IT UNIVERSITY OF COPENHAGEN

OPERATING SYSTEMS AND C

Solution for exam, category SWU fall 2022

Author
Jan PIROUTEK
Initials
jpir

November 28, 2022

Contents

My exam doesn't content snipets of code, but mostly the idea behind it. If the use of actuall code will be necessary, I will be referencing files, that I have handed in, so you can access them on github

Question 1 - Data lab

A. Describe your implementation of *howManyBits(x)*

The easiest solution for this would be create mask, that isolates each bit of the number, then take that bit and use the *!* twice (*!!*. Single *!* gives 1 if the number is only full of zeros, which would be if the bit, for which the mask was created was set to 0. So for counting them I need to negate the value again, so I would get 0 in this case and 1 otherwise. Then I can just sum those numbers together with *+* operator.
This solution is easy, but takes a lot of operations, so I needed to be smarter.
Let's start from broader perspective. If I want to represent the number in the twos complement I'll always need one bit to basically say if the number is positive or negative (MSB). And what with the rest of the number.
Now lets think little bit more how twos complement works. If I have some number represented in twos complement, then first bit of the number is giving the value we need to subtract from the value stored in the next $x - 1$ bits. Lets say I have two numbers x and its absolute value $y = |x|$. If x is positive, then $y = x$ and for representing x as twos complement. I just need as much bits as is the highest position of bit set to one (e.g. $x = 7 = 0b111$ I need 3 bits), plus 1 as the leading 0 for MSB.
Lets think about case when x is negative. Then I know the MSB is set to one, and the rest of the number tells me, how much I have to subtract from $2^{position\_of\_MSB}$. Neat thing about twos complement is this thing tho, if I have some negative value x, the positive value of $-x$ is also representable as twos complement, except for $0b10000\cdots$, where there are only zeros after MSB one. What if I just add one to the negative number?
I will state this -¿ if I increment negative number of twos complement by one, it will still need same amount of bits for representation as the old one.

- $-8 + 1 = -7$ and $-8 = 0b1000$; $-7 = 0b1001$
- $-3 + 1 = -2$ and $-3 = 0b101$; $-2 = 0b110$
- $-1 + 1 = 0$ and $-1 = 0b1$; $0 = 0b0$

This way I can just get the value of $x + 1$ and compute how many bits I need for the absolute value $|x + 1|$ (x is still negative). How to easily get this value? Twos complement is really nice to me. If I want value of $-x$ all I need to negate all bits and add 1. But what I want is $(\ x + 1) - 1 = \ x$.

So now I just need to get the absolute value of x and compute the position of first bit, that is set to one. Lets split it this into two parts.

Getting absolute value

First I need to know if the number is negative or positive. If x is positive I want to get x, otherwise I want to get $\ x$. First I isolate MSB with mask of $0x80 << 0x18$ (minimal number explained more in next subquestion). Then I can reuse code of conditional function (one of the other tasks). Conditional would be if x after mask is 0 or not. If it is 0, the x was positive, negative otherwise. Then I can flip the condition, negate it and add 1. This way I will either get integer consisting of all ones or all zeros. In the end I just use this condition as mask for x or $\ x$ and pick the desired one. One of them will be turned to all zeros, other will remain the same. So I can just use | to get x I want. Now I have aquired the requested value.

Computing number of bits

Now when I have the positive value all I need to do is to get position of first 1. First a copy the first bit to the right of it, in $\log_2(32)$ steps. Starting from one I shift all bits to the right by one and take $x|(x >> 0x01)$. This copies the most left one to the next bit. Then I do the same but with $0x02$, that copies the leftest one and its copy from last step to the next two bits. Then with $0x04$ and so on.
know I have copied the leftest one and I need to count how many bits I have in number. This either can be slow or I can use divide and conquer technique. Each bit has information how many ones are in him (one if set to one, zero if set to zero). Now what I do is to align bits, that I want to add and use mask to separete odd and even bits. In other words I move even bits under odd bits and add them together. Now I know how many bits are in the two consecutive bits. Then I do the same for the pairs of bits. Align (shift by two this time) and mask them (I need a different mask. First mask needed to mask even bits $\rightarrow 0b0101010101\cdots = 0x55555555$, this needs to mask even pairs $\rightarrow 0b00110011\cdots = 0x33333333$. Next for the pairs of four bits, 8bits and in the end 16bits. But because I didn't wanted to use more mask than necessary, my code could have left some bits set in first part of the number. So I needed to mask the remaining bits, so it wouldn't return bigger numbers than 32 (maximal amount of ones in integer).

In the end only thing I need is to add the 1 bit, that I have not acounted for earlier (the MSB of twos complement, saying if it is negative or positive)

B. Describe your implementation of *tmin(void)*

From definition of twos complement. the minimal number is composed of bits, where each bit is set to one.
The function should return minimal integer, so that means I need 32 bits, all set to one. Largest number I can use, which has all bits set is $0x80$ (-128). Then I can use the left shift (<<) operator in C, which moves bits to the left, direction to highest bit, and copies their value to the old place. So if I do $0x80 << 0x01$, shift 8 bits set to one to the left by one place, I get 9 bits all set to one, if it doesn't overflow. Integer has 32 bit, so to get them all set to one I just need to shift $0x80$ by $0x18$ (24) places. This gives me integer with all bits set to one -¿ smallest number, that fits into integer.

Question 2 - Attack lab

A. What happens when the c3 (ret) assembly instruction is executed? Does anything in the stack change?

B. What is a gadget farm? Describe an example of how you use one in your code.

Question 3 - Malloc lab

A. Explain in detail your implementation of the `mm_malloc` function.

B. What is pointer arithmetic? Describe how you use it in your version of mm.c

Question 4 - Topics from the class

A. What is the difference between traps, faults and aborts in the context of interrupts?

B. What is the difference between an ephemeral and a well-known port? Give examples of when either is used.

C. What is a memory leak? When does it occur? What can you do to avoid it?

D. What is a race-condition? Why is a race-condition hard to debug? Which instructions can you use to avoid race-conditions? Why are these instructions expensive?