C13: (from Robert Gordon, Naval Undersea Systems Center) IN THE NEXT DECADE, GENERAL PURPOSE SOFTWARE WILL BECOME AS IMPORTANT AS GENERAL PURPOSE HARDWARE.

C14: (from Robert Gordon) MODULAR HARDWARE, FIRMWARE, AND SOFTWARE WILL ALLOW EACH USER TO BE HIS OWN "DO IT YOURSELF" SYSTEM ARCHITECT. THE ONLY NEED FOR PROFESSIONAL COMPUTER ARCHITECTS WILL BE FOR THE REALLY "LARGE" INFORMATION PROCESSING SYSTEMS.

OTHER CONTRIBUTIONS

How to Make Commercial Computers Catch up with Technology

Andy Tanenbaum (Vrije Universiteit, The Netherlands) offers the following "modest proposal":

> The architecture of most commercially available computers is years behind the state of the art (i.e. what the existing technology will support). The biggest single contribution to reducing this gap would be to have Congress pass a law legally prohibiting any computer manufacturer from selling, renting, leasing, distributing, disseminating or giving away any software of any kind. This should specifically include operating systems and especially microprograms. Thus, for example, IBM could continue selling its 3125 microprocessors, but would be forbidden from equipping each one with an interpreter for 370/125 "machine language".
>
> This proposal would have the following consequence. Independent software houses would compete fiercely to produce the best microprograms and operating systems. The investment in producing a microprogram is small, but the potential market is enormous. Computer customers could then pick and choose among the offerings.
>
> Most important, many of these software houses would come to the conclusion that what the customers really want are microprograms for APL, COBOL, FORTRAN, PASCAL, PL/I, RPG etc. machines, and not conventional "machine language" machines. Admittedly the IBM 370 microprocessors and their imitators are not as suited for direct interpretation of a high level machine as say a Burroughs B1700, but they are a great deal more flexible than is generally realized. A 3125, for example, is a kind of overgrown PDP-8, with 13 instructions.
>
> High level machines for Euler (Weber, CACM, Sept. 1967, p. 549) and APL (Hassitt et. al, CACM, April 1973, p. 199) already exist on the 360, so the idea is clearly feasible. In this highly competitive market any ninny who tried to sell an interpreter for a (virtual) machine with a bunch of general registers, etc. would quickly be told where to GOTO.
>
> Interestingly enough, there is historical precedent for such a "Mandatory Unbundling Law". In the early days of photography, Kodak used to sell film with the cost of developing included in the price. When Kodak was the only company with any technical expertise in the then new technology of film developing, no one minded. Eventually the knowledge of how to develop film became widely enough distributed that any 10 year old kid with a bath tub could do it, and Kodak was forced to "unbundle" developing, to allow independents to compete. The time is now ripe for 10 year old kids with KSR-33 teletypes.

Are Present-Day Floating-Point Number Representations Lunatic?

J. J. Horning (Univ. of Toronto) and some of his colleagues think so, and they tried to illustrate the proposition with following "Real-Quiz." What do you think?

(Questions submitted by Holt, Horning, Lipson, McKeeman, and Sevcik, April 1972)

1. T/F One-third is an exact number.

2. T/F One-third can be represented exactly in a computer.

3. T/F There are 3,793,096,385 interesting single precision floating point numbers in the IBM System/360.

4. T/F We use floating point because that is what the engineers gave us.

5. T/F Computers can calculate only rational values.

6. T/F Quick answers are more important than correct answers.

7. T/F Small numbers are less accurate than large numbers.

8. T/F The numbers 1.0 and 49,268,314.25 are equally interesting.

9. T/F The numbers 1.0 and 49,268,314.25 are equally probable.

10. T/F Only God can compute with the real numbers.

11. T/F Double precision is twice as wrong.

12. T/F Because computation is expensive, an answer should always be given, regardless of accumulated error.

13. T/F Infinity cannot be represented in a computer.

14. T/F Interesting numbers are easy to write.

15. T/F $0/0 = 1/0$

16. T/F $(X + X)/2 = X$.

17. T/F $X + |Y| \geq X$.

18. T/F Dividing by 16 is a more precise operation than dividing by 3.

19. T/F Fixed-length binary numbers are natural.

20. T/F The smallest number is the reciprocal of the largest number.

21. T/F A loose error bound is better than a close error estimate.

22. T/F You cannot tell how accurate a number is by looking at it.

23. T/F Input data should be represented by as many bits as possible.

24. T/F You are not a numerical analyst until you do an error analysis.

25. T/F A small backward error bound indicates a well-posed problem.

26. T/F Numerical analysts cannot devise a better approximation to real arithmeti

27. T/F Variable precision is too hard to use.

28. Desk calculators are:
a) Our best model of electronic computers.
b) The ideal computational device.
c) The basis for all error analysis.
d) The worst thing that ever happened to numerical analysis.

29. 10**47 is

a) 10^{47}

b) $(10\ (1 + \epsilon))^{47}$.

c) $10^{47}\ (1 + \epsilon)$.

d) impossible to represent precisely in 64 bits.

30. If the error is hard to bound
a) You haven't read Wilkinson.
b) You've missed a "trick."
c) The problem is ill-posed.
d) There is something wrong with the computational system.

31. Most error analysis is
a) Backward.
b) Of little use to the user.
c) Capable of being automated.
d) Solving the wrong problem.
e) All of the above.

32. T/F The accuracy of a computation is independent of the accuracy of its inputs

33. T/F Double precision solves almost all problems of accuracy.

34. T/F The user knows what precision to request.

35. T/F Normalization increases the accuracy of floating point operations.

Which of the above questions are

a) Meaningless?
b) Meaningful but impossible to answer without further study?
c) Trivial?
d) Rhetorical?
e) Ambiguous?
f) All of the above?

Seperating Jump Address Computations From Jumps

Dave Parnas (T.H. Darmstadt) offers the following:

A machine jump or transfer instruction which uses indirect addressing and/or indexing is executed in two distinguishable stages:

1) computation of the destination address and
2) transfer or control

Observation of programs indicates that:

1) the destination could often be computed well in advance of the jump
2) often the same destination is explicitly recomputed many times

These observations suggest that performance could be improved by having two instructions: "compute destination" and "jump to previously computed destination." This would be especially feasible on machines with several instruction lengths. It would be expecially useful with pipelining. With cooperative software, the "guess which branch" circuitry can be eliminated.

Please try a turn at Conjecture Corner.

- -

CAN CONTRIBUTIONS

An Annotated Bibliography on Stacks

by

R. M. Glorioso, C. Chung, W. Der, J. Estep,

A. Kaplan, G. Koizumi, W. Schweber

-

University of Massachusetts