

## — COMPUTIST QUIZ ANSWER GUIDE —

The following informal answers riff on questions in the “Computist Quiz”. The level of detail and technicality varies to maximize congeniality and appeal, rather than rigor. Additional or alternative responses are solicited!

Just Four Chars. If the two’s complement representation of  $x$  is written

...???100...00

with however many trailing 0s (including none) then the two’s complement for  $-x$  will be

...??100...00

where the  $??$ s stand for the complement of the bits represented by the  $???$ s above. ANDing these gives

...000100...00

essentially clearing all those unknown bits to the left. This leaves a result with a lone 1 bit, which is of course some power of two. Thus the answer we are seeking is that the expression  $x \& -x$  essentially computes the largest power of two that divides the input number  $x$ . That is, for odd numbers it returns 1, for even numbers not divisible by 4 it returns 2, for evens divisible by 4 but not 8 it returns 4, etc. (Extra credit if you noted that  $x=0$  is an exceptional case.) Some general themes of this quiz are exemplified by this tiny expression.

First, there’s the peculiarity of applying what we normally think of as “logical bitwise” operations to get a result that is best described by *divisibility*, a concept that does not seem to appear explicitly in the source expression. We may often unconsciously project irrelevant preconceptions onto the computer, limiting our ability to lucidly perceive its essential nature, and perhaps blinding us to potentially useful computations.

Further, it exemplifies the fact that even tiny programs may do surprising things. Our normally imperative engineering orientation—always *telling* the computer what to do—might be usefully supplemented with a more “natural” computer science, that observes, analyzes and models the vast—and mostly unknown—space of what the computer *can* do. For example, what might the set of *all* such legal four-character expressions compute?

Parody Bit. This question is guilty of some “misdirection”—and seeing through it makes the answer much more obvious. The distraction is the parity of  $x$ . Ignore that, and the question simplifies to “if you keep subtracting  $q$  from  $x$ , what does it mean if the result is zero?” Obviously, it just means that  $x$  is divisible by  $q$ .

Those opportunistic halvings don’t alter this invariant, since it was given that  $q$  is an odd number. However they *do* change the expected performance of this divisibility test, improving the ultra-simple minded “divide by repeated subtraction” in an interesting way. In fact, the procedure is very similar to the “binary GCD” algorithm (Knuth TAOCP 4.5.2 Program B). (Extra credit: analyze the performance of this algorithm).

Duncing Lunks. The hack is simply to combine the two links into a single value using some handy information-preserving “reversible” operation such as XOR (or even just addition/subtraction, if your system permits you to ignore overflow!). During traversal (in either direction) you always know where you just came from, so you can use that information to recover the other link by “subtracting it out” of the combined value. This hack is interesting because it transcends the usual data typing prejudices wherein pointer arithmetic is considered dubious, and so the idea of *adding pointers to each other* is dismissed as meaningless, if not certifiably insane. (Extra credit: extend this idea to more than two pointers; discuss the relationship to error-correcting codes.)

## — COMPUTIST QUIZ ANSWER GUIDE —

++Unshuffle. This is mainly an exercise in visualization and abstraction. The key is to notice that unshuffling reorders things so that the *least* significant bit of an elements address—its parity—effectively becomes the *most* significant bit—distinguishing the first/second halves. The recursive application then takes the next least significant bit into the next most significant, and so forth. The effect of all the unshuffling is to *bit-reverse the addresses* of the elements. This allows us to answer the subsidiary questions:

The elements that wind up back in their original positions are the ones whose addresses are binary palindromes. Since there are  $2^{10}$  elements this constrains one five-bit half of the ten-bit address to be the mirror image of the other half. This tells us that there are 32 palindrome addresses out of the 1024 total.

The elements that wind up the furthest from their original positions are the pair such that the difference between the address and its bit reversal is maximized. This occurs for element addresses 31 and 992, whose difference is 961.

If you were to apply a *second* pass of the entire procedure you'd just reverse the reversed addresses, leaving everything in its original position.

A possible application for this unshuffling procedure is in a Fast Fourier Transform (Extra credit).

Siamese Sequences. The first sequence can be obtained by *inclusive*-ORing successive integers, while the second sequence is obtained by *exclusive*-ORing successive integers. In a sense they are “analogs” of the triangular numbers, with IOR and XOR replacing addition. (Sloane’s OEIS A003817 and A003815). The first sequence is somewhat opaque and non-descript, so deciphering the second sequence first is probably best. It’s interesting how distinct the “personalities” of these two strongly-related sequences are: the first is pretty much just a kind of uniform “binary pile-up” whereas the second exhibits a very pronounced “period-4” pattern.

Sordid Sort. This pretty sorting predicate can actually be implemented by a fairly simple state-machine that is not much more complicated than, and practically as efficient as, the usual system string compare. This machine has two major “modes”: scanning alphas—which is just the usual string compare—and scanning numbers—which is similar but accounts for the minor twist that a longer string of digits is greater than a shorter one. Thus it does not have to actually accumulate numerical values, but only keep track of the state transitions in the scan.

Coding this predicate (and similar state machines) elegantly and efficiently is an interesting and illuminating exercise in today’s “structured programming” systems, which eschew **goto** statements, and whose compilers might or might not apply tail-recursion removal optimizations.

Extra credit: discuss the behavior with respect to leading zeros, as in “007”. Extra *extra* credit: install this algorithm in some real-world application and start a trend! It’s curious how blithely the usual “unnatural” system string compare is simply accepted, when a better alternative is so easily implemented. (Of course that’s nothing compared to many other accepted legacies, such as the QWERTY keyboard...)

## — COMPUTIST QUIZ ANSWER GUIDE —

*Scary Natter.* The simplest analysis is to note that the odd and even bit positions essentially act like two independent adders with their bits interleaved. This answer describes the operation succinctly, modeling the operation as addition on pairs of integers, encoded by interleaving. But wait, there's more!

Using what we like to call *numbral* notation (from “umbral numeral”), let  $[n]$  denote an operand or result whose binary representation is  $n$ . Then we observe that

$$[n] \oplus [n] = [4n]$$

and in particular, looking at each individual bit, that

$$[2^n] \oplus [2^n] = 2[2^n] = [2^{n+2}]$$

“solving” for the value of the  $n$ -th bit, we get

$$[2^n] = (\sqrt{2})^n$$

That is, the  $\oplus$  operation can be viewed as implementing arithmetic in “base  $\sqrt{2}$ ”. The nice thing about this model is that it actually *interrelates* the odd and even bit positions—shifting left one bit is scaling by  $\sqrt{2}$ . This is a more fruitful interpretation than simple pairs. For example, if we now implement a multiplication analog  $\otimes$  with the usual shift-and-“add” procedure we get consistent results. So a fuller answer might be that this models arithmetic on numbers of the form  $a + b\sqrt{2}$ , encoded by interleaving the binary bits of  $a$  and  $b$ .

Possible applications might include cryptography, signal processing or Tinker Toy computers (whose rod sizes scale by powers of  $\sqrt{2}$ )

*Review.* For  $0 \leq x \leq 7$ , the expression evaluates to

$$0, 4, 2, 6, 1, 5, 3, 7$$

These are the 3-bit reversals of the input sequence. Thus a plausible answer is “the unshuffle question”. (This is just a simplified version of Richard Schroepel’s delightful bit-reversing expressions, HAKMEM Item 167). Extra credit: generalize this hack to other byte sizes and bit permutations. Extra *extra* credit: generalize to a theory that models *all* possible values  $(i,j,k)$  for the “magic constant” triple  $(65,322,15)$ .

*Amphibious Discursion.* Examining the output of the program quickly reveals that the “toads” are the primes. But then things get interesting. Many computists assume that the **frog** program *must* be doing some kind of division by iterated subtraction, much like in the “Parody Bit” question. However it *isn't*, so part of the challenge of this exercise is to set aside these preconceptions and accurately describe what it actually *is* doing.

Despite the false similarity to division, the **frog** program is counting *the ways a number  $n$  can be represented as the sum of four triangular numbers*. Of course even after carefully figuring this out most people will still wonder what the heck that has to do with primes! A theorem (Legendre, 1752-1833) says that this count is *equal to the sum of the divisors of  $2n+1$* . But (unless you’re an analytic number theory geek) asking a computist to reverse engineer this alien technology is, of course, a *completely unfair* quiz question! Nonetheless we think it makes for an interesting diagnostic “problem solving” exercise, and a lesson in (to parody Wigner’s phrase)

*“The effective unreasonableness of mathematics.”*

# — COMPUTIST QUIZ ANSWER GUIDE —

Distinctly Odd. We can show this with a bijection, transforming one kind of partition directly into the other. Start with a distinct partition. Replace every part of the form  $x = 2^n q$ ,  $q$  odd, by  $2^n$  copies of  $q$ . (Can we compute  $q$  with, say, an eight-character expression in  $x$ ?). Now all the partitions are odd. (Extra credit: show that two distinct partitions can't split into the same number of copies because of the uniqueness of binary expansions). We'd love to see a good pictorial proof of this (J. J. Sylvester apparently had one...)

Of course Euler, "Master of Us All", didn't do it this way—he showed it by equating infinite products!

The Dismal (Computer) Science. Although now more widely applied, dismal arithmetic was originally created to accommodate the special needs of people (such as ourselves) suffering from the scourge of dyscalculia:

### What Are Common Learning Disabilities?

The Public Libraries Learning Disabilities Initiative lists the following common learning disabilities:

...  
**Dyscalculia** causes people to have problems doing arithmetic and grasping mathematical concepts. While many people have problems with math, a person with dyscalculia has a much more difficult time solving basic math problems than his or her peers.  
 ...

By eliminating technical nuisances like carrying, and replacing the tedious rote learning of addition and times table "facts" with simple digit comparisons, dismal arithmetic optimizes learners' cognitive facilitation.

Anyway, for all integer  $x$ , the unique "dismal unit"  $u$  such that the dismal product  $ux$  always equals  $x$  is not 1 as in ordinary arithmetic. In fact, the dismal product of 1 and  $x$  causes all non-zero digits in  $x$  to be replaced by 1s! For example (writing  $*$  for multiplication)

$$9876543210 * 1 = 1111111110$$

It turns out that the dismal unit is 9! Moreover it's easy to describe the dismal primes <100; they are the eighteen two-digit numbers containing the digit 9:

$$19 \ 29 \ 39 \ 49 \ 59 \ 69 \ 79 \ 89 \ 90 \ 91 \ 92 \ 93 \ 94 \ 95 \ 96 \ 97 \ 98 \ 99$$

(Extra credit for explaining why the dismal unit is not considered prime.) Beyond this the dismal primes get more complicated, as is a more general dismal numbral theory. Extra extra credit: describe a model for the dismal numbrals—are they some kind of strange infinitesimals? Note that if we reduce the base from 10 to 2 then the resulting arithmetic is a more typical Boolean one equivalent to bitwise OR for addition and bitwise AND for multiplication (see the Poppseed question for more on this).

Two's Compliment. We can skin this cat multiple ways. First mechanically: call the unknown  $x$  and sum thus:

$$\begin{array}{r} \dots 0101010101 = x \\ + \dots 1010101010 = 2x \\ \hline \dots 1111111111 = 3x = -1 \end{array}$$

and so

$$x = -1/3$$

Another way is to "formally" evaluate the infinite sums represented by the binary strings. For example,

$$\dots 1111111111 = 2^0 + 2^1 + 2^2 + 2^3 + \dots$$

One might hesitate at this stage, thinking that this sum would be infinite. But, recalling the rule that the sum of an infinite geometric series with term ratio  $r$  is given by

# — COMPUTIST QUIZ ANSWER GUIDE —

$$r^0 + r^1 + r^2 + r^3 + \dots = 1/(1-r)$$

we just boldly substitute  $r=2$  to get  $-1$  (as expected) for the all-1s string and  $r=4$  to get (as we hoped)  $-1/3$  for the alternating string, thus confirming the mechanical manipulations algebraically. Playing fast and loose like this with infinite objects may seem strange, but it is actually quite vanilla *2-adic arithmetic* (Ramanujan is hinted to perhaps also have abused infinite sums in this Eulerian spirit—at least until Hardy beat it out of him).

Poppyseed Place. Which pair doesn't play well with the others, which pair isn't the same? We'd vote plus and minus off the island. The other pairs are all variants of infimum and supremum. Of course min and max apply directly to absolute values while the Boolean AND and OR operators essentially take the min or max in bitwise fashion. And gcd and lcm can be thought of as the application of min or max to the vectors of exponents in prime factorizations. (Of course, you can get extra credit for a really good justification of a different answer!)

Guilt by Dissociation. Some operations which always commute but aren't associative include:

$$|a-b| \quad -a-b \quad !a\&!b \quad 2a+2b \quad (a\&b)+1$$

although there are plenty of other interesting examples (give some for extra credit!). There seems to be an unfortunate tendency of late to assume (especially among computists enamored of category theory) that associativity is somehow necessarily prior to commutativity, implying some kind of “natural order” of organization for computational taxonomy. However useful such ideas may be in current practice, a lesson from the last century or two of mathematics is that almost any property may be independently varied—so the ultimate natural computational model must be a free composition of multiply inherited attributes, and not a hierarchy.

Plugh & Plover. We might say that *clever* is “elegant and unexpected”, while *smart* is “elegant and apt”. That is, a smart solution is not only clever, but also a solution to the *right* problem—addressing its “meta problem” as well as the problem.

Nullicorn. An increasing challenge for software that augments human enterprise will be computing effectively about epistemologically nontrivial domains—beyond what can be modeled with prosaic “objectivity” (just as quantum mechanics hints that the classical object-oriented paradigm at best only approximately models reality). While unicorn horn counting may seem facetious, asking how many customers for a not-yet-invented product exist (ie can dance on the head of a pin) may not be so silly a query after all!

Curiously, centuries ago, the so-called Medieval Logicians had some useful ideas along these lines. William of Ockham with his razor is a well-known member of this group, along with Peter of Spain (who may, or may not, have also been Pope John XXI—despite hundreds of copies of his textbook we're not sure—so much for the persistence of institutional knowledge!) and the wonderfully-named Garland the Computist (whose appellation I've swiped as a synonym for hacker). Apparently much of this thought also wasn't known to Russell, Gödel and other early modern logicians, but was only “recovered” by scholars in the 1960s. In fact, there are several logical disciplines mandated in the medieval curriculum whose procedures we know in detail, but whose motivations currently remain quite mysterious (see <http://plato.stanford.edu/entries/obligationes/> for example).

## — **COMPUTIST QUIZ ANSWER GUIDE** —

*Terasecond.* One *million* seconds is something over eleven days, one *billion* is over three decades (think of that next time someone is bandying about budgetary “illions”!) while one *trillion* seconds back plops us smack dab in the middle of the Paleolithic.

Our best bet might be to organize the temporal locals somehow, into a social enterprise whose goal would be monumental computing, instead of, say, pyramids. Of course our available talent pool of homies and hackers would be beta test *Homo* “early adopters”, perhaps with some legacy Neanderthals. And a contemporary example of a recently introduced disruptive killer product would be flaked “spearing point” technology...

And then, “suddenly”, there was an explosion of innovation: cave art, religious funerary practices, culture...

*Nanograph.*

### The Evangelist

The People have huddled around campfires in their caves for generations when the naked stranger appears in a violet flash. Some attack, but Gort intervenes. The weakling is obviously insane—gibbering at them, scratching burnt sticks on the wall—before being slapped unconscious. Into the night, Gort scowls puzzledly at the sooty drawings...

(I recommend Andrew Looney’s <http://www.wunderland.com/WTS/Andy/Nanofiction.html> especially “The Relentless Follower”)

Thanks to: David Applegate, Tom Duff, Bill Gosper, Mike Huskins, Shel Kaphan, Don Knuth, Howard Landman, Bill Schottstaedt, Neil Sloane & Henry Warren for their helpful and interesting commentary, inspiration and guidance. Further contributions welcome!