Translation of Arithmetic Operations

Let's us see how an arithmetic expression is translated into assembly code. Take example of this C code:

```
int a = 2;
int b= 3;
int c = 24;
a = a + b;
a = a + b * c;
```

Generated assembly code:

```
movl $2, -4(%ebp)
movl $3, -8(%ebp)
movl $24, -12(%ebp)
movl -8(%ebp), %eax
addl  %eax, -4(%ebp)
movl -8(%ebp), %eax
imull -12(%ebp), %eax
addl  %eax, -4(%ebp)
```

Location of local variables of the stack (local variables are explained here (memorymanagement.html)).

```
a => ebp-4 => -4(%ebp)
b => ebp-8 => -8(%ebp)
c => ebp-12 => -12(%ebp)
```

Comments on the generated assembly code:

```
# a = 2
    movl    $2, -4(%ebp)
# b = 3;
    movl    $3, -8(%ebp)
# c = 24
    movl    $24, -12(%ebp)
# tmp = b
    movl    -8(%ebp), %eax
```

The i386 has limitations on the number of memory addresses in a single instruction. It can not use an arbitrary number of memory addresses in an instruction. So it uses registers as temporary storage. It does arithmetic and other other kinds of operations using registers. A typical way of using the register is that first bring the content from memory to registers, do operation in registers and then store back the the value of registers into the memory. In the example below, the register eax is being used as temporary storage. In the commented code, tmp means the register eax.

```
# a = a + tmp
   addl    %eax, -4(%ebp)
# tmp = b
   movl    -8(%ebp), %eax
# tmp = tmp * c
   imull   -12(%ebp), %eax
# a = tmp + a
   addl    %eax, -4(%ebp)
```

QUESTION: How will an arithmetic expression translated into assembly code if the arithmetic expression contains brackets?

‹ Stack and Local Variables (/cin/memorymanagement.html)
up (/cin/cin.html)
Translation of Bitwise Operations › (/cin/bitwiseop.html)
