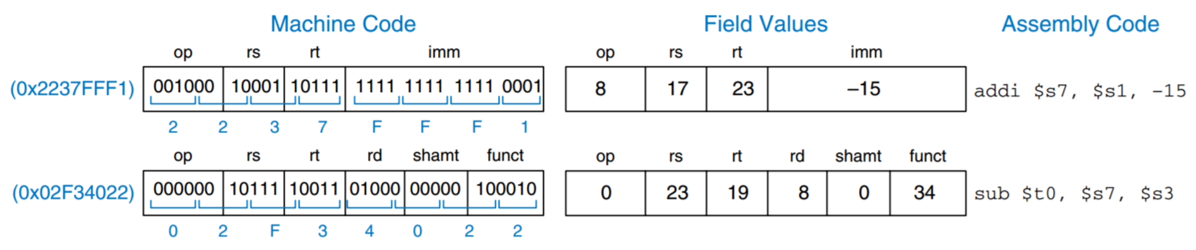
Translate the following machine language code into assembly language.

0x2237FFF1

0x02F34022

Solution:

First, we represent each instruction in binary and look at the six most significant bits to find the opcode for each instruction, as shown in figure below. The opcode determines how to interpret the rest of the bits. The opcodes are 0010002 (810) and 0000002 (010), indicating an addi and R-type instruction, respectively. The funct field of the R-type instruction is 1000102 (3410), indicating that it is a sub instruction. Figure shows the assembly code equivalent of the two machine instructions.



Exercise:

Translate the following assembly into high level code.

0x2001FFF4

0x02218822