Assignment 1: Instruction definitions

The table below gives the full set of instructions for our simulator and defines the behavior for each. Note that an 'x' in the reg1, reg2, or imm column indicates that those fields are used.

Name	opcode	reg1	reg2	imm	Example	Function
subl	0	x		x	subl $8, %esp	reg1 = reg1 - imm
addl	1	x	x		addl %edi, %edx	reg2 = reg2 + reg1
addl	2	x		x	addl $8, %esp	reg1 = reg1 + imm
imull	3	x	x		imull %edx, %eax	reg2 = reg1 * reg2
shrl	4	x			shrl %edi	reg1 = reg1 >> 1 (logical shift)
movl	5	x	x		movl %eax, %esi	reg2 = reg1
movl	6	x	x	x	movl 20(%esp), %eax	reg2 = memory[reg1 + imm] (moves 4 bytes)
movl	7	x	x	x	movl %eax, 20(%esp)	memory[reg2 + imm] = reg1 (moves 4 bytes)
movl	8	x		x	movl $6, %esi	reg1 = sign_extend(imm)
cmpl	9	x	x		cmpl %ebx, %ecx	perform reg2 - reg1, set condition codes does not modify reg1 or reg2
je	10			x	je foo	jump on equal
jl	11			x	jl foo	jump on less than
jle	12			x	jle foo	jump on less than or equal
jge	13			x	jge foo	jump on greater than or equal
jbe	14			x	jbe foo	jump on below or equal (unsigned)
jmp	15			x	jmp foo	unconditional jump
call	16			x	call foo	%esp = %esp - 4 memory[%esp] = program_counter + 4 jump to target
ret	17				ret	if %esp == 1024, exit simulation else program_counter = memory[%esp]

						%esp = %esp + 4
pushl	18	x			pushl %ebx	%esp = %esp - 4 memory[%esp] = reg1
popl	19	x			popl %ebx	reg1 = memory[%esp] %esp = %esp + 4
printr	20	x			printr %eax	print the value of reg1
readr	21	x			readr %eax	user input an integer to reg1

Condition Codes

The *cmpl* instruction sets the condition codes (CF, ZF, SF, OF) corresponding to certain bits in the %eflags register. The conditional jump instructions read these codes and either branch to the target or move to the subsequent instruction, based on the appropriate flags.

When the *cmpl* instruction is executed, reg2 - reg1 is performed to determine these condition codes:

	Condition	**%eflags bit**
CF (carry)	If reg2 - reg1 results in unsigned overflow, CF is true	0
ZF (zero)	If reg2 - reg1 == 0, ZF is true	6
SF (sign)	If most significant bit of (reg2 - reg1) is 1, SF is true	7
OF (overflow)	If reg2 - reg1 results in signed overflow, OF is true	11

These codes are stored using certain bits of the %eflags register. For example, consider the following sequence of instructions:

```
movl    $1, %eax
movl    $2, %ebx
cmpl    %ebx, %eax
```

After execution of the *cmpl* instruction, the %eflags register should contain the value 129 (0x81) because:

- The result of %eax - %ebx is 0xFFFFFFFF.
- The most significant bit is set (SF).
- Unsigned overflow occurred (CF).
- A number with the seventh bit (SF) and zeroth bit (CF) set is 128 + 1 = 129.

Jumping/Branching Offsets

je, *jl*, *jle*, *jge*, *jbe*, *jmp*, and *call* instructions use an immediate offset, which is the target instruction address relative to the **next** instruction's address.

For example, consider these instructions:

```
start:
    jl label1
    printr %eax
label1:
    printr %ebx
    jmp start
    ret
```

The address of the "start" label (the first *jl* instruction) is 0. The address of "label1" is 8.

The immediate value contained in the *jl* instruction is 4 because "label1" starts 4 bytes away from the **next** instruction **after** *jl*.

The immediate value contained in the *jmp* instruction is -16, since the "star"t label is -16 bytes away from the **next** instruction **after** *jmp*.

Conditional Jumps

The *je*, *jl*, *jle*, *jge*, and *jbe* instructions are conditional jumps. This means they move to the next instruction (program_counter + 4) if the condition is false, or they jump to their target offset if the condition is true.

The conditions are as follows:

Jump Type	**Jump If**
je	ZF
jl	SF xor OF
jle	(SF xor OF) or ZF
jge	not (SF xor OF)
jbe	CF or ZF

I/O instructions

There are two special (fake) instructions to help with I/O:

- *readr* (read register)
- *printr* (print register)

The course staff will primarily use *printr* to determine if your simulator is correct. These instructions are already implemented for you.