Some Intel assembly instructions

The following is a subset of gcc-supported Intel instructions that we are likely to use in CSC 220. There is a larger list on page 398, Figure 5-33 of our text.

If an instruction below ends with a *, then the true instruction name ends with a l, w, or b depending on the size of its operands. Here's the rule:

- "long" - 4 byte operands, instruction ends in 'l'
- "word" - 2 byte operands, instruction ends in 'w'
- "byte" - 1 byte operands, instruction ends in 'b'

We will mostly use long int commands in class. Remember - most commands do not allow 2 memory references as parameters. Parameters must be register-register or register-memory.

1. Data movement instructions

Instructions that shuffle data around.

Instruction	Description	Example
`mov* src, dest`	Move a value from memory to/from a register dest = src	`movl $17, %eax`

`xchg* src, dest`	Exchange values
	
	dest = src
	src = dest	

`push* src`	Push a value on the stack mem[stack] = src decrease %esp	`pushl %edx`
`pop* dest`	Pop a value from the stack dest = mem[stack] increase %esp	`pop %ebp`

2. Arithmetic instructions

Instructions for adding and such.

Instruction	Description	Example
`add* src, dest`	Add dest = dest + src	`addl $4, %esp`
`sub* src, dest`	Subtract dest = dest - src	`subl $8, %esp`
`inc* dest`	Increment dest++	
`dec* dest`	Decrement dest--	

`imul* value`
`mul* value`
	Integer multiply (signed and not)
	%eax = %edx:%eax * value
		`mull %ecx`

`idiv* divisor`
`div* divisor`
	Integer division (signed and not)
	%eax = %edx:%eax / divisor
		`divl %ecs`

3. Logical/Boolean instructions

These instructions perform bitwise Boolean operations.

Instruction	Description	Example
`and* src, dest`	Boolean and dest = dest AND src	
`or* src, dest`	Boolean or dest = dest OR src	
`xor* src, dest`	Boolean exclusive-or dest = dest XOR src	
`not* dest`	Boolean inversion dest = NOT dest	

4. Comparison and Jumps

These instructions simulate subtracting the source from the destination and set the flags in the eflags register. They are commonly used to create if-then-else blocks and loops. Jump addresses are usually labels in your assembly program.

Instruction	Description	Example
`cmp* src, dest`	Compare two values dest - src ==> set eflags reg	`cmpl $0,%eax`
`jmp addr`	Unconditionally jump to an address	`jmp LOOP_TOP`
`jz addr`	Jump if eflags is "zero"	`jz ELSE_BLOCK`
`jnz addr`	Jump if eflags is "not zero"	
`je addr`	Jump is eflags is "equal"	
`jne addr`	Jump if eflags is "not equal"	
`jl addr`	Jump if eflags is "less than"	
`jle addr`	Jump if eflags is "less than or equal"	
`jg addr`	Jump if eflags is "greater than"	
`jge addr`	Jump if eflags is "greater than or equal"	

5. Function-related instructions

Instructions related to handling function calls and returns. Remember: by convention, function return values are placed in register `%eax`.

Instruction	Description	Example
`call func`	Call function; pushes return address onto stack	`call _printf`
`ret`	Return from function; pops the return address that must be on the stack	`ret`

`leave`	Prepares for ret instruction, just like:
	`movl %ebp,%esp`
	`popl %ebp`
		`leave`