Here's February!

This issue is a bit of a special one for us - it is our 24th issue, which rounds out our second year. We here at CLOAD hope that all our faithful subscribers have had many happy loads these past years. Here's what we've got lined up for February:

CLOAD
MAGAZINE, inc
BOX 1267
GOLETA, CA
93017
FEBRUARY 1980

Level	Title	Turns Count CTR-41	Turns Count CTR-80
IIIIIIIII	Zap/Zonk (cover)	17 & 263	10 & 155
II	Kalah Instructions	59 & 294	35 & 173
II	Kalah	117 & 338	69 & 198
II	Dissertation	171 & 379	100 & 224
IIIIIIIII	Coefficients	216 & 416	126 & 245
IIIIIIIII	Zap/Zonk (cover)	10 & 36	6 & 21
II II	Election	61 & 177	37 & 104
II II	Kalah	278 & 319	164 & 188
II II	Monitor	359 & 383	211 & 226
IIIIIIIII	Dissertation	406 & 427	238 & 251

A lot of you will recognize "Kalah". It is a centuries - old game that originated somewhere in Africa. It can be played with a wide variety of equipment: with seashells in the sand (cost - nothing), with beans in a carved wooden board (cost - the effort of carving the board), with plastic tokens in an injection molded plastic tray (cost - $6.98 plus tax), or with bits in a computer (cost - $500 up). Which equipment you use all depends on your particular society's level of technology.

That's really not fair. When you play it with bits in a computer, you don't need a human opponent. The computer will always be willing to play, and you can thus match your skills against the kind of machine which will probably replace you in a few years. Good luck!

"Dissertation" is a program of the highest versatility and practicality. As we move further and further into a report-based society, it is becoming imperative that we utilize the newer levels of technology to aid us (this, of course, is nothing new - human society has long been noted for its ability to build machines to take over the drudge work). This program has been optimized to generate a dissertation, but it is certainly not limited to this task. With slight modifications, it can generate contract proposals, political speeches, medical and legal briefs and so on. As the use and understanding of computers increases in the years ahead, more and more communications will be handled with a direct link from computer to computer. At that time, a computer will be used to read the incoming messages. Humanity will then have successfully set up a society of computers that generate, disseminate, and digest data in much the same way and with much the same results as the current human/paper/human system. Then the humans can hand over the keys to the executive washroom and start all over again.

There are party poopers who feel that this is absurd. I can only reply that CLOAD magazine has a program much like this one that we have been using for years to generate text for these "yellow pages".

"Coefficiens" is a math problem solver. In the field of mathematics, there is a type of problem where you are given a set of "n" equations (hopefully nice, neat types) each having "n" unknown terms. A simple example would be:

$$3X + 5Y = 11 \quad \text{and} \quad 5X + 7Y = 17$$

Here we have two separate equations and two unknowns (X & Y) in each. It turns out that (1) if there are as many equations as there are unknowns and (2) if no equation is a direct multiple of another, then there is a unique solution to the problem. A solution is considered to be a list of the values for all the unknowns (in this example, X=2 and Y=1).

"Election" is a rather current topic, this being a year with an upcoming U.S. Presidential election. With this game you can simulate the results that would accompany your entry into the race. Your opponents are those who looked good at the time this program was written. Thus in the Democratic Party's semifinals Senator Kennedy is given a strong position over President Carter, and the main Republican contender is Ronald Reagan. Who can tell, by the time you run this program, Samuel Wilburforce might have captured both nominations.

When I ran this one, my campaign philosophy (and its results) were what prompted this statement by then-Senator John F. Kennedy:

"One Senator, since retired, said that he voted with the special interests on every issue, hoping that by election time all of them added together would constitute nearly a majority that would remember him favorably, while the other members of the public would never know about - much less remember - his vote against their welfare. It is reassuring to know that this seemingly unbeatable formula did not work in his case."

Perhaps your technique will be more successful. If so, you might consider a career in public life, armed with the "Dissertation" program modified to generate speeches.

"Monitor" is a program similar to the Radio Shack T-BUG monitor. It is written in BASIC, so you can see what is actually being done. Your capabilities with this program include converting between base ten and base sixteen (hexadecimal, or hex for short), altering memory, examining memory either as hex numbers or ASCII characters, loading and saving on cassette tape, executing any system code you may have written (through the SYSTEM command), and the usual return to BASIC. If you accidently return to BASIC, all is not lost and suicide need not be contemplated. Simply RUNning the program will set you back up.

Since I know that at least all you level IIer's have a monitor now, I'd like to talk a bit on the subject of machine language subroutines. Of necessity, the language will be laced with jive machine talk. My apologies to those who would prefer English.

Machine language has a "GOSUB" command which acts exactly like its BASIC counterpart. The assembler mnemonic is CALL and the instruction byte (which is what the computer sees) is CD in hex notation. We'll be using hex for the rest of the article. When the computer is tripping merrily along executing a machine language program, and arrives at a CD in memory, it goes into a

little dance to call a machine language subroutine. Here's a step by step description of what happens:

(1) The Z-80 (computer CPU chip) reads the CD we've talked about and puts into its internal "command" buffer.

(2) After pondering a microsecond or so, the Z-80 decides that it should do its "call a subroutine" act.

(3) The first step of this act is to grab the next two bytes of memory and use it as the address of the subroutine to call.

(4) After it has the address to find the subroutine, it has to write itself a note telling it where to come back. Since it wants to come back to the point just after the CALL sequence, it puts the address which immediately follows the CALL sequence on a little piece of paper and puts it on top of a stack of little pieces of paper that it has written itself in the past. This stack, incidentally, is referred to in computer jargon as "the stack".

(5) Thus having arranged its affairs for the return trip, it takes a deep breath and JMPs to the subroutine. The Z-80 can leap from any part of memory to any other part in a single bound. This is called "non-partitioned memory", a high gloss phrase that you can use to impress your friends.

(6) The Z-80 proceeds to execute the subroutine. There is no restriction on how many "nested" subroutines can exist. This means that a subroutine is free to call another subroutine, which in turn is free to call yet a third, and so on ad nauseum. Some (strange) programmers write subroutines that call themselves as subsubroutines hundreds of times. There is a sorting algorithm lled a "quicksort" that makes heavy use of this ability.

(7) The subroutine (let's keep it simple at one level) is finished when the Z-80 encounters a return instruction. The usual one used is the simple RET, which has an instruction byte of C9. When the Z-80 encounters a C9, it picks up the piece of paper it left on the stack and reads the address written on it. If everything went well, the piece of paper is the one it left there before and the return address is the correct one.

(8) The Z-80 places the piece of paper on the scrap heap and JMPs back to the place it started from.

Let's look at an example - assume that a section of memory is set up as follows:

Main routine		Subroutine	
Address	Data	Address	Data
5000	00	6034	3C
5001	00	6035	3C
5002	00	6036	05
5003	CD	6037	C9
5004	34	6038	2F
5005	60	6039	07
5006	37		

Assume that the Z-80 is executing the instruction at address 5000 when we first look in. The Z-80 fetches the byte out of RAM at that address and executes it (that is, it "does" it, not electrocute it). The hex command (called an operation code, or opcode) is a 00. This is the infamous NOP, or

no operation. It does absolutely nothing (though it is often handy). After doing nothing for about 2.25 microseconds, the Z-80 goes out and fetches the next byte, at address 5001 (sounds like a movie, doesn't it?). Same story, though. Eventually we get to the point where the Z-80 fetches the byte at 5003. It is a CD, or CALL opcode (step 1 above).

The Z-80 now knows that a subroutine call is coming up (step 2), but it does not know where. It does know, however, that the programmer was told to put the address in the next two bytes (in the usual insane reverse order). Fetching these (step 3) and re-reversing them to rational order takes the Z-80 to address 5006. It does not fetch the opcode at 5006 yet. Instead, it writes the bytes 06 and 50 (reverse order, of course - don't ask why, only the Z-80 designers know) into a particular area of memory (the stack). It now jumps to the byte at address 6034 (step 5).

We now resume a more normal mode of operation (at step 6). The Z-80 fetches the byte at address 6034. It is a 3C, which is slightly more interesting than the 00 opcode. It is the INC A instruction. It takes whatever is in the "accumulator" register and adds the number one to it, placing the result back in the accumulator. What was in the accumulator? Whatever was in it when this subroutine was called. The next byte (at address 6035) does it again. The byte at 6036 is an 05. That's a DEC B, or subtract-one-from-register-B instruction. Ho Hum.

Aha! The next byte fetched is a C9, or RETurn instruction (which takes us to step 7). The Z-80 can do a RETurn without any additional data, because it knows that the return address is on the stack (which is where it had better be). It fetches two bytes from the stack (reversed...), readjusts the stack and leaps back to address 5006. If this all seems vaguely similar to the GOSUB/RETURN commands in BASIC, that's because it is.

For discussion on what the stack is and how it is used, as well as a special type of CALL that is executed with an electrical pulse rather than a memory instruction, tune in next month. For now, enjoy your programs!

Ralph McElroy - Publisher