Left shift multiplication assembly

I am currently trying to understand how to multipli by left shifting in assembly, and therefore solving some assignments. I was asked to rewrite some calculations to not using `mul` or `div`.

Example:

```
mul $2, %eax
```

My answer:

```
sal $1, %eax
```

I understand that by shifting left once, this one is answered. But another example says

```
mul $3, %eax
```

and

```
mul $5, %eax
```

How do i solve that? From what i have learned its something like splitting it and then add it together somehow.

Anyone? Thanks!

assembly compilation

edited May 25 at 14:45
alk
44k 5 33 95

asked May 25 at 14:38
Christian Larsen
1

3 Answers

You need to decompose the multiplication by an addition and a shift operation, eg

```
x*3 == x*2 + x
```

Then it becomes trivial

```
mov %eax, %ebx # tmp = eax
sal $1, %eax # eax <<= 1
add %ebx, %eax # eax += tmp
```

answered May 25 at 14:43
Jack
88.8k 19 137 248

By shifting you can only multiply/divide by powers of `2`.

```
x * (2^n) = x << n
x / (2^n) = x >> n
```

If you want to multiply `x * y`, while `y` is not a power of two, you will need "decompose" it to powers of two, multiply by each component and add the result. Luckily, any number can be represented as a sum of powers of two. And guess what? The binary representation is telling you exactly how. For example:

```
y = 12 (not a power of two).
```

The binary representation will be

```
y = 1100
```

Now take the positions where ones are: `2, 3` (start with `0` on the right). It means that

```
y = 2^2 + 2^3.
```

Now, to multiply `x` by `y` you just need to:

```
x * y = x * (2^2) +  x * (2^3) = (x << 2) + (x << 3)
```

And you have replaced the multiplication with shifting and addition.

(BTW, if you are familiar with the long multiplication method for binaries, you will recognize it here...)

answered May 25 at 14:47

Eugene Sh.
7,980 9 33

---

this question was just recently asked and answered...

from grade school we know that

```
n * 3 = n * (2 + 1) = n * ((2^1) + (2^0)) = (n*(2^1)) + (n*(2^0))
```

and then you can apply the shifting

```
(n<<1)+(n<<0)
```

same for 5 which is 0b101 or 2^2 + 2^0.

you can also do this with long multiplication again from gradeschool but with binary is much simpler

```
    nmop
*   0101
=========
    nmop
   0000
  nmop
+0000
=========
```

(where n,m,o.p are individual bits) with decimal (grade school) you took the digit on the bottom and multiplied it by the whole top, the shifted it over one column per power of 10 of the bottom, same in binary but with base two you have only two choices multiply by 0 or multiply by 1, so either you dont add something in or you add in one times the number shifted over by the right number of columns.

either way you end up with the same result, pick one of the operands, for each binary bit that is a one, you shift the other operand by that many places, and add it all up.

division is a completely different beast, you can shift right to divide by powers of two, but it doesnt distribute the same way n / 5 != n/4 + n/1. get the hackers delight book on some tricks. Also note if it is a power of two then it only generally works with unsigned numbers, with signed numbers you often see zeros shifted in from the left and not a repeat of the sign bit. in assembly on some architectures there is an arithmetic shift right vs a logical shift right, the logical often shifts in the zero and this is the shift often used by high level languages, the arithmetic ideally repeats the sign bit. allowing you to use shift right as a power of two divisor for signed (but not necessarily unsigned)

edited May 25 at 15:11

answered May 25 at 15:06

dwelch
36k 5 44 96
