MAY

HELLO PEOPLE.

Another issue of CLOAD Magazine has managed to wend its path from concept to reality. We think you'll like this one, were back to a C-30 cassette, and we have got more programs, so we had to cut short on the audio portion. We're also putting a label on both sides starting this month, and with our luck, we'll probably put them on backwards. Bear with us a tad longer, while we attempt to get our act together. We've recovered from the April fiasco with only minor damage, and are back on the track towards our goal of "first of the month".

Last month, we had some data on the tape which caused a bit of confusion to some of our subscribers, in that it would load, but it would not run, nor would it list. That's right folks, it was not a program. For those of you who are not familiar with the technique, the black users manual has an appendix on saving data on cassette tape. We had data "block" prepared for this month's graphing program, but we decided not to put it in, because our mass duplicators had a hard time with April's data block. Regret the decision, as it was a beautiful drawing.

Two problems have come up with Level I/Level II conversion, and I'd like to talk a bit on them. The first is that in Level I, the "A" array does not to be "demensioned". In Level II it does. In the future we'll put a "REM" statement in the first few lines of each program, explaining the necessary change. The other problem involves the way that we decide whether a user has answered "yes" or "no", (or sand, gravel, brick, etc.). Level II does this differnetly, and to be upward compatible, in the future we'll be publishing programs which use a different technique. Example: Do you want to play again (1=yes, 2=no)? Again for those concerned future programs will be readable in Level II.

On to other things, this last weekend, yours truly was involved in two programs which illustrate the ability to use the TRS-80 for serious applications. The first was a program which selected various combinations of gears to cut a particular helix on a lathe. The programming technique was to pick a given combination of gears and lever settings, compute the pitch of helix that would be cut if those settings were used, and comparing that to the desired pitch. If they were in a certain tolerance, the results were put on the screen. Then another combination was selected and the process was repeated. The program continued until all combinations were selected, and the "best" combination was selected from those displayed.

The second application was similar, it involved converting an "english" lathe to cut metric threads in it, the leadscrew speed was altered through gear selection , and the entire existing thread chart was recomputed to show the affect of the change. Certain ratios yeilded many correct metric settings, others did not. The desirable charts were selected by inspection. It turned out that the range of interest could be acheived with about nine new gears. The overall savings, amounted to several times the cost of the computer. Those who have attemped to do this "by hand" might find it interesting that the entire process (including writing the program) took about four hours and was a lot of fun.

My last topic this month is to start a continuing series on "what's going on in there"- insight on the inside of a TRS-80 as it were. At first , I'll be explaining various "buzzwords" and then putting them into perspective with the design of the TRS-80. By the time the series is finished, the computer will be obsolete and we can all start over again.

This month's buzzword is "byte" we start by explaining bits. "Bit" is a contraction for binary digit. The bianary number system, based on 0 and 1, is easy for a machine to work with, as it only has to recognize two "states" (on or off, high volume or low volume, etc). The trouble is, a one bianary digit number doesn't carry much information. As we humans think of it, just two patterns, 0 and 1, so we stack them together, and make the computer work with many at once. If a four bianary digit number is used as our basic chunk of information, we have sixteen possible patterns from 0000 to llll which can stand for sixteen different people, sixteen different people, sixteen different recipes, sixtten different commands, etc. Such a pattern width is called a "four bit word", a "hex digit", a "half byte" or a "nibble" (Honest). That's better than two patterns, and allows us to juggle greater concepts than "democratic" or "republican", but humans aspire to unimaginable heights. If we choose an eight binary digit number, we have 256 distinctive patterns from 00000000 to llllllll. This can stand for 256 different recipes, etc. This wide a chunk is called a "byte". Remember: A

nibble is a hex digit is a four bit word, but a byte is a byte. Nobody knows why it is called that, but nobody calls it anything different. So what ? A sixteen binary digit chunk has 65,536 patterns possible. Why not use it? Because for most things it's a bit to large and unwieldy. How about a twelve bit chunk? Well, the good ole PDP-8 (a very nice computer from Digital Equipment Corp.) used a 12-bit chunk, and it worked just fine. However, fashions and popularity and all that work in the computer field as anywhere else, and a general tradeoff has occured with the "byte" chunk the winner. So people tend to talk of bytes. If they need 16 bits, they use two bytes. There talking about 64 bits , they speak of 8 bytes instead. Remember we said a byte could store up to 256 different patterns? Well, if we ignore about 200, or so, we can let each remaining pattern stand for a letter of the alphabet, or number or punctuation mark.

And that folks, is why your program is so many "bytes" long. We use one byte of storage for every letter, number, or punctuation mark, when we store it in "user memory space" (whatever that is)

Cogito, Ergo Hackum...

RD711 cElroy

RD McElroy
Publisher