Best of INTERFACE AGE™
VOLUME 2
GENERAL PURPOSE SOFTWARE

edited by the interface age staff

cover illustrated by Fino Ortiz

Best of Interface Age

Volume 2: General Purpose Software

Best of Interface Age

Volume 2: General Purpose Software

Edited by
Interface Age Staff

dilithium Press
Portland, Oregon

10 9 8 7 6 5 4 3 2 1

ISBN: 0-918398-37-1
Library of Congress catalog card number: 79-67462

Printed in the United States of America.

dilithium Press
P.O. Box 92
Forest Grove, Oregon 97116

Preface

Volume Number 2 of the five volume *Best of Interface Age* series is significant since it presents thirteen of the most-asked-for system and application software articles printed in *Interface Age.*

The articles that are contained within this volume were chosen not only for their value as working software systems, but also for their value in showing a number of different programming techniques. We at *Interface Age* firmly believe that serious students of software, and those that just enjoy making use of software will find this book invaluable.

Table of Contents

Chapter 1

Inside ASCII

by R. W. Bemer

The data alphabet called ASCII (Figure 1 and Reference 1), also has two other names—International Standard 646 (the ISO Code [Reference 2]) and Alphabet No. 5 of CCITT (the International Consultative Committee for Telephone and Telegraph). It is used throughout the world, incorporated in billions of dollars of equipment.

But is it used correctly and wisely? Not always. There are misinterpretations, and gaps in definition that permit nonstandard usage. This article will give you the background, peculiarities, preferred practices, and new developments for ASCII. You will find a lot of information not too generally known or realized; it should help in the correct and safe usage of ASCII. For additional help, you can reference the various national and international standards given in Table 1a. Some other detailed articles are listed in References 3, 4 and 5.

STICKS 4-7

ASCII, as a 7-bit code, is usually represented in 8 columns of 16 positions. The row positions are 0000 through 1111, the low-order 4 bits, 0 through 15 in decimal. The columns are 000 through 111, the next higher 3 bits, 0 through 7 in decimal. For some reason, the developers of ASCII found it convenient to refer to these eight columns as "sticks." So shall we. Each position will be represented in this article by its usual decimal representation. For example, capital A is position 4/1. Figure 2 is a representation of ASCII that is more convenient to those working in octal, rather than hexadecimal, notation.

The first positions of sticks 4 and 6 are respectively the "commercial at" and "accent grave." Then the upper and lower case Roman alphabets follow. This offset of one position is historical (from the United Kingdom), and of no importance as long as you remember that it is so.

Following the alphabet in both sticks 5 and 7 are three positions each that one must be very cautious about. In ASCII they are assigned as [, /[1], and] in stick 5—{, |[2], and } in stick 7. But in the ISO Code and CCITT versions they are reserved for national usage. Table 2 gives the

$b_8 b_7 b_6 b_5$ →		0000	0001	0010	0011	0100	0101	0110	0111
$b_4 b_3 b_2 b_1$	ROW \ COL	0	1	2	3	4	5	6	7
0000	0	NUL □	DLE ⊟	SP △	0	NOTE 1 @	P	NOTE 1 `	p
0001	1	SOH ┌	DC1 ◔	!	1	A	Q	a	q
0010	2	STX ⊥	DC2 ◑	"	2	B	R	b	r
0011	3	ETX ┘	DC3 ◕	NOTE 1 £/#	3	C	S	c	s
0100	4	EOT ↯	DC4 ◷	NOTE 1 $	4	D	T	d	t
0101	5	ENQ ⊠	NAK ✗	%	5	E	U	e	u
0110	6	ACK ✓	SYN ⎍	&	6	F	V	f	v
0111	7	BEL ⌂	ETB ⊣	'	7	G	W	g	w
1000	8	BS ↖	CAN ⧖	(	8	H	X	h	x
1001	9	HT →	EM ⧫	)	9	I	Y	i	y
1010	10	LF ≡	SUB ʔ	*	:	J	Z	j	z
1011	11	VT ↓	ESC ⊖	+	;	K	NOTE 1 [	k	NOTE 1 {
1100	12	FF ↡	FS ◳	,	<	L	NOTE 1 \	l	NOTE 1 ¦
1101	13	CR ←	GS ◱	–	=	M	NOTE 1]	m	NOTE 1 }
1110	14	SO ⊗	RS ◲	.	>	N	NOTE 1 ^	n	~ NOTE 1
1111	15	SI ⊙	US ◰	/	?	O	—	o	DEL ▨

Note 1
These 12 positions are variable for national usage -- 2 for currency, 7 primary national usage, and 3 secondary usage which are diacritical marks when preceded by BSP. The presently known assignments are given in the table below.

Figure 1. ASCII (ISO Code)

national use assignment for these positions. Surely you remember that the Scandinavian alphabet has 29 letters, not 26? My friend Ørjar Heen in Oslo is very protective of these positions. He says "If you Americans want to sell computers and software abroad, don't use the ASCII characters for these positions in your software."

To be more precise, positions 5/11, 5/12, 5/13, 7/11, 7/12, and 7/13 (noted above) are called *primary* national usage positions. So is 4/0, where ASCII has the "commercial at." Honeywell, for example, uses

	ISO	ECMA	ANSI	FIPS PUB	CSA	BS	AS	CCITT	JIS	GOST
Binary-coded Character Set	646	6	X3.4-1977 $4.50	1	Z243.4	4730	1776	V.3	C6220	13052-67
Graphics for Control Characters	2047	17	X3.32-1973 $3.50	36		4730				
Character Set for Handprinting	97/3 N119		X3.45-1974 $5.75	33	Z243.34.1					
Additional Controls Character Imaging		48	BSR X3.64							
4-bit Sets	963	14		15	Z243.6	4731/1	1070			
Code Extension Techniques	2022	35	X3.41-1974 $6.00	35	Z243.35	4953				
Registration Procedures for Escape Sequences	2375									
8-bit Coded Character Set	DIS 4873	43	X3L2/77/08							
Character Set for 7 x 9 Matrix Printers		42								
Keyboard	2530	23	X4.14-1971 $3.75			4822/1	1922			
Character Sets for Programming Languages	97/5 N436	53								

Legend

ISO	- International Standards Organization
ECMA	- European Computer Manufacturers Association
ANSI	- American National Standards Institute
FIPS	- Federal Information Processing Standard
CSA	- Canadian Standards Association
BS	- British Standard
AS	- Australian Standard
CCITT	- Consultative Committee International, Telephone & Telegraph
JIS	- Japanese Industrial Standard
GOST	- USSR Standard

Table 1a. Logical Standards for ASCII

the "at" in a timesharing system for deleting the previous character upon entry. But this isn't too serious, because many nations also have the "at" in their primary sets.

Also in sticks 4-7 are three diacritical marks. They are accent grave (`) in 6/0, circumflex (^) in 5/14, and tilde (~) in 7/14. These are called *secondary* national usage positions. In some countries the tilde is a straight overline.

But it is the circumflex where we have a lot of confusion. Teletype first made it an "up arrow" in an earlier version of ASCII, to serve as an exponentiation symbol, primarily for BASIC. But that doesn't do very well, because the exponentiation for FORTRAN is a double asterisk! The FORTRAN version is preferable in France, certainly, because they use such words as crâne, côte, coût, and so on.

A companion problem exists in position 5/15, with the underscore. The underscore is neither national nor diacritical; all countries use it just as underscore (and for typesetting it is a U.S. convention to indicate italics, but in Italy it means boldface, except when it is the last character in a line![3]). But Teletype's early version of ASCII used it as a

HIGH ORDER OCTAL DIGITS →	00	02	04	06	10	12	14	16	LOW ORDER OCTAL DIGIT
	NUL	DLE	SP	0	@	P	`	p	0
	SOH	DC1	!	1	A	Q	a	q	1
	STX	DC2	"	2	B	R	b	r	2
	ETX	DC3	#	3	C	S	c	s	3
	EOT	DC4	$	4	D	T	d	t	4
	ENQ	NAK	%	5	E	U	e	u	5
	ACK	SYN	&	6	F	V	f	v	6
	BEL	ETB	′	7	G	W	g	w	7

HIGH ORDER OCTAL DIGITS →	01	03	05	07	11	13	15	17	LOW ORDER OCTAL DIGIT
	BS	CAN	(	8	H	X	h	x	0
	HT	EM	)	9	I	Y	i	y	1
	LF	SUB	*	:	J	Z	j	z	2
	VT	ESC	+	;	K	[	k	{	3
	FF	FS	,	<	L	\	l	¦	4
	CR	GS	-	=	M	]	m	}	5
	SO	RS	.	>	N	^	n	~	6
	SI	US	/	?	O	___	o	DEL	7

Figure 2. ASCII in Octal Reference Form

"left arrow"—probably for an assignment symbol equivalent to := in ALGOL. The up and left arrow have been carried over from Teletype into many video terminals. Ask your terminal manufacturer to cease and desist and retrofit. It's not ASCII and will only cause trouble forever.

The last character in sticks 4-7 is the Delete, symbol DEL, in position 7/17. It was put here because the binary code is 1111111, which would be all punched holes in perforated (not always paper!) tape, and that is the only way to make sure that it cannot be misread as some other character. ASCII is a complete set; all positions are assigned to have meaning.

STICKS 2-3

These are usually called the sticks for digits and specials. Remember that they are the "digits" 0 to 9; not numbers, not numerals, not anything but digits! They are in 3/0 through 3/9 so that the low-order 4 bits are the representations for packed decimal. Originally we considered the possibility of a special 4-bit set for numerical applications (see the fifth entry in Table 1a), but it turned out that computer hardware became inexpensive enough to not deprive ourselves of the extra capabilities of the 7-bit and 8-bit sets.

Position 2/0 is officially called "space." I don't and didn't like it, and would have preferred "blank." Which is why the IBM community often uses a lower case "bee" with a slash through the vertical as its symbol. From the Univac side, the space has the official symbol "delta."

Having mentioned packed decimal, where two digits go into each 8-bit group ("byte" to the American, "octet" to the French), a word of caution on the plus and minus signs—they are in stick 2, rather than stick 3 with the digits. But the low order 4 bits are distinct, and + should be used only as 1011, − only as 1101. I mention this because the nonstandard code EBCDIC permits multiple representations of + and − in packed decimal. And the ASCII representations are not even coincident with any of these, with obvious dangers!

Watch out for the "currency" positions, 2/3 and 2/4. They also have national variations. In ASCII they are customarily # and $, but there are some things to be remembered:

- # is not "number sign" for many countries, most of which use "No." or "Nr." for that purpose. And when it is "number," it must precede the digits, not follow.
- # closely resembled the "sharp sign" in music.
- # is "pound sign" only for the U.S., the only major country still not using the metric system. To the rest, it's kilograms. For now, it's best to use the abbreviation "lb." in the U.S., not the #. In any case, both must *follow* the numeral.
- To the British, a "pound" has the symbol "£", which is why that is the symbol in position 2/3 for the UK. They get very irked when # is called a "pound" sign, especially in software manuals.
- The "dollar" is peculiar to the U.S., Canada, and some others. There are also francs, marks, escudos, pesos, lire, etc., etc. Which is why the ISO code uses the universal currency symbol in position 2/4. It's a circle with outside spikes at 45, 135, 225, and 315 degrees (¤), called "scarab." Table 2 also shows these assignments for several countries.
- ECMA has provided a separate guideline for specifying international currencies. See the "Where to Get More Information" at the end of this article.

It's a tough problem, and will get worse when we get into expanded character sets for photocomposition and such. For now, all we can do is follow the ASCII standard, which says that # is a "number sign."

Only a few more peculiarities remain for sticks 2-3. An important one is in the double quote, position 2/2, and the single quote, position 2/7. That is, you may think it is a single quote, and even use it so, but it is really an "accent acute" for vowels. It slants from top right to bottom left, to complement "accent grave" in 6/0, which slants from top left to bottom right. Some terminal makers do not realize this pairing, and will have accent grave slanting correctly, but put accent acute as a single quote in the unstylized up and down method. My Terminet is

	currency		1st 7 national				dia	dia	1st 7 national			dia
	2/3	2/4	4/0	5/11	5/12	5/13	5/14	6/0	7/11	7/12	7/13	7/14
Netherlands—A			@	[	\	]	^	`	{	|	}	~
Australia	#		↓	↓	↓	↓	↓	↓	↓	¦	↓	~
Belgium—A	↓		↓	↓	↓	↓	↓	↓	↓	¦	↓	‾
W. Germany—A	↓	$	↓	↓	↓	↓	↓	↓	↓	¦	↓	‾
US	↓	↓	↓	↓	↓	↓	↓	↓	↓	¦	↓	~
Japan	↓	↓	↓	↓	¥	↓		↓	↓	¦	↓	‾
UK	£	↓	↓	↓	\	↓	↑	↓	↓	¦	↓	‾
Italy—A	#	¤	↓	↓	↓	↓	^	↓	↓	¦	↓	‾
Switzerland—A	↓	¤	↓	↓	↓	↓	^	↓	↓	¦	↓	~
France—A	↓	$		↓	↓	↓				¦		~
USSR	↓	¤		↓	V	↓	∧	`				‾
Netherlands—B					IJ					ij		
Belgium—B			à		ç				é	ij	è	‾
France—B	£	$	à	·	ç	§	^	·	é	ù	è	¨
Switzerland—B			à		ç				é	ù	è	¨
Italy—B	#	$	§	·	ç	é	^	ù	à	é	è	ì
Switzerland—C					ç	é		ù	à	é	è	î
Hungary	#	Ft	§	É	Ö	Ü	Á	ó	é	ö	ü	á
W. Germany—B	£	$	§	Ä	Ö	Ü	^	·	ä	ö	ü	β
Switzerland—D				Ä	Ö	Ü			ä	ö	ü	β
Sweden	#	¤	É	Ä	Ö	Å	Ü	é	ä	ö	å	ü
Finland				Ä	Ö	Å			ä	ö	å	
Denmark				Æ	Ø	Å			æ	ø	å	
Norway				Æ	Ø	Å			æ	ø	å	
Spain					Ñ					ñ		

Table 2. National Usage

one of those that is OK.

Don't forget that to the typesetter, in contrast to typewriters, both single and double quotes have two forms—opening and closing. In fact, the typesetter gets his double quotes by using two single quotes, of either form, because the quote uses very little space in variable space typesetting. Most terminals, either video or hardcopy, use constant spacing. So double and single quotes must be distinct for that reason.

The last variation is in position 2/6, the ampersand. There are many legitimate different designs for the ampersand. Neither ASCII nor the ISO Code prescribe any particular one. But this leads us to the next topic—how to represent the ASCII characters in handprinted form, so that they may be input to computer systems.

HANDPRINTING FOR STICKS 2-7

The classical confusion for many years was between the digit zero and the letter "oh," but there are other possibilities for confusion. American Standard X3.45 specifies the handwritten character shapes shown in Figure 3.

This clears up a longstanding problem. The communications types, and the armed services, used to put a slash through the zero; somehow the IBM users got to putting the slash through the letter "oh" instead, confusing the Scandinavians greatly. Now it's neither (which helps), just a 180-degree rotation of the letter Q. The earlier German Standard DIN 66 002 prescribed the cursive loop in the upper right, as some may have learned in penmanship courses. It now permits the ANSI form as well.

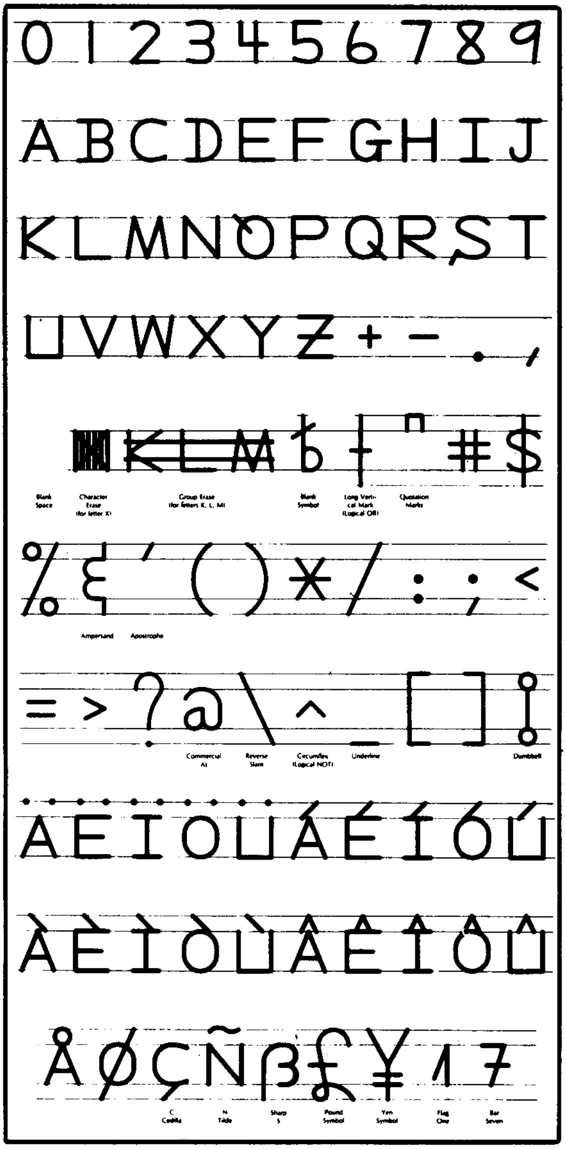


Figure 3. Handprinting for ASCII Characters

UPPER AND LOWER CASE LETTERS

Many people are accustomed to using upper case only. This is a hangover from early line printers and limited sets (until the Stretch computer of IBM, characters were usually 6 bits in size). It would have been far better if they had all been lower case in those smaller sets. Putting it simply, would you buy a book to read if it were all in upper case? Because lower case is much easier and faster to read, lower case should be the default case when one has only the one case. There is no reason why FORTRAN or BASIC processors cannot understand lower case variable names and verbs just as easily as they can understand upper case.

I always recommend getting a terminal with both cases if it is at all affordable. Second best is making sure that a single-case terminal is retrofittable later, if necessary. And if a single-case terminal, get it in lower case only, if possible. There has been much reportage in the computer trade press about eyestrain resulting from using computer terminals. Is the reason obvious?

STICKS 0, 1

These are the control characters. The most important distinction in ASCII is the split between sticks 0-1, Controls, and sticks 2-7, Graphics. We'll see this later on in the standards for Code Expansion (to 8 bits or more), and Code Extension (alternate sets, such as Cyrillic for the USSR, and Katakana for Japan).

Unfortunately, there is, despite the standard, much difference between the ways that various terminal devices handle these control characters. They may act differently, or they may not be operative at all. I have two very useful programs, written in the TEX language (Reference 6). One lists each symbol by name and then shows its action between parentheses. The other asks you to depress in turn all the funny keys on your terminal, and then tells you what control character(s) they generate, if any.

GRAPHICS FOR THE CONTROLS

There are standard graphical representations for the 32 controls, space, and delete. They are defined by ISO 2047, American Standard X3.32, and ECMA-17, and are shown integral to Figure 1. Some terminals are advertised as ASCII terminals, and yet generate Greek or other characters for these positions. Don't believe it! These symbols are every bit as useful as any Greek characters could be.

There are five groups in the basic control set.

STICKS 0, 1—Logical Communication Control (10)

This group is used for both communication and for labeling of media. It includes:

SOH (0/1)	(Start of Heading)—used as the first character in the heading of an information message.
STX (0/2)	(Start of Text)—terminates the heading just before the text.
ETX (0/3)	(End of Text)—Last character in the text message. Unfortunately, it is generated on many terminals via Control-C, and that's just to the right of Control-X on the keyboard,

which is commonly used to cancel a bad input line. And if you mis-key—ouch!

EOT (0/4) (End of Transmission)—the last character in any transmission, and usually it turns your device off!

ENQ (0/5) (Enquiry)—requests a response from a remote station, either an identification of that stations (Who are you?) or its status.

ACK (0/6) (Acknowledge)—used by a receiver to reply "yes" to a sender.

DLE (1/0) (Data Link Escape)—an Escape character, especially for communications, analogous to ESC (1/11). It signals the start of a character sequence that causes a shifting into another set of communication controls, whenever they are needed.

NAK (1/5) (Negative Acknowledge)—used by a receiver to reply "no" to a sender.

SYN (1/6) (Synchronous Idle)—needed by synchronous transmission systems to get into, or stay in, synchronization when no other such signal is available to them.

ETB (1/7) (End of Transmission Block)—indicates the end of some division of data that the transmission system must make, unrelated to any division in the format of the logical data itself.

STICKS 0, 1—Physical Communication (4)

This group is used for communications. It includes:

NUL (0/0) (Null)—the standard says that it is "used to accomplish media fill or time fill"..."may be inserted into or removed from a stream of data without affecting the information content of that stream." And that's exactly what the standard also says about DELete (7/15), which it lists as a control character even though it is not in the control sticks! The only difference I can see between them is that on perforated tape you can make any character into a DELete, but none into a Null.

CAN (1/8) (Cancel)—the receiver is to disregard the data received up to that point, starting from restart point that receiver and sender have agreed upon. It is common in timesharing for Cancel (often generated by a Control-X) to work on a line-at-a-time basis, to delete an unwanted string of entry characters, and effectively put one back to the position of reentering the entire line. In this case, the agreement between sender and receiver is "back to the last CR." But there are many other ways that Cancel could be used, and for parallel as well as serial transmission.

SUB (1/10) (Substitute)—a character that says probably we would have had another character in this position if we could have figured out what it was supposed to be! There are many reasons for such confusion—perhaps parity didn't check out. But it is better to put in a SUB to keep the field lengths and such correct. Moreover, note its symbol, a mirror image (not the Spanish inverted) question mark. If this is displayable, it will tell you definitively that the system doesn't know what it is, and you can make a good guess in many cases, particularly in word text.

EM (1/9) (End of Medium)—defines the previous character as the last usable character on that medium, whether or not there is more recordable space on the medium.

STICKS 0, 1—Device Control (11)

This group is used for control of devices such as terminals.

HT (0/9) (Horizontal Tabulation)—the standard says that is "advances" the active position to the next predetermined character position on the same line." There are two ways this can work:

1. Right at the terminal, if it has the horizontal tab capability built in. Sometimes you can set the tab positions by using the terminal only; almost always the computer can be made to set the tabs on the terminal. Then when you hit HT during entry, or HT is read from the computer output, the printing or displaying (active) position will skip to the next tab setting.
2. By a formatting program in the computer, which must be given some indication of the tab setting positions in force at any particular point in the file. The program then simulates horizontal tab movement by filling the lines with spaces as needed to achieve the alignment.

VT (0/11) (Vertical Tabulation)—the standard says that it "advances the active position to the same character position on the next predetermined line." And if you agree with somebody else, it can be to the first position in that line instead. This is a very dangerous character to use. It cannot be used directly on any terminal that I know of. Even if it could, the implementation rules are not supplied unambiguously in the ASCII standard. And for use by a formatting program, one would have to predefine the number of lines to be skipped. That's pretty tough when you are inserting and deleting lines, as every programmer knows.

LF (0/10) (Line Feed)—like vertical tab, but just to the next line, which is clean enough. If receiver and sender agree (again as in vertical tab), it can be to the first position of the next line, in which case it is called New Line (NL). Some manufacturers implement this. I personally prefer having a separate Carriage Return and Line Feed. Both codes can be generated with a single keystroke, and they often are.

FF (0/12) (Form Feed)—again like vertical tab, to the same character position unless sender and receiver agree that it is to the first position in the new line, except that the tab is to a new line position that is related to a form of some size (those that fold 11 inches apart, for example). This control could run wild if your terminal or other display device is not equipped to handle it, so use it with caution in files.

CR (0/13) (Carriage Return)—moves the active position to the first position on the *same* line! Not like typewriters. They have effectively incorporated the New Line feature. But the non-advancing CR is better for terminals, even if it is misnamed. Neither video terminals nor ball and daisy wheel typewriters have carriages, so live with it.

BS (0/8) (Backspace)—Backspace is a very tricky character. On some terminals, such as video terminals, there is no key to generate Backspace for entry into the text stream or buffer. On many it can be created via Control-H. Even then, it may or may not be operative.

Backspace is meant for physical movement of the active position (which may or may not coincide with a cursor position, when such exists). Historically, it was included for hardcopy terminals and other hardcopy devices for some of these uses:

- Underscoring (underlining).
- Other forms of highlighting, such as bold. For example, the sequence A BS A BS A would strike the A three times on a hardcopy device, and make it look boldface (such a sequence can also be translated to call a boldface font in photocomposition).
- Editing indications. For example, in legislative bill drafting to indicate the deleted or changed portion:
  T̸his̸ Is̸ o̸bs̸o̸le̸te̸.
- Forming composite characters, e.g.:
  ≚ ± ≠ ! ⊂⊃ ‖ } ₣(Hungarian forint)
- Forming accented letters, primarily for European languages. Examples:
  A Å Ø (Scandinavian letters following Z)
  N a a o u

Warning: Backspace is entirely different from a cursor movement on a video terminal! When the cursor is moved to a position where a character is already entered, succeeding entry in that position usually destroys the original character and replaces it with the new entry.

I personally haven't seen any video terminals with a true backspace. A former president of Infoton told me it could be done as an engineering special for about $5,000 one-time cost.

Warning: There are three ways to create underscored text for hardcopy terminals:

1. The characters, that many backspaces, and that many underscores (or vice versa).
2. A character, BS, underscore, the next character, etc. This is called the canonical form, and is used quite commonly.
3. Underscore, BS, character, underscore, etc.

I have noticed a lot of difficulty moving back and forth between hardcopy (at my home) and video (in my office) terminals. One tends to underscore on the hardcopy terminal and forget that half of the pairs are going to be wiped out by the cursor on the video terminal. In the first two methods above, it's the text that gets wiped out, and it's hard to read on the fly. So if you plan to display a file on a video terminal, find another highlighting method, or use the third underscoring convention. Even that may give problems if done by embedding an underscoring command in the file you pass to a formatting program; most such programs put the underscore last instead of first.

BEL (0/7) (Bell)—sounds an audible signal to get the user's attention. Some terminals are not so equipped, but they should be. It's good human engineering. But please give me an adjustable volume control!

And then there are the four device controls for unspecified purposes, DC1, DC2, DC3, and DC4—in positions 1/1 through 1/4. Different manufacturers treat these like a wild card in poker—they make them anything that they want. Doesn't lead to much compatibility, so beware.

STICKS 0, 1—Information Separators (4)

This group is used for formatting and string processing. These are the separators in positions 1/12 to 1/15. I got the idea originally from the Word Mark in the IBM 1401, which used an extra bit in the low-order character in a field as a delimiter. ASCII uses special and separate characters to indicate a hierarchical structure. Originally I put in eight such characters, but only these four remain:

FS (File Separator	—1/12)
GS (Group Separator	—1/13)
RS (Record Separator	—1/14)
US (Unit Separator	—1/15)

FS is most inclusive, US the least inclusive. And we can consider the blank/space as the next lower order separator from these. Suppose we had a line of text like this:

(text1)US(text2)US(text3)RS(text4)US(text5)GS(text6)

On many terminals these delimiting control characters would not print, so we would see only a continuous stream. On others they might show as spaces. A TEX command to break the line at the record separator would be:

scan:line:*rs

The variable *left would contain "(text1)...(text3)". The variable *right would contain "(text4)...(text6)".

STICKS 0,1—Changing Sets (3)

This group is used for moving to and from alternate graphic and control sets. This includes ESCape (1/11), Shift Out (0/14), and Shift In (0/15).

These basic control characters have permitted design of a quite marvelous structure for extension and expansion. It allows us to code and classify most of the world's graphic symbols for computer storage, interchange, and display.

THE ASCII COLLATING SEQUENCE

The abstract aspects of ASCII have been treated. Now we come to some aspects of usage and implementation. Certainly one major use area is the ordering of files.

To put items in some ordering, the entire precedence relationship for that ordering must be defined. Higher or lower, precedes or follows, or whatever. For single characters, this ordering relationship is called the "collating sequence."

The ASCII standard used to say that the collating sequence for both graphics and control characters is defined simply by their binary representations. Later it added a warning that this collating sequence "cannot be used in many specific applications that define their own sequence." What an understatement!

The 1977 version hedges and speaks all around the problem without making it clear. It's not all that difficult. Suppose you have two files, and you want to know how they differ and/or how they are the same. For this purpose, the implied collating sequence (straight binary comparison) is just fine. The two files will be in the same order, and can be matched.

Whether that straight binary ordering can be used for any other purpose is doubtful. It won't work for signed numbers.

Ordering Numerals

Take these four values: 22, 13, minus 6, and minus 31. If the sign is placed before the digits, ordering by the ASCII collating sequence yields:

+13
+22
−06
−31

This is obviously worthless. It's because ordering is decided left to right, and the minus sign has a binary value 2 higher than the plus sign. Or if the sign were to follow the numeric values we would get:

```
06 –
13 +
22 +
31 –
```

because the complete decision is made in the leading digit. Again, a worthless sequence.

The way to achieve a proper ascending sequence is to separate the values into two groups, ordering those with plus signs in ascending sequence, and those with minus signs in descending sequence. Then put the plus group following the minus group. And vice versa for a total descending sequence. Notice that this works regardless of whether the sign precedes or follows the digits.

Ordering Alphabetic Fields

Alphabetic ordering is even more complex, particularly in handling both upper and lower case. Again the implied ASCII collating sequence can go wrong. People who have not studied the collating problem for data containing both upper and lower case are inclined to jump to wrong conclusions. I did myself, for the IBM Stretch computer in 1958, assigning the ascending binary sequence as AaBbCc. Using this for a telephone directory would give us the left hand column. The straight binary sequence of ASCII would yield the righthand column, just slightly different:

De Carlo	De Carlo
De La Rue	De La Rue
De Long	De Long
DeLair	DeLaRue
DeLancey	DeLair
DeLaRue	DeLancey
Delancey	Delancey
de Carlo	de Carlo
de la Rue	de la Rue
deLancey	deLancey

Either version will get a lot of anguished subscribers!

In the simplest case, two alphabetic items must be compared with the case ignored. Only if they are *then* equal is case called into consideration to break the tie, and it is also applied successively left-to-right!

In short, the upper and lower case versions of a letter do not both get full graphic significance. Typing either "Y" or "y" will indicate a "yes" reply, but "N" will not. Because the case distinction is minor, comparisons must first be made on major distinctions, with the minor distinctions used only as tiebreakers. Accenting of letters must also be considered minor, if accomplished via backspace, but this leads us into rules controlled by foreign governments, and won't be considered here.

Real life is more complicated than this. The ordering and sequencing of characters and words cannot always be accomplished by simple binary comparison of codes. There are constructions such as O'Reilly, l'Informatique (as data processing is called in French), and Smith-Jones—to say nothing of the Juniors, IIIs, Esq., FBCS (which I am), and so on.

Making an ASCII comparison, with the case as a minor, gives us:

De Carlo
de Carlo
De La Rue
de la Rue
De Long
DeLair
DeLancey
Delancey
deLancey
DeLaRue

Because we at first ignored case here, De Carlo and de Carlo have identical bit patterns. Tiebreaking is done by appending the binary pattern representing case, "0" for upper, "1" for lower. Specifically, 01001111 for De Carlo, 11001111 for de Carlo.

	D	E		C	A	R	L	O	
De Carlo	44	45	20	43	41	52	4C	4F	(4F)
de Carlo	44	45	20	43	41	52	4C	4F	(CF)

But even this method will not put "DeLaRue" and "De La Rue" in the same cluster. And surely this is desirable and even mandatory. It will require some special handling for spaces. The New York Telephone Company's document on this problem runs to several pages! They'd probably give you a copy upon request. You might need to know those rules before trying one of the toughest acts in data processing—putting last name first, or vice versa.

Using Controls in Ordering

There is one more aspect of ASCII useful to the ordering problem. In the days of punch cards, before computers, one often used several card files related by a key. A sorter (with pockets for the cards to drop into) might be used to select the cards for all redheaded females between 18 and 24 years of age. But these cards would have only the employee number and such characteristics on them. To get the name, address, and telephone number one might have to go to a second (related) deck of cards. So the first deck (the subset of interest) would be placed in the first hopper of a collator, and the deck with all names and phone numbers in the second hopper. Then a card would be fed from the first hopper, followed by successive cards from the second hopper, until a match was found on employee number. Obviously both decks had to be in the same ordering for this to work, and thus the term "collating sequence."

				b7	0	0	0	0	1	1	1	1
				b6	0	0	1	1	0	0	1	1
				b5	0	1	0	1	0	1	0	1
b4	b3	b2	b1		0	1	2	3	4	5	6	7
0	0	0	0	0	▨	▨	SP	0	▨	P	←	→
0	0	0	1	1	▨	▨	▨	1	A	Q	α	∨
0	0	1	0	2	▨	▨	"	2	B	R	⊥	⍴
0	0	1	1	3	▨	▨	▨	3	C	S	∩	⌈
0	1	0	0	4	▨	▨	▨	4	D	T	⌊	~
0	1	0	1	5	▨	▨	▨	5	E	U	∈	↓
0	1	1	0	6	▨	▨	&	6	F	V	×	∪
0	1	1	1	7	▨	▨	'	7	G	W	∇	⍵
1	0	0	0	8	▨	▨	(	8	H	X	∆	⊃
1	0	0	1	9	▨	▨	)	9	I	Y	ι	∧
1	0	1	0	10	▨	▨	*	:	J	Z	∘	⊂
1	0	1	1	11	▨	▨	+	;	K	[	÷	≤
1	1	0	0	12	▨	▨	,	<	L	\	⎕	|
1	1	0	1	13	▨	▨	−	=	M	]	≠	≥
1	1	1	0	14	▨	▨	.	>	N	↑	⊤	¯
1	1	1	1	15	▨	▨	/	?	O	_	○	▨

Figure 4. APL Character Set

In effect, we were sticking the cards of the first deck upright just in front of the corresponding cards of the second. To do this with ASCII requires that we have characters that collate lower than the lowest graphic, the space (2/1). We do have them. The best to use are NUL, FS, GS, RS, and US. Put one of these after each search key, then put the two files together and order them as adjoined. Now those records having a search key with one of our five control characters appended will precede the corresponding record having an ASCII graphic following the key.

Note that the four information separators (FS, GS, RS, US) are designed to collate just behind Space, in that order. This contiguity means that they can be used as a hierarchy of spaces of different class.

Other Collating Features

ASCII was designed when there was substantial investment in files already ordered on a Topsy-class IBM sequence, where the basic punctuation was low to the alphabet, but the digits were high to it. How then to accomodate this and still provide a 4-bit subset? My morning shower provided a solution (it still does!).

The 4-bit subset is formed of the first 10 graphics of stick 3 (the digit graphics) and the last 6 of stick 2. This job was shown shaded in the early forms of ASCII, but has all but disappeared from memory now. It enables stick 3 (with the digits and new special graphics) to be ordered high to all the others via passive logic, thus overcoming opposition to the adoption of ASCII.

ASCII AND PROGRAMMING LANGUAGES

Standard ECMA-53 (1978 Jan), "Representation of Source Programs for Program Interchange," gives the subsets and/or modifications of ASCII as they are used for these five programming languages:[4]

	NO. OF CHARACTERS USABLE	
Language	**Subset of ASCII**	**Other**
APL	57	32
Minimal BASIC	60	0
COBOL	51	0
FORTRAN	49	0
PL/I	55	2

Figures 4 through 8 are the character sets for these languages as given in ECMA-53. They show the only characters permissible for use in source programs, except for:

non-numeric literals	in COBOL
comment-entries	"
comment lines	"
character constants	in FORTRAN
comments	"
character-string-constants	in PL/I
comments	"

For these purposes only, other ASCII characters may be used, providing there is agreement between the sender and receiver for any interchange of source programs.

				b_7	0	0	0	0	1	1	1	1
				b_6	0	0	1	1	0	0	1	1
				b_5	0	1	0	1	0	1	0	1
b_4	b_3	b_2	b_1		0	1	2	3	4	5	6	7
0	0	0	0	0	▨	▨	SP	0	▨	P	▨	▨
0	0	0	1	1	▨	▨	!	1	A	Q	▨	▨
0	0	1	0	2	▨	▨	"	2	B	R	▨	▨
0	0	1	1	3	▨	▨	#	3	C	S	▨	▨
0	1	0	0	4	▨	▨	$	4	D	T	▨	▨
0	1	0	1	5	▨	▨	%	5	E	U	▨	▨
0	1	1	0	6	▨	▨	&	6	F	V	▨	▨
0	1	1	1	7	▨	▨	'	7	G	W	▨	▨
1	0	0	0	8	▨	▨	(	8	H	X	▨	▨
1	0	0	1	9	▨	▨	)	9	I	Y	▨	▨
1	0	1	0	10	▨	▨	*	:	J	Z	▨	▨
1	0	1	1	11	▨	▨	+	;	K	▨	▨	▨
1	1	0	0	12	▨	▨	,	<	L	▨	▨	▨
1	1	0	1	13	▨	▨	-	=	M	▨	▨	▨
1	1	1	0	14	▨	▨	.	>	N	^	▨	▨
1	1	1	1	15	▨	▨	/	?	O	_	▨	▨

Figure 5. Minimal BASIC Character Set

The TEX language has gone farther than this general caution. There the specific characters have permanent names. For example, one could say:

				b7	0	0	0	0	1	1	1	1
				b6	0	0	1	1	0	0	1	1
				b5	0	1	0	1	0	1	0	1
b4	b3	b2	b1		0	1	2	3	4	5	6	7
0	0	0	0	0	▨	▨	SP	0	▨	P	▨	▨
0	0	0	1	1	▨	▨	▨	1	A	Q	▨	▨
0	0	1	0	2	▨	▨	"	2	B	R	▨	▨
0	0	1	1	3	▨	▨	▨	3	C	S	▨	▨
0	1	0	0	4	▨	▨	$	4	D	T	▨	▨
0	1	0	1	5	▨	▨	▨	5	E	U	▨	▨
0	1	1	0	6	▨	▨	▨	6	F	V	▨	▨
0	1	1	1	7	▨	▨	▨	7	G	W	▨	▨
1	0	0	0	8	▨	▨	(	8	H	X	▨	▨
1	0	0	1	9	▨	▨	)	9	I	Y	▨	▨
1	0	1	0	10	▨	▨	*	▨	J	Z	▨	▨
1	0	1	1	11	▨	▨	+	;	K	▨	▨	▨
1	1	0	0	12	▨	▨	,	<	L	▨	▨	▨
1	1	0	1	13	▨	▨	-	=	M	▨	▨	▨
1	1	1	0	14	▨	▨	.	>	N	▨	▨	▨
1	1	1	1	15	▨	▨	/	▨	O	▨	▨	▨

Figure 6. COBOL Character Set

```
linefeed = "
            "  (actual line feed inside the quotes)
if *lf:eqs:linefeed....
```

				b7	0	0	0	0	1	1	1	1
				b6	0	0	1	1	0	0	1	1
				b5	0	1	0	1	0	1	0	1
b4	b3	b2	b1		0	1	2	3	4	5	6	7
0	0	0	0	0	▨	▨	SP	0	▨	P	▨	▨
0	0	0	1	1	▨	▨	▨	1	A	Q	▨	▨
0	0	1	0	2	▨	▨	▨	2	B	R	▨	▨
0	0	1	1	3	▨	▨	▨	3	C	S	▨	▨
0	1	0	0	4	▨	▨	$	4	D	T	▨	▨
0	1	0	1	5	▨	▨	▨	5	E	U	▨	▨
0	1	1	0	6	▨	▨	▨	6	F	V	▨	▨
0	1	1	1	7	▨	▨	'	7	G	W	▨	▨
1	0	0	0	8	▨	▨	(	8	H	X	▨	▨
1	0	0	1	9	▨	▨	)	9	I	Y	▨	▨
1	0	1	0	10	▨	▨	*	:	J	Z	▨	▨
1	0	1	1	11	▨	▨	+	▨	K	▨	▨	▨
1	1	0	0	12	▨	▨	,	▨	L	▨	▨	▨
1	1	0	1	13	▨	▨	-	=	M	▨	▨	▨
1	1	1	0	14	▨	▨	.	▨	N	▨	▨	▨
1	1	1	1	15	▨	▨	/	▨	O	▨	▨	▨

Figure 7. FORTRAN Character Set

and it would be true, because "*lf" is the permanent name of Line Feed. The control characters have names that are the letters from the ASCII chart, preceded by the asterisk to show that they are read-only

				b_7	0	0	0	0	1	1	1	1
				b_6	0	0	1	1	0	0	1	1
				b_5	0	1	0	1	0	1	0	1
b_4	b_3	b_2	b_1		0	1	2	3	4	5	6	7
0	0	0	0	0			b̸	0		P		
0	0	0	1	1			|	1	A	Q		
0	0	1	0	2				2	B	R		
0	0	1	1	3				3	C	S		
0	1	0	0	4			$	4	D	T		
0	1	0	1	5			%	5	E	U		
0	1	1	0	6			&	6	F	V		
0	1	1	1	7			'	7	G	W		
1	0	0	0	8			(	8	H	X		
1	0	0	1	9			)	9	I	Y		
1	0	1	0	10			*	:	J	Z		
1	0	1	1	11			+	;	K			
1	1	0	0	12			,	<	L			
1	1	0	1	13			-	=	M			
1	1	1	0	14			.	>	N	¬		
1	1	1	1	15			/		O	_		

Figure 8. PL/I Character Set

variables with permanent content. TEX can in fact operate upon all 256 characters of ASCII in an 8-bit byte, all 512 in a 9-bit byte.

Specific Notes on the Figures

APL—Sticks 6 and 7 (ordinarily lower case alphabet) are replaced entirely except for the DELete position.
—Space is nonprinting, although the symbol shown is SP.
—Ampersand (2/6) is not used for writing source programs, except as the last character of a line if that line is to be continued on the next line.

PL/I—In position 2/1, the exclamation point is replaced by a vertical bar for OR.
—In position 5/14, the circumflex is replaced by the symbol shown, for NOT.
—If you have to use your terminal for both PL/I and some other programming language, forget that foolishness. You can get by with the exclamatin point as OR, and the circumflex as NOT. The important point in source program interchange is to have the encoded representations of the characters exchanged correctly.

(all)—Although the character BLANK (space) is shown as the flagged lower case "b" in the FORTRAN and PL/I sets, there is no printing graphic to indicate it. For all practical purposes, it is really the Space of ASCII (2/0).
—Four of these five languages (not APL) have the "$" shown in 2/4. When the International Reference Version of the code is used, this becomes the universal currency symbol, which is also acceptable.
—Minimal BASIC uses "#," which is the International Reference Version symbol. The national symbols, such as the English pound sign, are also acceptable.

ASCII AND MEDIA

ASCII and Punch Cards

Reading the punching equipment for punch cards, be ing very mechanical, is so expensive that microcomputer people are unlikely to use them. So you might ask why we bother here with the representation of ASCII on this medium? I can think of at least three reasons:

- A scientist at the U.S. National Bureau of Standards said once that if punch cards were on the way out, it was the only product he ever saw dying on an upward usage curve. Thus they are likely to be around for a long time, and you may need to transfer some of those files to other media that you do use.
- There is some likelihood that microcomputers could be used in the reading and punching equipment itself, to make it less expensive.
- ASCII users are going to be confronted for a while yet with one of the several versions of IBM's EBCDIC, and the punch card assignments provide the only legitimate link for conversion of EBCDIC files to ASCII.

So Figure 9 defines the hole patterns for the binary encodings. And Figure 10 defines the encodings for the hole patterns. Don't worry

	ISO	ECMA	ANSI	FIPS PUB	CSA	BS	AS	CCITT	JIS	GOST
Hollerith Punched Card Code	1679 2021	44	X3.26-1970 $4.25	14	Z243.14 .36	4636/3 /4	1063			
Track Assignment - 25.4 mm Perf. Tape	1113	10	X3.6-1965 $3.00	2	Z243.8	3880/3	1062		C6221	
Track Assignment - 12.7 mm Mag Tape 200 cpi NRZI 7-track	1861	5	X3.14-1973 $3.25			3968	1007			
Track Assignment - 12.7 mm Mag Tape 800 cpi NRZI 9-track	962 1863	12	X3.22-1973 $3.75	3-1		4503/1	1009		C6222	
Track Assignment - 12.7 mm Mag Tape 1600 cpi PE 9-track	3788	36	X3.39-1973 $3.75	25		4503/2				
Track Assignment - 12.7 mm Mag Tape 6250 cpi GCR 9-track	DP 5652		X3.54-1976 $5.25							
Labeling & File Structure - 12.7 mm MT	1001	13	X3.27-1977 (unpriced)		Z243.7	4732	1068			
Track Assignment - Magtape Cassette 3.81 mm, 32 bpmm	3275 3407	34	X3.48-1977 $5.75			5079/1				
Labeling & File Struct. - 3.81 Magtape Cassette	DIS 4341	41								
Track Assignment - 6.35 mm Cartridge Tape 64 bpmm PE	DIS 4057	46	X3.56-1977 $4.24							

Table 1b. Standards for ASCII on Physical Media

	ISO	ECMA	ANSI	FIPS PUB	CSA	BS	AS	CCITT	JIS	GOST
Bit Sequencing in Serial Transmission			X3.15-1976 $3.00	16-1				V.4 X.4		
Char. Structure & Parity Sense - Serial-by-Bit			X3.16-1976 $3.50	17-1				V.4 X.4		
Char. Structure & Parity Sense - Parallel-by-Bit			X3.25-1976 $3.50	18-1				V.4 X.4		
Procedures for Using Commun. Control Chars.	1745	16	X3.28-1976 $10.50		Z243.13	4505/1	1484/1			
Message Heading Formats	1745		X3.57-1977 $5.25							

Table 1c. Standards for ASCII in Communications

about the inconsistency in the relationships. Nothing can be done about it now, because it started with Herman Hollerith's first U.S. Census machines in 1890. At first only digits and + and − signs were used. Then the code was expanded to the upper case alphabet. And other special characters for commercial use. When FORTRAN came along in 1964, it turned out that the limited capability of the subset of a 6-bit set would not permit the graphics needed for scientific work. For a long while there were dual graphic representations for several of the punch card code combinations, and this carried over into printer chains, and so on.

The only logic that the patterns follow is that they do or do not have a punch from among these six possibilities:

12-punch (top row)
11-punch (next to the top row)
0-punch

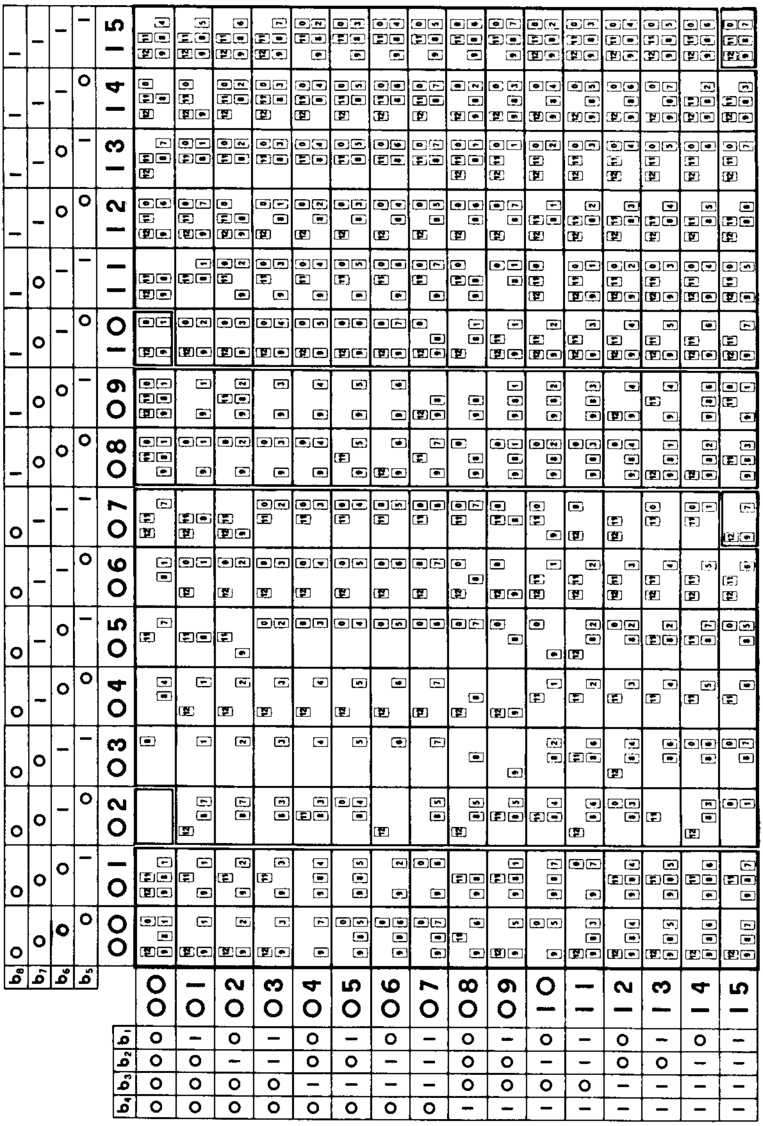


Figure 9. Card Hole Patterns Assigned by Codes

	12	11	0		12 0	12 11	11 0	12 11 0
	02/06	02/13	03/00	02/00	07/11	07/12	07/13	11/10
1	04/01	04/10	02/15	03/01	06/01	06/10	07/14	13/09
2	04/02	04/11	05/03	03/02	06/02	06/11	07/03	13/10
3	04/03	04/12	05/04	03/03	06/03	06/12	07/04	13/11
4	04/04	04/13	05/05	03/04	06/04	06/13	07/05	13/12
5	04/05	04/14	05/06	03/05	06/05	06/14	07/06	13/13
6	04/06	04/15	05/07	03/06	06/06	06/15	07/07	13/14
7	04/07	05/00	05/08	03/07	06/07	07/00	07/08	13/15
8	04/08	05/01	05/09	03/08	06/08	07/01	07/09	14/00
9	04/09	05/02	05/10	03/09	06/09	07/02	07/10	14/01
8 2	05/11	05/13	05/12	03/10	12/04	12/11	13/02	14/02
8 3	02/14	02/04	02/12	02/03	12/05	12/12	13/03	14/03
8 4	03/12	02/10	02/05	04/00	12/06	12/13	13/04	14/04
8 5	02/08	02/09	05/15	02/07	12/07	12/14	13/05	14/05
8 6	02/11	03/11	03/14	03/13	12/08	12/15	13/06	14/06
8 7	02/01	05/14	03/15	02/02	12/09	13/00	13/07	14/07

12	11	0		12 0	12 11	11 0	12 11 0	
10/08	11/01	11/09	06/00	12/03	12/10	13/01	13/08	8 1
00/01	01/01	08/01	09/01	10/00	10/09	09/15	11/11	9 1
00/02	01/02	08/02	01/06	10/01	10/10	11/02	11/12	9 2
00/03	01/03	08/03	09/03	10/02	10/11	11/03	11/13	9 3
09/12	09/13	08/04	09/04	10/03	10/12	11/04	11/14	9 4
00/09	08/05	00/10	09/05	10/04	10/13	11/05	11/15	9 5
08/06	00/08	01/07	09/06	10/05	10/14	11/06	12/00	9 6
07/15	08/07	01/11	00/04	10/06	10/15	11/07	12/01	9 7
09/07	01/08	08/08	09/08	10/07	11/00	11/08	12/02	9 8
08/13	01/09	08/09	09/09	00/00	01/00	08/00	09/00	9 8 1
08/14	09/02	08/10	09/10	14/08	14/14	15/04	15/10	9 8 2
00/11	08/15	08/11	09/11	14/09	14/15	15/05	15/11	9 8 3
00/12	01/12	08/12	01/04	14/10	15/00	15/06	15/12	9 8 4
00/13	01/13	00/05	01/05	14/11	15/01	15/07	15/13	9 8 5
00/14	01/14	00/06	09/14	14/12	15/02	15/08	15/14	9 8 6
00/15	01/15	00/07	01/10	14/13	15/03	15/09	15/15	9 8 7

Figure 10. Codes Assigned by Card Hole Patterns

8-punch
9-punch (bottom row)
a punch from among the digits 1 through 7

Including the no-punch-at-all combination (NUL), this gives 256 combinations, just right for the 8-bit code. Although ASCII was technically only a 7-bit code at the time this rule was formulated, it was felt necessary to plan ahead a little.

ASCII and Magnetic Tape

Figure 11 gives a compact representation of several relationships, among which is the assignment of ASCII bit pattern to 9-track magnetic tape. The jumbled assignment may remind you of the "firing order" for the cylinders of an automobile engine. In fact, we used to call it just that. It was intentional for increased reliability. As in so many cases, better technology has removed the need for peculiar design, but the assignments are unchangeable because of data file investment.

There is no parallelism in recording and reading on cassettes and cartridges. The ASCII bits are recorded serially in the track. Thus Figure 11 does not consider these media.

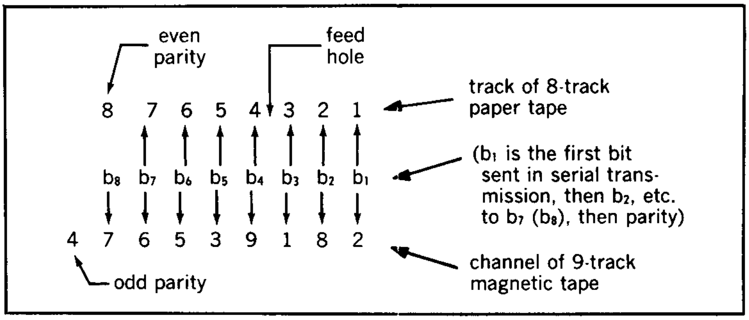

even parity
feed hole
8 7 6 5 4 3 2 1
track of 8-track paper tape
b8 b7 b6 b5 b4 b3 b2 b1
(b1 is the first bit sent in serial transmission, then b2, etc. to b7 (b8), then parity)
4 7 6 5 3 9 1 8 2
odd parity
channel of 9-track magnetic tape

Figure 11. Bit Sequences—Media and Communications

ASCII and Communications

Not only is the topic of ASCII and communications a very complex and large dissertation for this article—it is also undergoing substantial rethinking, enlargement, and invention. You will have to follow on your own the workings of the CCITT, the various networking systems of the several large and many small manufacturers of computer systems, and the offerings of the common carriers—either on the local distribution system (via ATT) or direct distribution (via Satellite Business Systems).

Many of the existing standards are listed in Table 1c. Many more are under development. Arguments are raging internationally on the merits of packet switching, byte protocols, value-added systems, open-working systems, tariffs, data movement across national borders, the X.25 protocol, etc., etc. ATT is offering a new service bureau because they suddenly discovered data-under-voice (DUV). All I can tell you now is that it is all based upon ASCII, and the proposed protocols are all dependent upon the ASCII control characters in sticks 0 and 1. It will take years for this to shake out, and for now all one can do is get on the CBEMA mailing list (see "Where to Get More Information" at the end of this article).

ASCII AND THE METRIC SYSTEM

The full ASCII graphic set (both cases) is sufficient to indicate all symbols and prefixes of the SI (International System of Units, the new metric system), with three exceptions. They are the Greek letters "omega" for "ohm," and "mu" for "micro," and the degree symbol for Celsius temperature. These three characters will be provided in 8-bit ASCII. Meanwhile, for these, and also for such equipment that has only a single case, there is a standard way of representing the SI units and prefixes. This is given in International Standard 2955, "Representations of SI Units and Other Units for Use in Systems with Limited Character Sets," and also in American Standard X3.50-1976.

To keep the record straight, let's first look at the characters used for the prefixes. They're shown in Table 3, which indicates multiples from 10^{-18} up to 10^{-18}.

10^{+i}	i	10^{-i}
exa (E)	18	atto (a)
peta (P)	15	femto (f)
tera (T)	12	pico (p)
giga (G)	9	nano (n)
mega (M)	6	micro (μ)
kilo (k)	3	milli (m)
hecto (h)	2	centi (c)
deka (da)	1	deci (d)

Table 3. Metric Prefixes

Above 3 there are no powers except multiples of 3. This practice breeds better comprehension, like marking off three's in writing numbers of many digits. Also, as a memory convenience, all symbols are capitals for powers greater than +3. And there are no conflicts with the symbols for the units of measurement.

Now, again for the record, here are the ASCII character(s) used as symbols for the units:

A	ampere	cd	candela
Bq	becquerel	d	day
C	coulomb	g	gram
°C	degree celsius	h	hour
F	farad	l	litre
Gy	gray	lm	lumen

H	henry	lx	lux
J	joule	μ	micro
K	kelvin	m	metre
N	newton	min	minute (time)
Ω	ohm	mol	mole
Pa	pascal	rad	radian
S	siemens	s	second (time)
T	tesla	sr	steradian
V	volt	t	tonne/metric ton/ megagram
W	watt		
Wb	weber		

Table 4. Metric Units

Table 4 shows the rules clearly. Units not named after people are all lower case, as shown in the righthand column (although I do know a Mr. Day). In the lefthand column are the units that are named after people. The names of the units are not capitalized at all, but the symbols begin with an upper case letter.

I said previously that there were no conflicts between unit and prefix symbols. But you've probably noticed "d" for both "day" and "deci," "h" for both "hour" and "hecto," "m" for both "metre" and "milli," and "T" for both "tesla" and "tera." OK. But there isn't any confusion in actual usage, because the prefix precedes the unit:

dd	is a deciday (2.4 hours)
hh	is a hectohour (100 hours)
hH	is a hectohenry (but don't ever use the term)
mm	is a millimetre
Mm	is a megametre ($\frac{1}{300}$ the distance light travels in a second)
TT	is a teratesla (Wow!)

I am not suggesting that the prefixes should be applied to other than the primary metric units (the second is the primary time unit; hour and day are not), even though the timesharing system I customarily use figures my time in millihours. But when you get accustomed, the prefixes are very valuable in other ways. For example, an American billion is a kilomillion, whereas the British billion is a megamillion! And my metric teaching program understands such things as kilofathoms.

The "space" character is also vital to correct SI usage. It must occur between values and units, like 123.6 mm, and 22°C.

And don't forget another peculiarity of ASCII as an international alphabet. (1/14) is absolutely not defined as a "decimal point" (nor is it defined as "period," which in Europe is "full stop"). For most of the rest of the world, the comma (1/12) is the decimal marker, and the period is used to mark off threes. That's why the recommended practice for marking off threes is to use the space, not either comma or period. E.g., "1 234 567 mm."

To save you the bother of looking up the standards for use with limited character sets, here is the algorithm:

1. If you have ASCII with both cases of alphabet, the three missing symbols are handled as:

ohm	for Ω
Cel (initial cap)	for °C
u (lower case)	for μ (micro)

2. If you have only one case of alphabet (either upper or lower), use it, and these three replacements remain as:

OHM		ohm
CEL	or	cel
U		u

And in addition:

S (siemens)		SIE		sie
h (hour)	become	HR	or	hr
t (tonne)		TNE		tne

Examples:

16 UOHM is 16 μΩ

373.15 K = 100 Cel

Notice that no plurals are used in symbol combinations—MICROHMS, but UOHM.

ASCII AND KEYBOARDS

Technically, a keyboard is an ASCII keyboard if it generates the proper codes for the full set of ASCII graphic and control characters. Moreover, none of the graphic characters should have any control properties.

There are many types of special keyboards—Dvorak, a two-sided one used like an accordian with the hands in a vertical planes, Touch-Tone and its derivatives, etc. There are no formal standards to relate these keyboards to ASCII. For typewriter-style keyboards, however, there are two versions given in the American National Standard. One is derived from the usual electric typewriter keyboard, the other is called the "bit-paired" keyboard. Only the bit-paired keyboard will be shown and discussed here, because the other form is the subject of proposals for extensive change due to the growth of Word Processing. ANSI Committee X4A12 is studying this now.

The bit-paired keyboard was designed for minimum circuitry cost. Thus the "at" symbol (4/0) is paired with the accent grave (6/0), "A" (4/1) with "a" (6/1), and " + " (2/11) with ";" (3/11). Thus the shift key affects each other key by only a 1-bit change.

This keyboard is shown in Figure 12. It is the interchange keyboard of ECMA-23. The numbered arrows key to the notes on changes that would make this ECMA keyboard into the ANSI keyboard for ASCII. It is also equivalent to the keyboard of ISO Standard 2530-1975.[5]

Notes for Figure 12

1. For the ANSI keyboard, this key is put to the right of the circumflex key, on the top row (see Note 6). The Shift Key is put

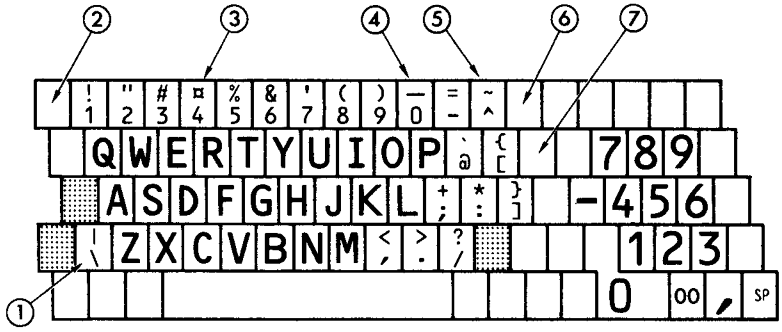


Figure 12. Basic ISO/ECMA/ASCII Keyboard

in its place.

2. *If this key exists and is available, the ECMA and ISO standards put the underscore here, removing it from the "zero" key.*
3. *The ANSI keyboard of course puts a "$" here in place of the international currency symbol.*
4. *This is where the underscore is removed for the 48-key keyboard (see Note 2).*
5. *Here ECMA and ISO show the "overline" instead of the "tilde." It's a question of styling.*
6. *The ANSI keyboard has the reverse slash and vertical bar here, rather than between the shift key and "Z" (see Note 1).*
7. *The ANSI keyboard specifies the underscore here, in both shifts, rather than the positions shown as options in Notes 2 and 4. Practically no keyboards follow this. In fact, as I am entering this text, this is the* **only** *key where my Infoton Vistar deviates from the ANSI standard. It has Line Feed there, with Return to its right—a very sensible arrangement.*

Customarily, the Control Key is also tied to bit-pairing in such keyboards. The standards recommend that characters created in combination with the Control Key should use the graphic key in sticks that are 4 or 6 units higher. Thus "X" (5/8) or "x" (7/8) in combination with the Control Key produce CAN (1/8). Unfortunately this also means that Control-C generates ETX (0/3). And whereas Control-X as CAN is used frequently, to erase an input line of text, ETX is not often wanted. Yet it is a common miskeying to hit C rather than X. In many timesharing systems you will get a disconnect rather than a line delete.

Control and Function Keys

The so-called "QWERTY" arrangement is prevalent throughout the Anglo-Saxon world. Even the French "AZERTY" set is being considered for change. But on top of these basics there are hundreds of keyboard varities. Some of them have "dead keys" (i.e., the platen or printing element is not advanced when they are hit). This avoids having to use BS for accented letters, but it also creates difficulties in code generation.

There are some general good practices that ASCII keyboards should follow. To facilitate usage by those experienced with typewriters, all controls not used with typewriters should be located outside the customary touch-typing area. As a specific example, the Break/Interrupt key should be located where it is a definite effort to reach it (not mixed in with the keyboard). ISO 3244 may be consulted for these considerations.

Function Keys are those that generate sequences of more than one ASCII character. Examples are cursor keys, Erase-to-EOL, etc. They should be located in special clusters. Most importantly, they must all generate ASCII codes for transmission when in character-at-a-time mode. I know of video terminals where the cursors do not generate codes, as they should not while in full page buffered mode; but they still operate in line mode without generating codes. In this case the screen is alterable, but there is no way of detecting it in the computer.

Many keyboards will have some function keys that are unlabeled, for do-it-yourself assignment. These should also be clustered separately, and generate code sequences when in line mode.

ASCII AND DISPLAY/PRINTING

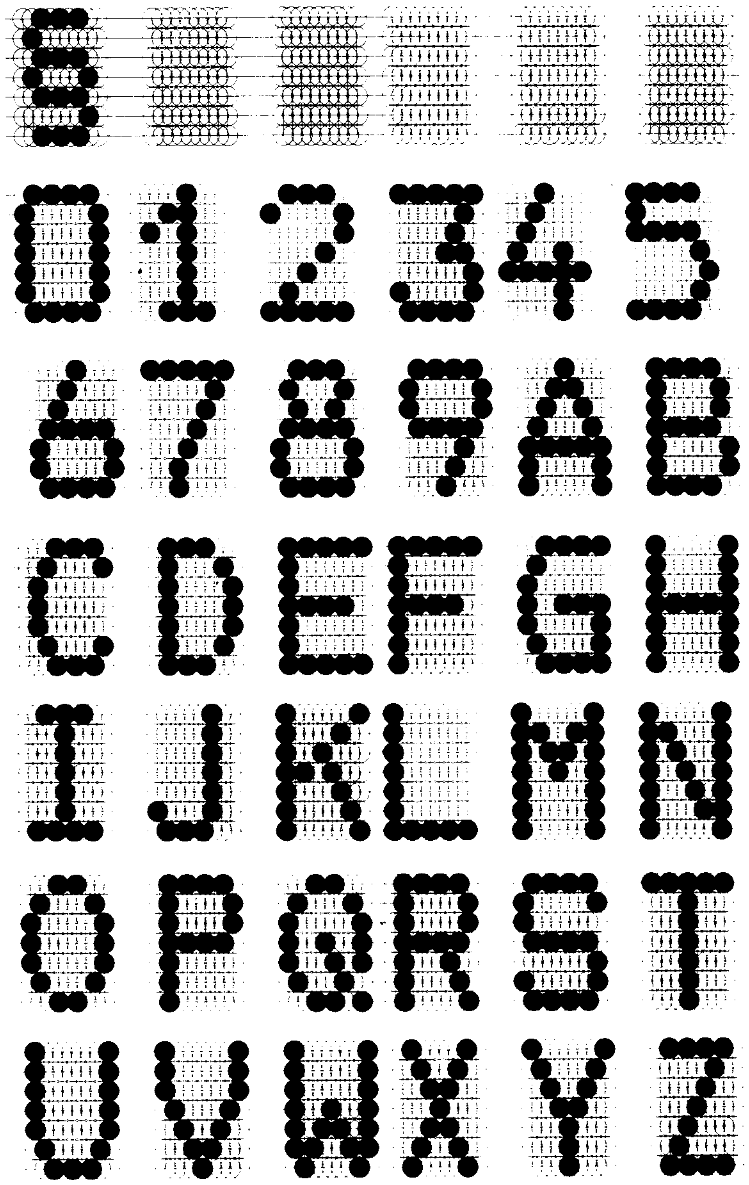
When ASCII characters are displayed, it may be on a video screen, paper, or COM (microfiche).

On the video screen there are a number of methods to form the characters, mostly at the manufacturer's preference. They are usually at pica (constant-width) spacing for economy, so an approximation of graphic quality (such as typesetting) is not obtainable. When lower case is available, the risers and tails extend above and below the line for some screens. In others, they fall within the boundary lines of the upper case characters. They may be shown in inverse video (light background block), or highlighted by different brightness or blinking. Controls for this work will be taken up later in this article.

For paper copy one usually finds either direct impact of a formed letter, or stylus printing. Either method is suitable to proportional spacing if desired. Recently there has been a general trend toward using the 7×9 dot matrix shapes of ECMA Standard 42 for stylus printers. This set of graphics is shown in Figure 13.

For hard print elements, of course, one can get a nearly infinite variety of styles and fonts. There are only two, however, specifically associated with computers—OCR-A and OCR-B. "OCR" stands for "Optical Character Recognition," meaning that the shapes are so styled that a computer-controlled scanner can read the characters as printed on paper, and encode them directly from their shapes.

OCR-A is not suitable for human reading. It's the funny looking one with the diamond-shaped letter "Oh." I won't dignify it by showing the font here. It was thought formerly, with technology of that day, that making humans work harder to read letters would make it easier and thus cheaper for computers to read them. This argument turned out to



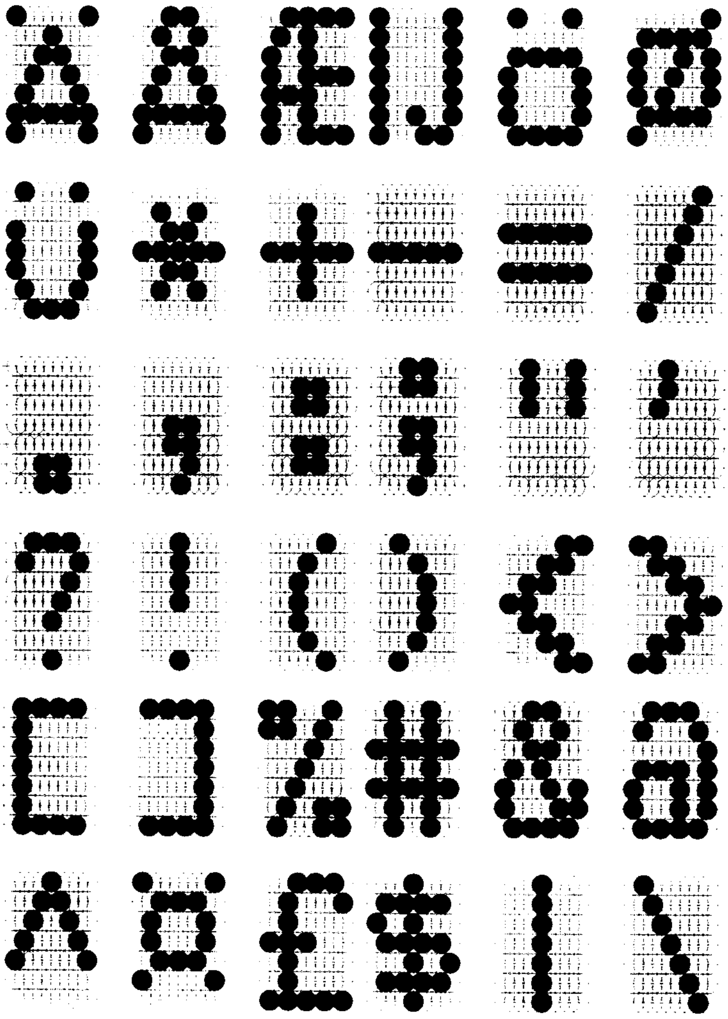


Figure 13. 7 x 9 Dot Matrix Shapes

be specious, and with today's technology there is no need to use anything other than OCR-B.

OCR-B is specified in ISO 1073/2, ECMA-11, and ANSI X3.49. It is the font shown in Figure 14. I have it on my IBM golfball typewriter at home, and on my daisywheel element at the office. So it should be available for most hard elements, including the carousel type.

The first six rows correspond to ASCII sticks 2-7. In the first row, the pound and universal currency symbol are for replacement as needed.

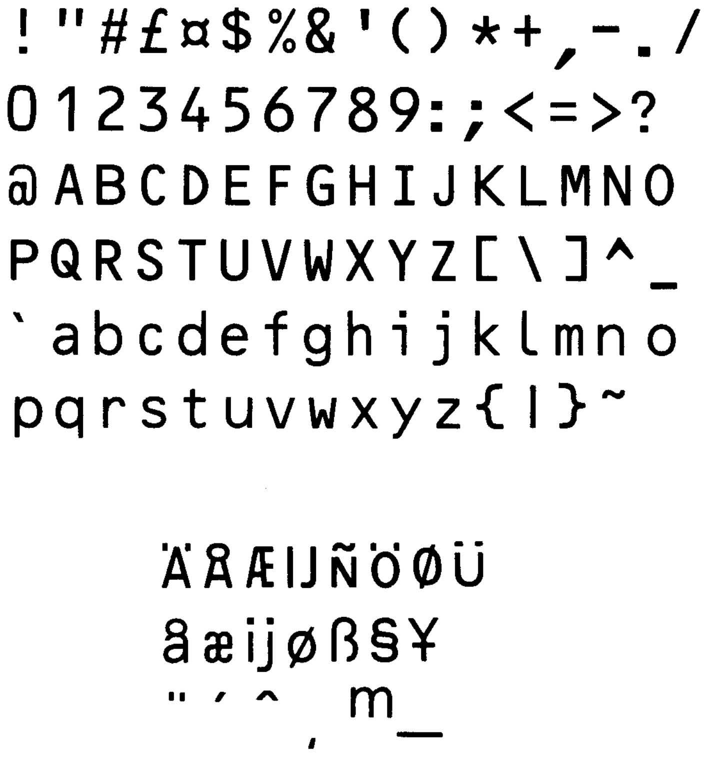

! " # £ ¤ $ % & ' ( ) * + , - . /
0 1 2 3 4 5 6 7 8 9 : ; < = > ?
@ A B C D E F G H I J K L M N O
P Q R S T U V W X Y Z [\ ] ^ _
` a b c d e f g h i j k l m n o
p q r s t u v w x y z { | } ~
Ä Å Æ IJ Ñ Ö Ø Ü
å æ ij ø ß § ¥
" ´ ^ ¸ m _

Figure 14. OCR-B Font

In the fourth row, the underline is discontinuous; a continuous form is shown in the auxiliary set. This set also contains a matching accent acute instead of single quote, the real circumflex (besides an up arrowhead), a cedilla, and an alternative "m" for variable pitch printing only.

CODE EXTENSION—GENERAL PRINCIPLES

Over ten years ago it was recognized that ASCII was the basis for codification of the various symbols used throughout the world. Through it, libraries could store encoded books as well as printed books. And while electronic mail may be quite simple with ASCII and its Roman alphabet, that's not the alphabet of all countries. The USSR uses Cyrillic, the Japanese use Katakana, and the Arab world uses its own semi-script alphabet. Moreover, to send a mathematics textbook by electronic mail one would have to be able to encode the formulas and special symbols peculiar to mathematics, which includes many Greek characters!

This is where the ESCape character and ESCape sequences come in. You can get the whole complicated story from ISO Standard 2022 (or ECMA-35) on Code Extension Procedures. But it will be easier to think of reproducing many ASCII Code Tables on the pages of a book, then replacing the ASCII symbols on all but the first page with the other alphabets we need.

Then we make sure that everyone in the world has the same (code) book. (The resemblance to military code books is intentional.) That's done by registering the page number assignment to characters (actually either a control set or a graphic set, but not both) with the French Standards Body AFNOR. That's the Association Francaise de Normalisation, Tour Europe Cedex 7, 92080 Paris La Defense, FRANCE. But you'll find it perhaps easier to get it from ANSI.

The registration procedure is spelled out in ISO Standard 2375. It is carefully controlled to prevent frivolity and cluttering up the assignment books, for that all costs money. But the important control and graphic sets of the world may be registered and assigned their own unique ESCape sequence for calling or invoking them.

CODE EXTENSION—BASIC RULES

The control ESC, when encountered in a datastream, means that all characters following it, up to and including the first character from sticks 3 to 7, have special interpretation. The delimiting character is called a "final" (F). Those between ESC and the final are called "intermediates" (I). All of the codes in stick 2 can serve as intermediate characters in ESCape sequences of 3 or more characters in length. The entire group of characters from ESC through the final is called an ESCape sequence.

ESCape sequences obviously require buffers for interpretation, for we cannot know, when they begin, how long they will be. Sequences of length 2 are for single controls. If the character following ESC is from stick 3, the sequences are for private usage, of the class Fp. If it is from sticks 4 or 5, they mean single controls, of the class Fe, from an appropriate set of 32. If from sticks 6 or 7 (except 7/15), they are of the class Fs, composed of single controls. This is elementary extension.

A more complex type of extension is the simulation of one or more 8-bit character sets by alternating between two 7-bit sets. The home base set consists of the C0 (32 controls) set and the G0 (94 graphics plus space and DEL). The alternate sets consist of the C1 (32 controls) set and the G1 (94 graphics plus space and DEL). The 8-bit set (it doesn't have to be just theoretical if you have a full 8-bit capability) consists of the four parts C0-G0-C1-G1.

These four types of sets are all invoked (designated) by 3-character ESCape sequences in this manner, where F is the final (3rd) character:

Sequence	Invokes Set Type
ESC 2/1 F	C0
ESC 2/2 F	C1
ESC 2/8 (or 2/12) F	G0
ESC 2/9 (or 2/15) F	G1

The final character "F" selects the particular set to invoke. Once invoked, encountering or entering an SO shifts to the G1 set in force; an SI shifts to the G0 set in force. SO and SI do not affect the control set.

ISO Standard 2022 defines these matters in far more detail, but that is enough for here. That document is complicated and ingenious, and deserves substantial study.

THE CODE EXTENSION REGISTRY

Table 5a identifies the graphic sets registered to date. Table 5b identifies the control sets registered to date. Remember that these assignments, once registered, may never be changed!

Regis. No.	Final Char.	Name
002	4/0	IRV (Intl. Reference Version) Graphics
004	4/1	UK Graphics
006	4/2	US Graphics (ASCII)
008-1	4/3	NATS Main Graphic Set (Finland, Sweden)
008-2	4/4	NATS Additional Set (Finland, Sweden)
009-1	4/5	NATS Main Graphic Set (Denmark, Norway)
009-2	4/6	NATS Additional Set (Denmark, Norway)
010	4/7	Swedish Basic Graphics
011	4/8	Swedish Graphics for Names
013	4/9	JIS Katakana Graphics
014	4/10	JIS Roman Graphics
015	5/9	Italian Graphics
017	5/10	Spanish Graphics
018	5/11	Greek Graphics
019	5/12	Latin-Greek Graphics
021	4/11	German Graphics
025	5/2	French Graphics
027	5/5	Latin-Greek Mixed.Graphics (Greek Capitals only)
031	5/8	Greek Alphabet Set for Bibliographic Use

For a G0 set the ESCape sequence is
ESC 2/8 plus the final shown.
For a G1 set the ESCape sequence is
ESC 2/9 plus the final shown.

Table 5a. Registered Graphic Character Sets

Regis. No.	Final Char.	Name
001	4/0	ISO 646 Controls
007	4/1	Scandinavian Newspaper Controls
026	4/3	IPTC Controls

The ESCape sequence for a C0 set is
ESC 2/1 plus the final shown.

Table 5b. Registered Control Character Sets

The registry set is available from AFNOR for approximately 172 French francs, say $35. It would be vital for an equipment or software manufacturer to have it, and it comes in a beautiful 4-ring binder symbolizing worldwide interchange compatibility. But the summary provided here will fill most needs.

CONTENT OF THE EXTENDED SETS

Figure 15a shows, against the ISO Code, International Reference Version, how the other graphic sets differ in the column/row positions

col	02				03		04	05					06	07			
row	01	02	03	04	10	15	00	11	12	13	14	15	00	11	12	13	14
002	!	"	#	¤	:	?	@	[	\	]	^	_	`	{	|	}	‾
004			£	$													
006				$													~
008-1				$			ua	Ä	Ö	Å	■		ub	ä	ö	å	
009-1		≪	≫	$			ua	Æ	Ø	Å	■		ub	æ	ø	å	
010								Ä	Ö	Å				ä	ö	å	
011							É	Ä	Ö	Å	Ü		é	ä	ö	å	ü
014				$					¥								
015			£	$			§	°	ç	é			ù	à	ò	è	ì
017			£	$			§	¡	Ñ	¿					ñ	ç	~
018			£	$			ʼ										
019			£	$													¨
021				$			§	Ä	Ö	Ü				ä	ö	ü	ß
025			£	$			à	°	ç	§				é	ù	è	¨
027	Ξ		Γ		Ψ	Π	Δ	Ω	Θ	Φ	Λ	Σ					

Figure 15a. Registered Graphic Character Substitution

008-2 and 009-2 are shown in Figure 15b. Here these are not exceptions from the IRV, but rather the only graphics assigned in the set. The additions are necessary to set type for newspapers throughout Scandinavia. See the Crossbar-D, Crossbar-O, the A-E ligature, and the Icelandic Thorn.

col	04			05				06			07			
row	01	04	05	00	05	11	12	01	04	05	00	05	11	12
008-2	À	Đ	É	Þ	Ü	Æ	Ø	à	đ	é	þ	ü	æ	ø
009-2	À	Đ	É	Þ	Ü	Ä	Ö	à	đ	é	þ	ü	ä	ö

Figure 15b. Registered Additional Graphic Sets

shown. The rows are keyed to Table 5a, reminding you that ASCII is "006," or ISO 646-006."

From this figure we can see that many countries need accented letters as individual characters, not compound via BS (BackSpace). This is particularly true for the double sets 008 and 009, for Scandinavian newspaper transmission, which have characters that cannot be made from ASCII in compound form. For example—Ring-A, a solid, and the angle open and closed quotes.

Figure 1a doesn't show Set 031 because it deviates more and is not of that much general interest. It doesn't show the Japanese Katakana set because that is completely different from the IRV. In fact, Japanese Industrial Standard C6220-1969 is an 8-bit coded set with the IRV (see Set 014 for the dollar and yen signs) in the lower (bit 8 = 0) portion, and Set 013 in the higher portion, with space reserved for future additional controls. This Set 013 is shown in Figure 16. It is shown in its high-order position, to indicate the card codes at the same time.

Figure 16 also shows the Cyrillic set of the USSR state standard GOST 13052-67, but it is not half of an 8-bit set as the Japanese do it. Rather it is another page of extensions. After SO (Shift Out) is used, the Russian register is operative. Following SI (Shift In) it is the IRV. Although this set has no registry number now, it was submitted recently by ECMA, and we expect an assignment soon. By the way, both Katakana and Cyrillic are shown in their OCR font.

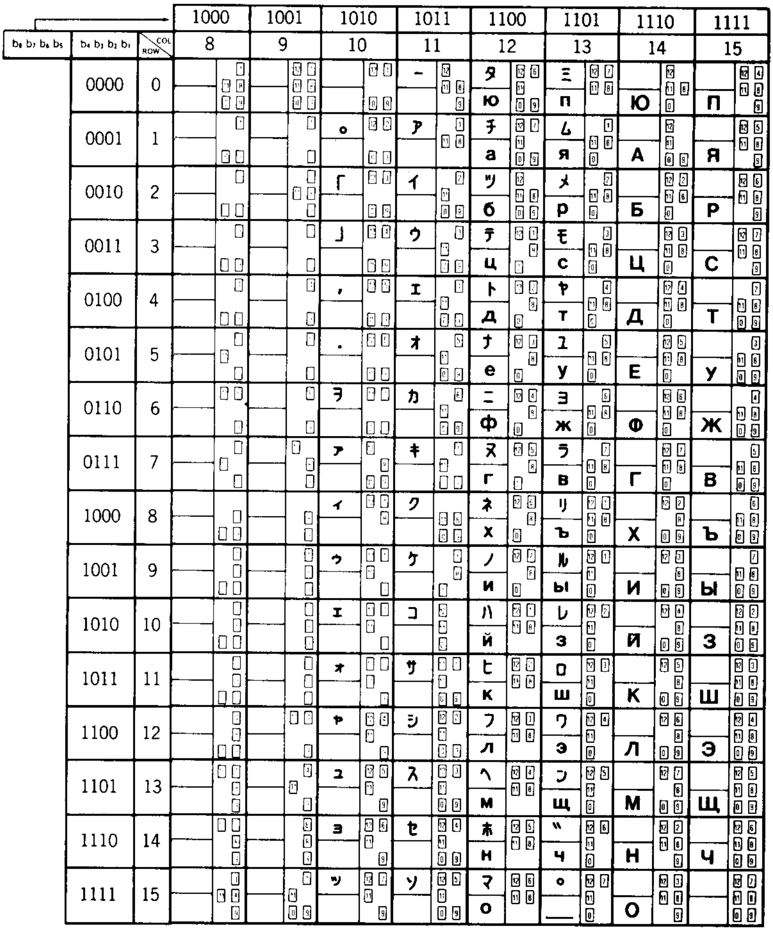
Figure 17 shows the contents of the registered control sets. Set 007 serves as control set for the graphic sets 008-1,2 and 009-1,2 for Scandinavian newspaper transmission. And set 026 is the control set for the worldwide newspaper transmission, defined by the IPTC (International Press Telecommunications Council). The 18 control positions not shown, and those where there is no entry, are the same as in the International Reference Version (646-001).

These newspapers are driving composition equipment, not line printers, so they don't need VT and FF. Their set is already defined, so they don't need SO and SI. They have (properly) assigned meaning to three device controls. And they're probably not doing payroll, so they don't need the four information separators. But they do transmit, and instead of choosing their own functions and placement they have chosen to be a registered variant of the ISO Code. And all variants within this controlled and registered cluster can at least recognize each other, even if they can't print it!

CODE EXTENSION IN ACTION

To illustrate the operation of code extension, let's imagine some equipment that may not exist now:

- A microfiche reader with automatic location controls.
- A microfiche with ASCII (the 8-bit form) on the first two pages, the other pages containing other sets such as Katakana, Cyrillic, Arabic, Greek, Hebrew, mathematical symbols, astronomical sym-



b8 b7 b6 b5			1000	1001	1010	1011	1100	1101	1110	1111
	b4 b3 b2 b1	COL / ROW	8	9	10	11	12	13	14	15
	0000	0				ー	タ ю	ミ п	Ю	П
	0001	1			。	ア	チ а	ム я	А	Я
	0010	2			「	イ	ツ б	メ р	Б	Р
	0011	3			」	ウ	テ ц	モ с	Ц	С
	0100	4			、	エ	ト д	ヤ т	Д	Т
	0101	5			・	オ	ナ е	ユ у	Е	У
	0110	6			ヲ	カ	ニ ф	ヨ ж	Ф	Ж
	0111	7			ァ	キ	ヌ г	ラ в	Г	В
	1000	8			ィ	ク	ネ х	リ ь	Х	Ь
	1001	9			ゥ	ケ	ノ и	ル ы	И	Ы
	1010	10			ェ	コ	ハ й	レ з	Й	З
	1011	11			ォ	サ	ヒ к	ロ ш	К	Ш
	1100	12			ャ	シ	フ л	ワ э	Л	Э
	1101	13			ュ	ス	ヘ м	ン щ	М	Щ
	1110	14			ョ	セ	ホ н	゛ ч	Н	Ч
	1111	15			ッ	ソ	マ о	゜	О	

Figure 16. Katakana and Cyrillic Sets

bols, etc. Also, symbol sets for selecting typestyles, weights, rotations, sizes, and elongations.

- A display screen for the microfiche; it is touch-sensitive and generates 7-bit codes according to location touched on the display.
- As an alternative, keyboard tops with fibre optic bundles molded in as a matrix, so that the keytops can be lighted with different symbols as selected.

Now imagine that we are writing an astrology book:

- Type

  Those of you born under the sign of Aries (

- Depress the "astro" key on the special keyboard.

Position	IRV 001	007	026	
0/09	HT	FO	FO	Format Control
0/11	VT	ECD	ECD	End (a typographical) CommanD
0/12	FF	SCD	SCD	Start (") CommanD
0/13	CR	QL	QL	Quad Left
0/14	SO	UR		Upper Rail
0/15	SI	LR		Lower Rail
1/01	DC1		Font 1	Change to normal
1/02	DC2		Font 2	Change to italic
1/03	DC3		Font 3	Change to bold
1/08	CAN	KW	KW	Kill Word (through previous space)
1/12	FS	SS	SS	SuperShift
1/13	GS	QC	QC	Quad Center
1/14	RS	QR	QR	Quad Right
1/15	US	JY	JY	JustifY

Figure 17. Registered Control Character Substitution

- Notice the shift in display for the fiche screen and/or the keytop lighting.
- Touch the Aries symbol on the screen (or the keytop).
- Depress SI (Shift In) on the special keyboard.
- And return to typing the rest of the sentence.

) will find this month...

Now imagine what a computer would do to the input stream in driving photocomposition equipment. The "astro" key generated an ESCape sequence for an astronomical graphic symbol set that would have been registered by AFNOR. When the input parser recognizes ESC, it analyzes the following characters, and then calls this set of character formation methods from the backup store, generates the character shape of Aries according to the character code after the final character, notices SI, and returns to normal mode.

Now we can envision how all of the world's printed material can be stored in machine-readable form, and interchanged recognizably!

ALTERNATE CONTROLS

Work has been in progress for several years to develop a companion standard for controls for devices such as CRT terminals. In the US this is contained in the ANSI document BSR X3.64, Additional Controls for Character Imaging. In a similar form, this C1 set was before the Codes Committee of ISO Technical Committee 97 (Computers and Information Processing) as document 2 N 868, for consideration at its 1978 May 24-26 meeting.

I had hoped to give the essence of this work here. There were only two negative votes in X3, which one could presume might be answered. Unfortunately, the work I had to do to compact the standard, trying to make it understandable, turned up more than unreadability. It turned up many logical flaws and ambiguities. So it's back to the drawing board, perhaps for a considerable period of time.

Figures 18a through 18e will give, however, some flavor of the controls under consideration.

Figure 18a shows the controls of Format Type (FT) 1 and 2. Format 1 is either the single character of the 8-bit set, shown in the first column as "Ce", or the 2-character sequence of the type "ESC Fe", where Fe is a final character taken from 4/00 to 5/15, and whose column designation is 4 less than Ce. I.e., in an 8-bit code, INDex would be 8/04. In a 7-bit code it would be ESC 4/04. Format 2 is of the type "ESC Fs", where Fs is a final taken from 6/00 to 7/14.

Figures 18b through 18e show controls with formats beginning with the control "CSI", defined in Figure 18a to be either 9/11 (in the 8-bit set) or "ESC [" (in the 7-bit sets). The six possible formats are:

3a = CSI Pn F	4a = CSI Pn I F
3b = CSI Pn ; Pn F	4b = CSI Pn ; Pn I F
3c = CSI Ps F	4c = CSI Ps I F

Pn stands for numeric parameter(s), Ps for a variable number of selective parameters separated by semicolons. The type 4 formats differ from type 3 only in inserting the intermediate character 2/00 just prior to the final.

In the figures, the parameter value enclosed in parentheses is the default value. That is, if the parameters are not actually inserted, i.e., being null, then the effect is the same as if the default value(s) were inserted.

To give an example of how these controls operate, look in Figure 18d for the second mnemonic, SGR (Select Graphic Rendition). It is represented first by CSI, the Control Sequencer Introducer, the parameter, and the final 6/13. This means that when the 4-character string

ESC [6 m

is encountered, it should turn on rapid blink in the field(s) specified on your video screen.

AL	=	Active Line (containing AP)
AP	=	Active Position (where the cursor is)
EF	=	Editor Function
FE	=	Format Effector
HT	=	Horizontal Tabulation
IN	=	INtroducer
PAD	=	Primary Auxiliary Device
RD	=	Received Datastream
SAD	=	Secondary Auxiliary Device
SD	=	String Delimiter
VT	=	Vertical Tabulation
QA	=	Qualified Area (defined by DAQ, SPA, EPA)
rfs	=	reserved for future standardization

Abbreviations for Figures 18a through 18e

Ce	FT	Type	Param	Mnem	Name
8/00-03	1				(rfs)
8/04	1	FE		IND	INDex
8/05	1	FE		NEL	NExt Line
8/06	1			SSA	Start of Selected Area
8/07	1			ESA	End of Selected Area
8/08	1	FE		HTS	Horizontal Tabulation Set
8/09	1	FE		HTJ	Horiz. Tabul. with Justification
8/10	1	FE		VTS	Vertical Tabulation Set
8/11	1	FE		PLD	Partial Line Down
8/12	1	FE		PLU	Partial Line Up
8/13	1	FE		RI	Reverse Index
8/14	1	IN		SS2	Single Shift 2
8/15	1	IN		SS3	Single Shift 3
9/00	1	SD		DCS	Device Control String
9/01	1			PU1	Private Use 1
9/02	1			PU2	Private Use 2
9/03	1			STS	Set Transmit State
9/04	1			CCH	Cancel CHaracter
9/05	1			MW	Message Waiting
9/06	1			SPA	Start of Protected Area
9/07	1			EPA	End of Protected Area
9/08-10	1				(rfs)
9/11	1	IN		CSI	Control Sequence Introducer
9/12	1	SD		ST	String Terminator
9/13	1	SD		OSC	Operating System Command
9/14	1	SD		PM	Privacy Message
9/15	1	SD		APC	Application Program Command

Fs	FT	Mnem	Name
6/00	2	DMI	Disable Manual Input
6/01	2	INT	INTerrupt
6/02	2	EMI	Enable Manual Interrupt
6/03	2	RIS	Reset to Initial State

Figure 18a. Controls for Character-Imaging Devices

Final	FT	Type	Param	Mnem	Name
4/00	3a	EF	(1)	ICH	Insert CHaracter
4/01	3a	EF	(1)	CUU	CUrsor Up
4/02	3a	EF	(1)	CUD	CUrsor Down
4/03	3a	EF	(1)	CUF	CUrsor Forward
4/04	3a	EF	(1)	CUB	CUrsor Backward
4/05	3a	EF	(1)	CNL	Cursor Next Line
4/06	3a	EF	(1)	CPL	Cursor Preceding Line
4/07	3a	EF	(1)	CHA	Cursor Horizontal Absolute
4/08	3b	EF	(1;1)	CUP	CUrsor Position
4/09	3a	EF	(1)	CHT	Cursor Horizontal Tabulation
4/10	3c	EF		ED	Erase in Display
			(0)		From AP to end (inclusive)
			1		From start to AP (inclusive)
			2		All of display

4/11	3c	EF		EL	Erase in Line
			(0)		From AP to end (inclusive)
			1		From start to AP (inclusive)
			2		All of line
4/12	3a	EF	(1)	IL	Insert Line
4/13	3a	EF	(1)	DL	Delete Line
4/14	3c	EF		EF	Erase in Field
			(0)		From AP to end (inclusive)
			1		From start to AP (inclusive)
			2		All of field
4/15	3c	EF		EA	Erase in Area
			(0)		From AP to end (inclusive)
			1		From start to AP (inclusive)
			2		All of QA
5/00	3a	EF	(1)	DCH	Delete CHaracter
5/01	3c			SEM	Select editing Extent Mode
			(0)		Edit in display
			1		Edit in AL
			2		Edit in field
			3		Edit in QA
5/02	3b		(1;1)	CPR	Cursor Position Report
5/03	3a	EF	(1)	SU	Scroll Up
5/04	3a	EF	(1)	SD	Scroll Down
5/05	3a	EF	(1)	NP	Next Page
5/06	3a	EF	(1)	PP	Preceding Page
5/07	3c	EF		CTC	Cursor Tabulation Control
			(0)		Set HT stop at AP
			1		Set VT stop at AL
			2		Clear HT stop at AP
			3		Clear VT stop at AL
			4		Clear all HT stops in AL
			5		Clear all HT stops in device
			6		Clear all VT stops in device
5/08	3a	EF	(1)	ECH	Erase CHaracter
5/09	3a	EF	(1)	CVT	Cursor Vertical Tabulation
5/10	3a	EF	(1)	CBT	Cursor Backward Tabulation

Figure 18b. Controls for Character-Imaging Devices

Final	FT	Type	Param	Mnem	Name
6/00	3a	FE	(1)	HPA	Horizontal Position Absolute
6/01	3a	FE	(1)	HPR	Horizontal Position Relative
6/02	3a		(1)	REP	REPeat
6/03	3a		(0)	DA	Device Attributes
6/04	3a	FE	(1)	VPA	Vertical Position Absolute
6/05	3a	FE	(1)	VPR	Vertical Position Relative
6/06	3b	FE	(1;1)	HVP	Horiz. and Vertical Position
6/07	3c	FE		TBC	TaBulation Clear
			(0)		Clear HT stop at AP
			1		Clear VT stop at AL
			2		Clear all HT stops in AL
			3		Clear all HT stops
			4		Clear all VT stops

6/08	3c			SM	Set Mode
			1	GATM	Guarded Area Transfer Mode
			2	KAM	Keyboard Action Mode
			3	CRM	Control Representation Mode
			4	IRM	Insertion-Replacement Mode
			5	SRTM	Status Reporting Transfer Mode
			6	ERM	ERasure Mode
			7	VEM	Vertical Editing Mode
			8		(rfs)
			9		(rfs)
			10	HEM	Horizontal Editing Mode
			11	PUM	Positioning Unit Mode
			12	SRM	Send-Receive Mode
			13	FEAM	Format Effector Action Mode
			14	FETM	Format Effector Transfer Mode
			15	MATM	Multiple Area Transfer Mode
			16	TTM	Transfer Termination Mode
			17	SATM	Selected Area Transfer Mode
			18	TSM	Tabulation Stop Mode
			19	EBM	Editing Boundary Mode
			20	LNM	Line feed New line Mode
6/09	3c			MC	Media Copy
			(0)		To PAD
			1		From PAD
			2		To SAD
			3		From SAD
			4		Turn OFF copying RD to PAD
			5		Turn ON copying RD to PAD
			6		Turn OFF copying RD to SAD
			7		Turn ON copying RD to SAD

Figure 18c. Controls for Character-Imaging Devices

Final	FT	Type	Param	Mnem	Name
6/10-11					(rfs)
6/12	3c			RM	Reset Mode (same parameters as SM)
6/13	3c	FE		SGR	Select Graphic Rendition
			(0)		Primary rendition
			1		Bold, or increased intensity
			2		Faint, decreased intensity, or secondary color
			3		Italic
			4		Underscore
			5		Slow blink (< 2.5/second)
			6		Rapid blink (> 2.5/second)
			7		Negative (reverse) image
			8		(rfs)
			9		(rfs)
			10		Primary Font
			11-19		1st to 9th alt. font (via FNT)
			20		Fraktur
6/14	3c			DSR	Device Status Report
			(0)		Ready, no malfunctions detected
			1		Busy - retry later
			2		Busy - DSR will notify ready

		3		Malfunction - retry
		4		Malfunction - DSR will notify ready
		5		Please report status (DSR or DSC)
		6		Please report AP via CPR
6/15	3c		DAQ	Define Area Qualification
		(0)		Accept all input
		1		Accept no input (protected); do not transmit (guarded)
		2		Accept graphics
		3		Accept numerics
		4		Accept alphabetics
		5		Right justify in area
		6		Zerofill in area
		7		HT stop at start of area (field)
		8		Accept no input (protected); permit transmit (unguarded)
		9		Spacefill in area

Figure 18d. Controls for Character-Imaging Devices

Final	FT	Type	Param	Mnem	Name
4/00	4a	EF	(1)	SL	Scroll Left
4/01	4a	EF	(1)	SR	Scroll Right
4/02	4b	FE	(100;100)	GSM	Graphic Size Modification
4/03	4a	FE		GSS	Graphic Size Selection
4/04	4b	FE	(0;0)	FNT	FoNT selection
			(0;0)		Primary font
			1;0		First alternative font
			...		...
			9;0		Ninth alternative font
4/05	4a	FE		TSS	Thin Space Specification
4/06	4c	FE		JFY	JustiFY
			(0)		Terminate all justify actions
			1		Fill action ON (text to/from other lines)
			2		Interword spacing
			3		Letter spacing
			4		Hyphenation
			5		Flush left margin
			6		Center text between margins
			7		Flush right margin
			8		Italian form (underscore last)
4/07	4b	FE		SPI	SPacing Increment
4/08	4c	FE		QUAD	Quad
			(0)		Flush left
			1		Flush left, fill with leader
			2		Center
			3		Center, fill with leader
			4		Flush right
			5		Flush right, fill with leader

Figure 18e. Controls for Character-Imaging Devices

CODE EXPANSION

We have seen how ASCII was *extended* by making many related pages of the 7-bit code. It is also possible to *expand* ASCII into an 8-bit code, or even 9-bit and 10-bit if we wished, for that matter. But an 8-bit code is obviously the most logical one to concentrate on, and this has been under development for several years.

The proposed 8-bit Expanded ASCII Code is shown in Figure 19. The identification of the graphic symbols is given in Table 6.

One can observe many interesting things about this set. For example, it has the entire Greek set of small letters except for "omicron," with eleven capitals to go with others from the Roman capitals to complete the Greek set. But apparently the committee didn't follow 646-031, the Greek alphabet mentioned in Table 5a. They didn't use the customary ordering "alpha-beta-gamma," the way we learn our "a-b-c's." I suppose it is argued that this set will never be used for language, only math symbols. And 646-027, shown in Figure 15a, does not demand the special capital "upsilon" shown in position 13/5. If the Greeks can agree to using a Roman capital "Y" for upsilon, could the Americans?

You'll notice some math symbols, but not enough for APL. In fact, the whole set seems highly slanted to mathematics, rather than business. Of course there are the four corner symbols for forms. Presumably the card suits will strike your eye, and you will wonder why so many other useful symbols were ignored in favor of these. Don't worry, they will always come in handy; it's sometimes useful to have symbols whose meaning you can reassign without harm to programming languages, etc. The committee were obviously bridge players, for spades collate high.

This proposal has not had real public scrutiny yet, and it must be considered no more than a proposal. Presumably X3 will agree about July that it should be sent out for public review and letter ballot. My guess is that it will not be adopted in just the form you see here.

FUTURE FOR ASCII

The methods are in place for codifying all symbols that people use. They may be language alphabets, signs, drawing symbols, or controls for equipments. Robots, for example. Satellites are augmenting conventional telecommunications systems, so that one can borrow cheaply and permanently from electronic libraries.

To prepare for this, other sets are being developed for registry, many through ISO Technical Committee 46/1, Automated Documentation. A 2-page mathematical symbol set is near submission, as are African sets. Work is started for Arabic, which will take about 5 sets to handle fully, although there is a commercial subset of 94 graphics. Another C1 set is being proposed for bibliographic controls. It contains four types—annotation controls, filing controls, reference controls, and subject designators. Other C1 sets can come from process control, animation and other graphics applications, etc.

Code	Symbol	Code	Symbol
10/00	(same as 02/00)	11/00	Large circle
10/01	Opening double quote	11/01	Dagger
10/02	Closing double quote	11/02	Superior (superscript) 2
10/03	Club suit	11/03	Superior (superscript) 3
10/04	Diamond suit	11/04	Rectangle
10/05	Heart suit	11/05	Parallel
10/06	Spade suit	11/06	Partial derivative
10/07	Closing single quote	11/07	Lower left corner, floor
10/08	Is implied by	11/08	Upper left corner, ceiling
10/09	Implies	11/09	Upper right corner
10/10	Multiply	11/10	Lower right corner
10/11	Plus or minus	11/11	Perpendicular
10/12	Nabla, or del	11/12	Less than or equal
10/13	Em dash	11/13	Not equal, other than
10/14	Radix point	11/14	Greater than or equal
10/15	Divide	11/15	Paragraph mark, pilcrow
12/00	Section mark	13/00	Capital pi
12/01	Double dagger	13/01	Capital psi
12/02	Dot bullet	13/02	Square bullet
12/03	Capital theta	13/03	Capital sigma
12/04	Capital delta	13/04	Integral
12/05	At least one exists	13/05	Capital upsilon
12/06	Capital phi	13/06	Therefore
12/07	Capital gamma	13/07	Capital omega
12/08	Upward arrow	13/08	Downward arrow
12/09	Right arrow	13/09	Left arrow
12/10	Dot product	13/10	Approximately equal
12/11	Degree	13/11	Opening angular bracket
12/12	Capital lambda	13/12	Logical AND
12/13	Register	13/13	Closing angular bracket
12/14	Copyright mark	13/14	Logical NOT
12/15	Capital xi	13/15	Infinity
14/00	Opening single quote	15/00	Small pi
14/01	Small alpha	15/01	Small psi
14/02	Small beta	15/02	Small rho
14/03	Small theta	15/03	Small sigma
14/04	Small delta	15/04	Small tau
14/05	Small epsilon	15/05	Small upsilon
14/06	Small phi	15/06	Check mark, radical mark
14/07	Small gamma	15/07	Small omega
14/08	Small eta	15/08	Small chi
14/09	Small iota	15/09	Logical universal quantifier
14/10	Identically equivalent	15/10	Small zeta
14/11	Small kappa	15/11	Cap intersection
14/12	Small lambda	15/12	Logical OR
14/13	Small mu	15/13	Cup, union
14/14	Small nu	15/14	Overbar
14/15	Small xi	15/15	(same as 7/15)

Table 6. Names of the Additional Graphics, 8-Bit Set

	08	09	10	11	12	13	14	15
0			▨	○	§	Π	‘	π
1			“	†	‡	Ψ	α	ψ
2			”	²	●	■	β	ρ
3			♣	³	Θ	Σ	θ	σ
4			♦	▭	Δ	∫	δ	τ
5		RESERVED FOR CONTROLS	♥	❘	∃	Υ	ϵ	υ
6			♠	∂	Φ	∴	ϕ	√
7			’	└	Γ	Ω	γ	ω
8			⊂	┌	↑	↓	η	χ
9			⊃	┐	→	←	ι	∨
10			×	┘	·	≅	≡	ζ
11			±	⊥	°	⟨	κ	∩
12			∇	<	∧	∧	λ	∨
13			–	≠	®	⟩	μ	∪
14			✦	>	©	¬	ν	‾
15			÷	¶	Ξ	∞	ξ	▨

Figure 19. 8-Bit ASCII Proposal

West Germany has proposed a new ISO Project on text communication, to harmonize teleconnection of the more than one hundred varieties of typewriters (and keyboards) throughout the world. The extension method of multiple 7-bit codes is ideal for this (8-bit codes imply too many keys or shift combinations for people to use easily).

I am convinced that microcomputer users are going to develop some fantastic applications that will become widespread enough for their special graphic and control sets to be registered. How about a control set or two for sewing machines?

In fact, it is very difficult ot think of any general application where one could not find a usage for these registered variants and extensions.

WHERE TO GET MORE INFORMATION

There are four sets of Information Processing Standards that may be of concern to you:

- ISO. Sold only through ANSI (American National Standards Institute), which has the franchise. That makes the prices high—much higher than in other countries.
- ANSI. These are American National Standards developed via the X3 and X4 committees, mostly. Prices still pretty high.
- ECMA (European Computer Manufacturers Association), 114 Rue du Rhone, 1204 Geneva, Switzerland). Free, and they have a lot more

advanced standards than ISO and ANSI. But a modest donation would not be unwelcome.

- Your friendly U.S. Government, in the person of the Department of Commerce, National Bureau of Standards, Institute for Computer Sciences and Technology, in Gaithersburg, MD 20760. If by any chance you are employed by the U.S. Government, you get FIPS PUBS (Federal Information Processing Standards Publications) for cheap. Otherwise, see ANSI. (Refer to Tables 1a, 1b, and 1c). In many cases they are essentially reprints of the ANSI standards, for a fraction of the cost.

If you can't wait for the standards to be approved and published, catch them in progress. Ask CBEMA, the sponsor of ANSI X3, to put you on an observer list for the committee in your area of interest. The address is:

Director of Standards
Computer & Business Equipment Manufacturers Association
1828 L Street NW
Washington, D.C. 20036
(202) 466-2288

REFERENCES

1. ANS X3.4-1977, available from the American National Standards Institute, 1430 Broadway, New York, NY 10018.
2. ISO 646, available from ANSI (Reference 1).
3. R.W. Bemer, "ASCII—the data alphabet that will endure," in *Management of data elements in information processing,* National Bureau of Standards, 1975 October, 17-22.
4. R.W. Bemer, "A view of the history of the ISO character code," Honeywell Computer J. 6, No. 4, 1972, 274-282.
5. E.H. Clamons, "Character codes: who needs them?", Honeywell Computer J. 5, No. 3, 1971, 143-146.
6. The TEX Subsystem of the Timesharing System, Series 60 Level 66, Honeywell Information Systems, 200 Smith Street, Waltham, MA 02154, Order DF72.

ACKNOWLEDGEMENTS

Thanks go to co-workers at Honeywell Information Systems: Eric Clamons for much background, insight, and experience gained from working for a long time as chairman of X3J2—the committee charged with the development of ASCII. And to Pat Skelly, ACM representative on ANSI X3, for collecting all the various national and international standards documentation upon which many of the figures were based.

FOOTNOTES

[1] For those curious about the reverse slash, it came from ALGOL 58. The reference language specified $\wedge$ and $\vee$ as the symbols for AND and OR respectively. I put the reverse slash in so these could be made as 2-character groups—/\ and \/.

[2] You will still see many terminals where this vertical bar is broken in the middle. This resulted from a hassle with the PL/I people, who wanted to stylize the exclamation point (2/1) as a vertical bar for OR in that language. And of

course that would make the graphics the same. The compromise (at horrendous cost in people time) was to break the real vertical bar in ASCII. But it turned out that the PL/I people didn't really need it, or else it gained no momentum, so the real vertical bar is back to normal in ASCII-1977. Let's fix those terminals.

[3] The Italians also have a different solution to hyphenation and right justification. It ignores the syllable structure and simply demands that if, when you get to the last position in the line, the current word is not yet completed, that last character shall be underscored, and the word continued without fuss on the next line. I rather like it.

[4] With the distribution, ECMA said "ECMA-53 is an attempt to improve portability of programs. It links the language character sets defined by the language standards, their coded representatives by means of the 7-bit code and the implementations on data carriers (punched tape, punched cards, magnetic tape and magnetic tape cassettes and cartridges). It is a standard of a new type in which already standardized features are assembled in a new standardized combination aimed at supporting interchange and decreasing implementation dependency."

[5] ISO 2530 is for the alphanumeric area of the keyboard only. It is augmented by ISO 3243-1975—Keyboards for Countries whose Languages have Alphabetic Extenders, Guidelines for Harmonizations, and also by ISO 3244-1974—Principles Governing the Positioning of Control Keys on Keyboards.

The fact that these are "guidelines" and "principles" indicate the complexity of the subject. Typewriter manufacturers now supply over a hundred different keyboard arrangements, as their catalogs will indicate.

Chapter 2

BASIC Cross Assembler for the 8080

By Peter Reece

INTRODUCTION

A cross assembler is a high level language program which assembles low level code for a goal computer. Since the host machine is usually much larger than the goal computer, code which is too long to be assembled in the smaller machine can be easily handled by the cross assembler. The binary tape which is produced can then be loaded directly into the goal computer.

CROSS

The cross assembler reproduced on the following pages is written in BASIC for a PDP-10 computer, and will accept and assemble code for 8080-based computers. It allows the user to produce ASCII listings in octal or HEX of his source code, define variables for a symbol table, and generate a binary file of the assembled source.

The program requires two passes (i.e., it reads the source code twice), hence random access files are not required. This permits the user with a smaller secondary memory, such as sequential cassette storage, to use the system. In addition, much less storage space for variables is required by the host computer.

The first pass scans each line of source looking for an OPCODE. If one is found, the address pointer a(i) is advanced an appropriate number of bytes. If a *comment only*, or an *equate* or *origin* line is found, a(i) is set equal to a(i-1), where 'i' corresponds to the previous statement number. Pass One also creates a symbol table of all labels and equates with user defined mnemonics.

Pass Two translates OPCODES and their arguments into octal, and outputs the results in both a binary file (as address, byte one, byte two, byte three) and an ASCII form (see example in Figure 1). All translation is initially done into octal. If all attempts at translation fail, byte one is set equal to '777' (i.e. an extra bit is flagged), the error

```
options:      ?hex    list  1f
      adrs     op      b2  b3
      00000                      /
      00000                      /Here's a simple do-nothing
      00000                      /example of the program.
      00000                      /
      00006                            *7#d
      00007    31      3D        start: lxi sp'61#d       /label line
      0000A    AF                       xra a
      0000B    47                       mov b'a      /"'" is a separator
      0000C    21      08        ok:    lxi h'load   /fwd ref
      0000F    48                       mov c'b
      00010    0E      24        write: mvi c'44#o   /octal num
      00012    21      08               lxi h'load
?     00015    **      48               xra j        /note error code
      00016    0E      0A        mvi c'$j            /alpha-num conversion
      00018    D5                       push d
      00019    76                end:   hit
                                        load=10#o    /equate line

start=07
ok=0C
write=10
end=19
load=08

          # ERRORS = 1
```

```
options:      ?octal  list  1f
      adrs     op      b2  b3
       0       0      0    0     /
       0       0       0   0     /Here's a simple do-nothing
       0       0       0   0     /example of the program.
       0       0       0   0     /
       6       0       0   0           *7#d
       7       61      75  0     start: lxi sp'61#d       /label line
       12      257     0   0            xra a
       13      107     0   0            mov b'a      /"'" is a separator
       14      41      10  0     ok:    lxi h'load   /fwd ref
       17      110     0   0            mov c'b
       20      16      44  0     write: mvi c'44#o   /octal num
       22      41      10  0            lxi h'load
?      25      1027    88  0            xra j        /note error code
       26      16      12  0     mvi c'$j            /alpha-num conversion
       30      325     0   0            push d
       31      166     0   0     end:   hit
                                        load=10#o    /equate line

start=7
ok=14
write=20
end=31
load=10

          # ERRORS = 1
```

Figure 1

Below is a simple example of the use of the program:

```
OPTIONS? octal-lf-nobin-list

 adrs      op     b2  b3
  0        0     0    0     /
  0        0     0    0     /
  0        0      0   0     /Here's a do-nothing
  0        0      0   0     /sample output from
  0        0      0   0     /the prog CROSS.
  0        0      0   0            *1
  1        61     0   4     start: lxi sp'stack          /set stack
? 4        777   100  0            lbi h'buffer          /set buffer
  7        176    0   0     strt1: mov a'm
                            /Note that CROSS automatically
                            /sets the spacing of the output.
                            /The tabs were not in the source.
  10       315    26  0            call write
  13       176    0   0            mov a'm               /finished?
  14       376   166  0            cpi 166#o
  16       302    22  0            jnz strt2             /nope
  21       166    0   0            hlt                   /yes
  22       16     11  0     strt2: mvi c'i$              /ascii char
  24       166    0   0            hlt
                            /
                            /
                            stack=2000#o
                            buffer=100#o
                            write=26
                            /
                            /
start=1
strt1=7
strt2=22
buffer=100
stack=2000
write=26

                        # ERRORS = 1

        options:  ?show
        /
        /Here's a simple do-nothing
        /example of the program.
        /
        *7#d
        start: lxi sp'61#d        /label line
        xra a
        mov b'a /"'" is a separator
        ok: lxi h'load   /fwd ref
        mov c'b
        write: mvi c'44#o         /octal num
        lxi h'load
        xra j   /note error code
        mvi c'$j          /alpha-num conversion
        push d
        end: hlt
        load=10#o         /equate line
        .end
        ox=32#d /equate line
        jmp write
        end: hlt
        .end
        options:  ?end
```

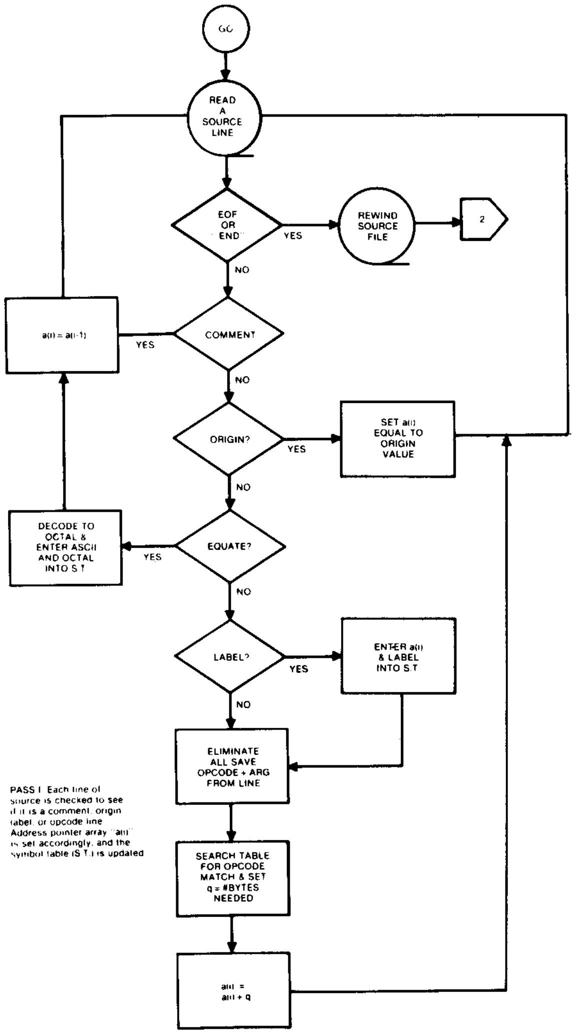
Figure 1 continued

count is increased by one, and Pass Two continues. If, during a listing, a '777' is encountered in byte one, a question mark is output beside the line being listed. The accompanying flow charts indicate the proceedings during the two passes. (See Figures 2 and 3)

Use of the program is straight forward. A number of options are available:

$	The character immediately preceeding the dollar sign in a line of source will be translated into octal, with the ASCII letter 'a' being translated as '001' and 'z' as '032'.
#	If followed by a 'd', the numeric preceding the '#' will be translated into octal from decimal; if followed by 'h', the translation will be from HEX to octal; if followed by 'o', the numeric will be assumed to be octal.
:	Any ASCII string preceding a colon will be assumed to be a label, and will be entered in the symbol table.
=	The right hand side of an equate will be translated into octal, and set equal to the left hand side as an entry in the symbol table.
/	A slash is followed by a comment.
*	An asterisk precedes a numeric expression which is to be used as an origin address.
?	Occurs beside all untranslatable lines of output.
HEX	If a user wishes output to be listed in HEX, type this when the program types 'Options:'.
OCTAL	(default condition)—output is listed in octal if this option is chosen.
LF	Choosing this option produces seven line feeds per 66 lines of output, thereby producing 66 lines per 9*11-inch page.
NOLF	Suppresses LF (default option).
LIST	Lists assembled code beside each line of source at the end of Pass Two (default option).
NOLIST	Suppresses 'list'.
BIN	This option produces an output file which may be read by the goal computer. For each line of source which is not a comment or an origin or equate line, 'BIN' produces the current address, byte one, byte two, byte three, and a carriage return (default option).

NOBIN	Supresses 'BIN'.
SHOW	Print the source listing; do not assemble it.
END	End the program.
MAKE	To create a source file, type "make" when the program starts, then type your source (one line of source per input line). Terminate "make" with the line "end".


GO
READ A SOURCE LINE
EOF OR END
YES
REWIND SOURCE FILE
2
NO
a(i) = a(i-1)
YES
COMMENT
NO
ORIGIN?
YES
SET a(i) EQUAL TO ORIGIN VALUE
NO
DECODE TO OCTAL & ENTER ASCII AND OCTAL INTO S.T.
YES
EQUATE?
NO
LABEL?
YES
ENTER a(i) & LABEL INTO S.T.
NO
ELIMINATE ALL SAVE OPCODE + ARG FROM LINE
SEARCH TABLE FOR OPCODE MATCH & SET q = #BYTES NEEDED
a(i) = a(i) + q

PASS I Each line of source is checked to see if it is a comment, origin label, or opcode line Address pointer array "a(i)" is set accordingly, and the symbol table (S.T.) is updated

Figure 2

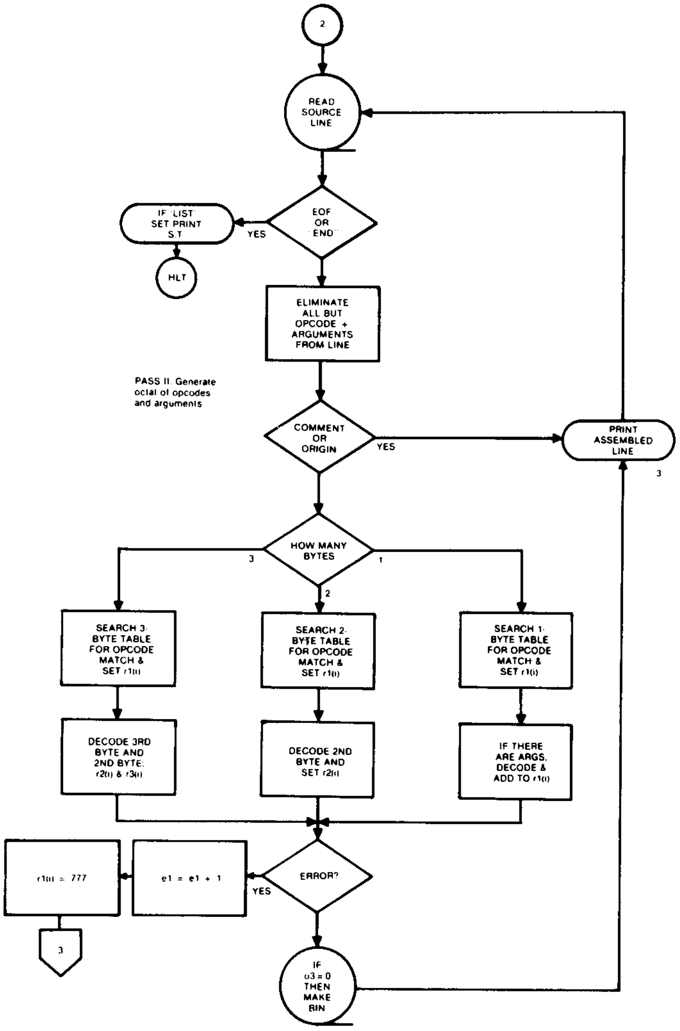

2
READ SOURCE LINE
EOF OR "END"
YES
IF "LIST" SET PRINT S.T
HLT
ELIMINATE ALL BUT OPCODE + ARGUMENTS FROM LINE
PASS II. Generate octal of opcodes and arguments
COMMENT OR ORIGIN
YES
PRINT ASSEMBLED LINE
3
HOW MANY BYTES
3
2
1
SEARCH 3-BYTE TABLE FOR OPCODE MATCH & SET r1(i)
SEARCH 2-BYTE TABLE FOR OPCODE MATCH & SET r1(i)
SEARCH 1-BYTE TABLE FOR OPCODE MATCH & SET r1(i)
DECODE 3RD BYTE AND 2ND BYTE: r2(i) & r3(i)
DECODE 2ND BYTE AND SET r2(i)
IF THERE ARE ARGS. DECODE & ADD TO r1(i)
ERROR?
YES
e1 = e1 + 1
r1(i) = 777
3
IF o3 = 0 THEN MAKE BIN

Figure 3

PROGRAM LISTING

```
10rem************************************************************
20rem*                                                          *
30rem*  A BASIC LANGUAGE CROSS-ASSEMBLER FOR THE INTEL 8080 UP  *
40rem*                                                          *
50rem*                   by P. Reece, 1977                      *
60rem*                                                          *
70rem************************************************************
```

```
80rem
90rem
100rem ******************
110rem * BASIC VERSION: *
120rem ******************
130rem PDP-10 Basic is used.  Functions are as follows:
140rem 1) n=instr(a$,b$).  This  returns n equal to the
150rem position of the 1st letter of a$ which matches the
160rem string b$.  Eg. n=instr("abcd","bc") returns n=2.
170rem If no match is found, n is returned as zero.
180rem 2) n$=mid$(a$,a,b).  N$ is returned equal to the substring
190rem of a$ begining with character a, and extending for b characters.
200rem Eg. n$=mid$("abcde",3,2) returns n$="de".
210rem 3) n$=chr$(a).  Converts the integer a to the ascii string
220rem n$. Eg. n$=chr$(56) returns n$="A" (PDP-10 ascii).
230rem 4) n$=str$(a).  Converts the integer a to its ascii equivelent.
240rem Eg. n$=str$(12) returns n$="12".
250rem 5)n=val(a$). Converts the integer ascii a$ to an integer.
260rem Eg., if a$="12", then n is returned equal to 12.
270rem NOTE: The array a(*) must be set equal to the number of statements
280rem in the source.  For most programs, this is generally not more than
290rem 500 statements, so a(500) is sufficient. If an output file is to be
300rem gnerated (via the 'bin' command), the dimensions on r1(*), r2(*), and
310rem r3(*) must also be set.  Otherwise, they may be left small if only a
320rem listing and assembly is desired.

330rem ************
340rem * COMMANDS *
350rem ************
360rem hex:     produces hex listing
370rem octal:   produces octal listing (default)
380rem lf:      prints 7 lf's every 66 lines of listing
390rem nolf:    cancels lf (default)
400rem nolist:  suppresses listing
410rem list:    gives listing (default)
420rem nobin:   suppresses output file
430rem bin:     creates output file (default)
440rem symbol:  prints symbol table of pass#1, then stops
450rem make:    create a source file
460rem .end:    termimate a source file (used during make)
470rem end:     terminate the program
480rem show:    list the unassembled source

490rem **********************
500rem * SPECIAL CHARACTERS *
510rem **********************
520rem $        the next char is ascii & will be translated to octal
530rem :        a label preceeds a colon
540rem #        follows a numeric
550rem d        follows a '#' if the numeric is decimal
560rem o        follows a '#' if the numeric is octal
570rem h        follows a '#' if the numeric is hex
580rem *        preceeds a numeric which is to be used as an origin
590rem /        preceeds a comment
600rem =        indicates an equate line of source
610rem ?        indicates an error in assemblage
620rem **       indicates a hex number too large for the listing
630rem 48       with a '?' for a hex listing, indicates the error byte

640rem 777      as above for '48', but for an octal listing
650rem***********************
660rem*     OPTIONS         *
670rem***********************
680rem Options in effect if switches = 1:
690rem 01:      hex listing
700rem o2:      suppress listing
710rem o3:      suppress binary output
720rem o4:      generate 7 LFs every 66 lines
730rem o5:      list symbol table, then stop
740rem o6:      'make' a source file
```

```
750rem*************************
760rem*     DEFINITIONS       *
770rem*************************
780rem r$(*)=   registers
790rem r1(*)=   byte #1
800rem r2(*)=   byte #2
810rem r3(*)=   byte #3
820rem a(*)=    decimal address of each source line
830rem s$(*)=   symbol table
840rem s1(*)=   octal of s$(*)
850rem m$(*)=   binary of dec 1 -> 7
860rem h2$(*)=  array of hex digits
870rem b1=      pointer for s$(*),s1(*)
880rem b=       pointer for a(*)
890rem e1=      number of assemble errors
900rem********************************
910rem There are two files - a source file, and an output file.
920rem The output file will contain the assembled octal of the source
930rem in the format: address-1st byte-2nd byte-3rd byte-carriage return.
940rem The address is 6 digits, the bytes are each 4 digits long.
950rem All files are sequentially read (i.e. no direct access is used)
960rem to ensure that non-disc systems can also use the program  (i.e.
970rem a cassette oriented Basic could also use the cross-assembler).
980rem*          INITIALIZE          *
990rem********************************
1000files s8080.bas$80,o8080.bas$80
1010dimr$(8),r1(100),r2(100),r3(100)
1020dima(200),s$(50),s1(50),m$(8)
1030dimh2$(16)
1040fori=0to9
1050h2$(i)=str$(i)
1060nexti
1070fori=10to15
1080h2$(i)=chr$(i+55)
1090nexti
1100e1=0
1110r$(0)="b"
1120r$(1)="c"
1130r$(2)="d"
1140r$(3)="e"
1150r$(4)="h"
1160r$(5)="l"
1170r$(6)="m"
1180r$(7)="a"
1190m$(0)="000"
1200m$(1)="001"
1210m$(2)="010"
1220m$(3)="011"
1230m$(4)="100"
1240m$(5)="101"
1250m$(6)="110"
1260m$(7)="111"
1270b=0
1280a(0)=0
1290b1=0
1300print"options: ";
1310inputc$
1320ifinstr(c$,"show")<>0then6750
1330ifinstr(c$,"nolist")=0then1350
1340o2=1
1350ifinstr(c$,"nobin")=0then1370
1360o3=1
1370ifinstr(c$,"lf")=0then1390
1380o4=1
1390ifinstr(c$,"symbol")=0then1410
1400o5=1
1410rem
1420print" "
1430ifinstr(c$,"hex")=0then1450
1440o1=1
```

```
1450ifc$="make"then6380
1460rem**********************************
1470rem* CREATE ADDRESSES & SYMBOLS     *
1480rem**********************************
1490rem
1500rem Pass one creates a symbol table, and
1510rem calculates all addresses for every
1520rem line of source.
1530rem
1540q=1
1550ifend:1then2290
1560input:1,c$
1570ifinstr(c$,".end")<>0then2290
1580rem comment line only?
1590n1=instr(c$,"/")
1600ifn1=0then1690
1610ifn1>3then1690
1620b=b+1
1630ifq<2then1660
1640a(b)=a(b-1)+q-1
1650goto1530
1660a(b)=a(b-1)
1670goto1530
1680rem origin line?
1690n=instr(c$,"*")
1700ifn=0then1810
1710c$=mid$(c$,n+1)
1720b=b+1
1730k$=c$
1740gosub4770
1750ifk=777then1770
1760c$=str$(k)
1770a(b)=val(c$)
1780a(b)=a(b)-1
1790goto1530
1800rem equate line?
1810n=instr(c$,"=")
1820ifn=0then1920
1830b1=b1+1
1840s$(b1)=mid$(c$,1,n-1)
1850c$=mid$(c$,n+1)
1860k$=c$
1870gosub4770
1880s1(b1)=k
1890b=b+1
1900a(b)=a(b-1)
1910goto1530
1920rem label line?
1930n2=instr(c$,":")
1940ifn2=0then2030
1950b=b+1
1960a(b)=a(b-1)+q
1970b1=b1+1
1980s$(b1)=mid$(c$,1,n2-1)
1990k=a(b)
2000gosub5340
2010s1(b1)=k
2020goto2060
2030rem opcode line? - get the number of bytes
2040b=b+1
2050a(b)=a(b-1)+q
2060ifn1>n2then2080
2070n1=80
2080k$=mid$(c$,n2+1,n1-1)
2090ifinstr(k$,"lxi")=0then2120
2100q=3
2110goto1550
2120rem
2130ifinstr(mid$(k$,1,3),"j")<>0then2100
2140ifinstr(k$,"pchl")<>0then2100
```

```
2150ifinstr(k$,"lda")<>0then2100
2160ifinstr(k$,"sta")<>0then2100
2170ifinstr(k$,"hld")<>0then2100
2180ifinstr(k$,"call")<>0then2100
2190ifinstr(k$,"out")=0then2220
2200q=2
2210goto1550
2220rem
2230ifinstr(k$,"inx")<>0then1530
2240ifinstr(k$,"cpi")<>0then2200
2250ifinstr(k$,"inr")<>0then1530
2260ifinstr(k$,"in")<>0then2200
2270ifinstr(k$,"i")<>0then2200
2280goto1530
2290rem
2300rem********************************
2310rem* DECODE OPCODES & VARIABLES   *
2320rem********************************
2330rem
2340rem Pass two decodes opcodes and their
2350rem variables.  The results are stored
2360rem in arrays representing the
2370rem first, second, and third bytes of
2380rem the source line.  The stored values
2390rem are octal.
2400ifo5=1then2840
2410b=0
2420printtab(4);"adrs";
2430printtab(12);"op";
2440printtab(17);"b2";
2450printtab(21);"b3"
2460restore:1
2470goto2510
2480ifend:1then2840
2490rem go print the result for this source line
2500gosub3060
2510input:1,c$
2520e$=c$
2530ifc$=".end"then2840
2540b=b+1
2550rem let k$= opcode + variables only
2560ifinstr(c$,"*")<>0then2480
2570ifinstr(c$,"=")<>0then2480
2580n=instr(c$,"/")
2590ifn=0then2620
2600ifn<6then2480
2610c$=mid$(c$.1.n-1)
2620n=instr(c$,":")
2630ifn=0then2650
2640c$=mid$(c$,n+1)
2650k$=c$
2660rem****************
2670rem* DECODE 'MOV'  *
2680rem****************
2690n=instr(c$,"mov")
2700ifn=0then3480
2710k$=mid$(c$,n+4,1)
2720q=n
2730gosub4630
2740ifk=777then2810
2750l=k
2760k$=mid$(c$,q+6,1)
2770gosub4630
2780ifk=777then2810
2790k$="1"+str$(l)+str$(k)
2800goto2820
2810k$="777"
2820r1(b)=val(k$)
2830goto2480
```

```
2840rem end of decode phase
2850rem print the symbol table
2860print" "
2870forq=1tob1
2880ifo1<>1then2940
2890k$=str$(s1(q))
2900gosub5820
2910gosub5950
2920prints$(q);"=";k$
2930goto2950
2940prints$(q);"=";s1(q)
2950nextq
2960print" "
2970ife1<>0then3000
2980printtab(10);"NO ERRORS"
2990goto3020
3000printtab(10);"# ERRORS = ";
3010printe1
3020stop
3030rem********************************
3040rem*  PRINT THE RESULTS           *
3050rem********************************
3060i=b
3070k=a(i)
3080ifk>0then3110
3090a(i)=0
3100goto3150
3110gosub5340
3120i=b
3130a(i)=k
3140ifa(i)=a(i-1)then3330
3150ifr1(i)>377then3170
3160ifr2(i)<400then3200
3170print"?";
3180e1=e1+1
3190r2(i)=88
3200ifo4<>1then3250
3210t1=t1+1
3220ift1<66then3250
3230t1=0
3240print" "
3250gosub6470
3260ifo2=1then3460
3270ifo1=1then6100
3280rem
3290printtab(4);a(i);
3300printtab(12);r1(i);
3310printtab(17);r2(i);
3320printtab(22);r3(i);
3330n=instr(e$,"/")
3340n1=instr(e$,":")
3350ifn1=0then3400
3360k$=mid$(e$,1,n1)
3370e$=mid$(e$,n1+2)
3380printtab(30);k$;
3390goto3420
3400ifn=0then3420
3410ifn<3then3450
3420printtab(37);e$
3430goto3460
3440printtab(30);e$
3450printtab(30);e$
3460return
3470rem***************
3480rem* DECODE ´MVI´ *
3490rem***************
3500n=instr(c$,"mvi")
3510ifn=0then3620
3520k$=mid$(c$,n+4,1)
```

```
3530u=n
3540gosub4630
3550r1(b)=15+k
3560n=u
3570k$=mid$(c$,n+6)
3580rem see if k$ is in s.t.
3590gosub4760
3600r2(b)=k
3610goto2480
3620rem*******************
3630rem* DECODE 2 BYTE OP *
3640rem*******************
3650restore
3660data in ,333,out ,323,adi,306,aci,316,sui,326
3670data sbi,336,ani,346,xri,356,ori,366,cpi,376
3680data end,0
3690readk$,k
3700ifk$="end"then3810
3710rem
3720n=instr(c$,k$)
3730ifn=0then3690
3740r1(b)=k
3750l=len(k$)
3760k$=mid$(c$,n+l)
3770c$=k$
3780gosub4760
3790r2(b)=k
3800goto2480
3810rem******************
3820rem* DECODE 'LXI'    *
3830rem******************
3840data lxi b,1,lxi d,21,lxi h,41,lxi sp,61
3850read k$,k
3860ifk$="end"then3980
3870n=instr(c$,k$)
3880ifn=0then3850
3890r1(b)=k
3900k$=c$
3910gosub4760
3920gosub5110
3930goto2480

3940data jnz,302,jz,312,jnz,322,jc,332,jpo,342,jpe,352
3950data jmp,303,jm,372,jp,362,cnz,304,cnc,324,cz,314
3960data cc,334,cpo,344,cpe,354,cp,364,cm ,374,call,315
3970data sta,062,lda,072,shld,042,lhld,052,end,0
3980rem***********************
3990rem* DECODE SINGLE BYTE   *
4000REM***********************
4010data rnz,300,rz,310,rnc,320,rc,330,rpo,340,rpe,350
4020data rp,360,rm,370,ret,311,rlc,7,rrc,17,ral,27,rar,37
4030data xchg,353,xthl,343,sphl,371,pchl,351,hlt,166,nop,0
4040data di,363,ei,373,daa,47,cma,57,stc,67,cmc,77,end,0
4050readk$,k
4060ifk$="end"then4100
4070ifinstr(c$,k$)=0then4050
4080r1(b)=k
4090goto2480
4100rem see if line is a one byte + register instr
4110data pop,301,push,305,stax,2,ldax,12
4120data inx,3,dcx,13,dad,11,end,0
4130readk$,k
4140ifk$="end"then4290
4150n=instr(c$,k$)
4160ifn=0then4130
4170l=len(k$)
4180k$=mid$(c$,n+l+1,1)
4190n=instr("bdh",k$)
4200ifn=0then4240
4210k=k+(n-1)*20
```

```
4220r1(b)=k
4230goto2480
4240ifk$<>"s"then4270
4250k=k+60
4260goto4220
4270ifk$="p"then4250
4280goto4530
4290rem decode final single byte opcode
4300data add,20,adc,21,sub,22,sbb,23
4310data ana,24,xra,25,ora,26,cmp,27,end,0
4320readk$,a
4330ifk$="end"then4410
4340n=instr(c$,k$)
4350ifn=0then4320
4360k$=mid$(c$,n+4,1)
4370gosub4630
4380r1(b)=a*10+k
4390goto2480
4400data inr,4,dcr,5,end,0
4410readk$,a
4420ifk$="end"then4500
4430n=instr(c$,k$)
4440ifn=0then4410
4450k$=mid$(c$,n+4,1)
4460gosub4630
4470ifk=777then4510
4480r1(b)=k*10+a
4490goto2480
4500rem*******************
4510rem*      ERROR       *
4520rem*******************
4530r1(b)=777
4540goto2480
4550rem
4560rem
4570rem
4580rem             SUBROUTINES
4590rem             -----------
4600rem
4610rem
4620rem
4630rem*************************
4640rem*  DECODE A REGISTER     *
4650rem*************************
4660rem enter with:     k$= register
4670rem exit with:      k= octal of register
4680rem                 k= 777 if error
4690k=777
4700fori=0to7
4710ifk$<>r$(i)then4740
4720k=i
4730i=7
4740nexti
4750return
4760rem************************
4770rem* DECODE  #,ASCII,S.T.  *
4780rem************************
4790rem enter with: k$ = string for decoding
4800rem exit with:  k = octal if s.t. entry
4810rem             k = octal if ascii entry
4820rem             k = octal if numeric entry
4830rem             k = 777 if none of the above
4840k=777
4850a$=k$
4860k$=c$
4870n=instr(k$,"'")
4880ifn=0then4900
4890k$=mid$(k$,n+1)
4900n=instr(k$,"#")
4910ifn=0then4980
```

```
4920rem reach here if a numeral existed
4930b$=mid$(k$,n+1,1)
4940k=val(mid$(k$,1,n-1))
4950ifb$="o"then5100
4960gosub5330
4970goto5100
4980rem see if k$ is ascii
4990n=instr(a$,"$")
5000if n=0then5040
5010k=instr("abcdefghijklmnopqrstuvwxyz",mid$(a$,n+1,1))
5020gosub5330
5030goto5100
5040rem see if a$ is in the symbol table
5050fori=1tob1
5060ifinstr(a$,s$(i))<>0then5090
5070nexti
5080goto5100
5090k=s1(i)
5100return
5110rem*************************
5120rem*   DECODE BYTE #3     *
5130rem*************************
5140rem enter with:  k = octal
5150rem exit with: r2(b) = octal if k<377
5160rem            : r3(b) = octal if k>377
5170r2(b)=k
5180ifk>377then5200
5190goto5320
5200rem convert k to a binary string
5210k$=str$(k)
5220gosub5460
5230h$=mid$(k$,1,8)
5240l$=mid$(k$,9)
5250rem convert h & l to octal
5260k$=h$
5270gosub5640
5280r3(b)=val(k$)
5290k$=l$
5300gosub5640
5310r2(b)=val(k$)
5320return
5330rem**********************************
5340rem* CONVERT DECIMAL K TO OCTAL K *
5350rem**********************************
5360a1=k
5370h$=""
5380fori=4to0by-1
5390a=int(a1/8**i)
5400k1=a1-a*8**i
5410a1=k1
5420h$=h$+str$(a)
5430nexti
5440k=val(h$)
5450return
5460rem**************************************
5470rem*   CONVERT OCTAL K$ TO BINARY K$   *
5480rem**************************************
5490h$=""
5500fori=1tolen(k$)
5510k2$=mid$(k$,i,1)
5520fork=0to7
5530ifstr$(k)<>k2$then5560
5540h$=h$+m$(k)
5550k=8
5560nextk
5570nexti
5580k$=h$
5590rem make k$ 16 characters long
5600fori=1to16-len(k$)
```

```
5610k$="0"+k$
5620nexti
5630return
5640rem**********************************
5650rem*   MAKE BINARY K$ OCTAL K$      *
5660rem**********************************
5670rem enter with:  k$ = 8-char binary
5680rem exit with:   k$ = octal equivilant
5690k$="0"+k$
5700k3$=""
5710fori=1to9step3
5720k2$=mid$(k$,i,3)
5730fork=0to7
5740ifinstr(k2$,m$(k))=0then5770
5750k3$=k3$+str$(k)
5760k=8
5770nextk
5780nexti
5790k$=k3$
5800return
5810rem*************************************
5820rem*  CONVERT OCTAL K$ TO DECIMAL K$   *
5830rem*************************************
5840k=0
5850l=len(k$)
5860l1=l
5870l=l+1
5880fori=1tol1
5890l=l-1
5900k=k+val(mid$(k$,l,1))*8**(i-1)
5910nexti
5920k$=str$(k)
5930return
5940rem*************************************
5950rem*  CONVERT DECIMAL K$ TO HEX K$     *
5960rem*************************************
5970a1=val(k$)
5980h1$=""
5990fori=n7to0step-1
6000k=int(a1/16**i)
6010k1=a1-k*16**i
6020ifk<17then6050
6030h1$="**"
6040goto6080
6050h1$=h1$+h2$(k)
6060a1=k1
6070nexti
6080k$=h1$
6090return
6100rem***********************
6110rem*  OUTPUT A HEX LINE   *
6120rem***********************
6130k$=str$(a(i))
6140n7=4
6150gosub5820
6160gosub5950
6170printtab(4);k$;
6180k$=str$(r1(b))
6190n7=1
6200gosub5820
6210gosub5950
6220ifk$<>"00"then6240
6230k$="  "
6240printtab(12);k$;
6250k$=str$(r2(b))
6260gosub5820
6270gosub5950
6280ifk$<>"00"then6300
6290k$="  "
```

```
6300printtab(17);k$;
6310k$=str$(r3(b))
6320gosub5820
6330gosub5950
6340ifk$<>"00"then6360
6350k$="  "
6360printtab(21);k$;
6370goto3330
6380rem********************************
6390rem*       MAKE COMMAND           *
6400rem********************************
6410inputc$
6420ifc$<>".end"then6450
6430print:1,c$
6440stop
6450print:1,c$
6460goto6410
6470rem***********************************
6480rem*    CREATE OCTAL OUTPUT FILE     *
6490rem***********************************
6500k$=str$(a(i))
6510gosub6640
6520k1$=k$
6530k$=str$(r1(i))
6540gosub6640
6550k1$=k1$+k$
6560k$=str$(r2(i))
6570gosub6640
6580k1$=k1$+k$
6590k$=str$(r3(i))
6600gosub6640
6610k1$=k1$+k$
6620print:2,k1$
6630return
6640rem***************************
6650rem* MAKE K$ = 3 CHARACTERS  *
6660REM***************************
6670n=len(k$)
6680ifn=3then6730
6690ifn=2then6720
6700k$="00"+k$
6710goto6730
6720k$="0"+k$
6730return
6740rem *******
6750rem *SHOW *
6760rem *******
6770ifend:1then6810
6780input:1,c$
6790printc$
6800goto6770
6810restore:1
6820goto1300
6830end

READY
```

Chapter 3

TLABEL: An 8080 Program to Punch Human-Readable Labels on Paper Tape

By Alan R. Miller, Contributing Editor

Have you ever discovered unlabeled tapes lying about and wondered what they were? Did you attempt to print them, only to find that they were punched in a binary or hexadecimal format? Would you like to have the file name and address in a form that you can read at the beginning of each tape? TLABEL can do that. TLABEL is an 8080 assembly-language program that can be used to punch human-readable messages on paper tape. It uses the set of 63 ASCII characters:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

0123456789

!“#$%&’()* +,-./:;<=>?@[\]Δ.

and a blank. These represent the ASCII values 20 to 5E HEX (040 to 135 octal).

TLABEL can be used two ways: either by itself or as a subroutine that can be called by a monitor, or by BASIC. To use TLABEL in the stand-alone mode, jump to the first instruction “START” at address 5E00 HEX. To use it as a subroutine, call ‘SUBR” at address 5E09 HEX. In the latter case, your calling program must provide four levels (8 bytes) of stack. In either case, a 5-inch leader is punched out when TLABEL is started.

Type the desired message, including any necessary blanks, then signal the end of the message by typing a Control-Z (all other control characters are ignored). Another five inches of blank tape will now appear and the program counter will jump to the address defined "MONIT" in the source program (zero in this case) if TLABEL is being used in the stand-alone version. Alternately, control will return to your calling program if TLABEL was called as a subroutine.

TLABEL can also be used to punch labels on BASIC source tapes. For use with MITS BASIC, answer the question "MEMORY SIZE?" during initialization with a value that will keep BASIC from plowing through TLABEL (24063 for TLABEL at 5E00 HEX). For MITS extended BASIC versions 4.0 and 4.1, put the following two lines at the end of your regular source program:

```
5000 DEFUSR = ?H 5E09: REM POINT USR TO TLABEL
5010 X = USR(9): LIST
```

Be sure that the statement prior to 5000 is an unconditional branch, such as a RETURN or GOTO, or is a STOP or END. Then give the direct command:

```
RUN 5000
```

The USR function will call TLABEL, allowing you to punch out the message on the tape leader. Type a Control-Z when finished and control will return to BASIC at the next expression after the USR command. Since this is LIST, the source program will then automatically be punched out.

Other versions of MITS BASIC should be changed to:

```
5000 US = 73: POKE US, 9: POKE US + 1,94
```

This command will patch USRLOC with the address of TLABEL's subroutine entry for MITS 8K, versions 3.2 and higher. For extended BASIC version 3.2, USRLOC is at 65 decimal. Therefore, the above statement should read US = 65.

In the 4K versions USRLOC is at 111 octal (49 HEX). However, a manual patch must be made since the POKE function is not available. Now only one line is needed:

```
5000 X = USR(9): LIST
```

The direct command is still:

```
RUN 5000
```

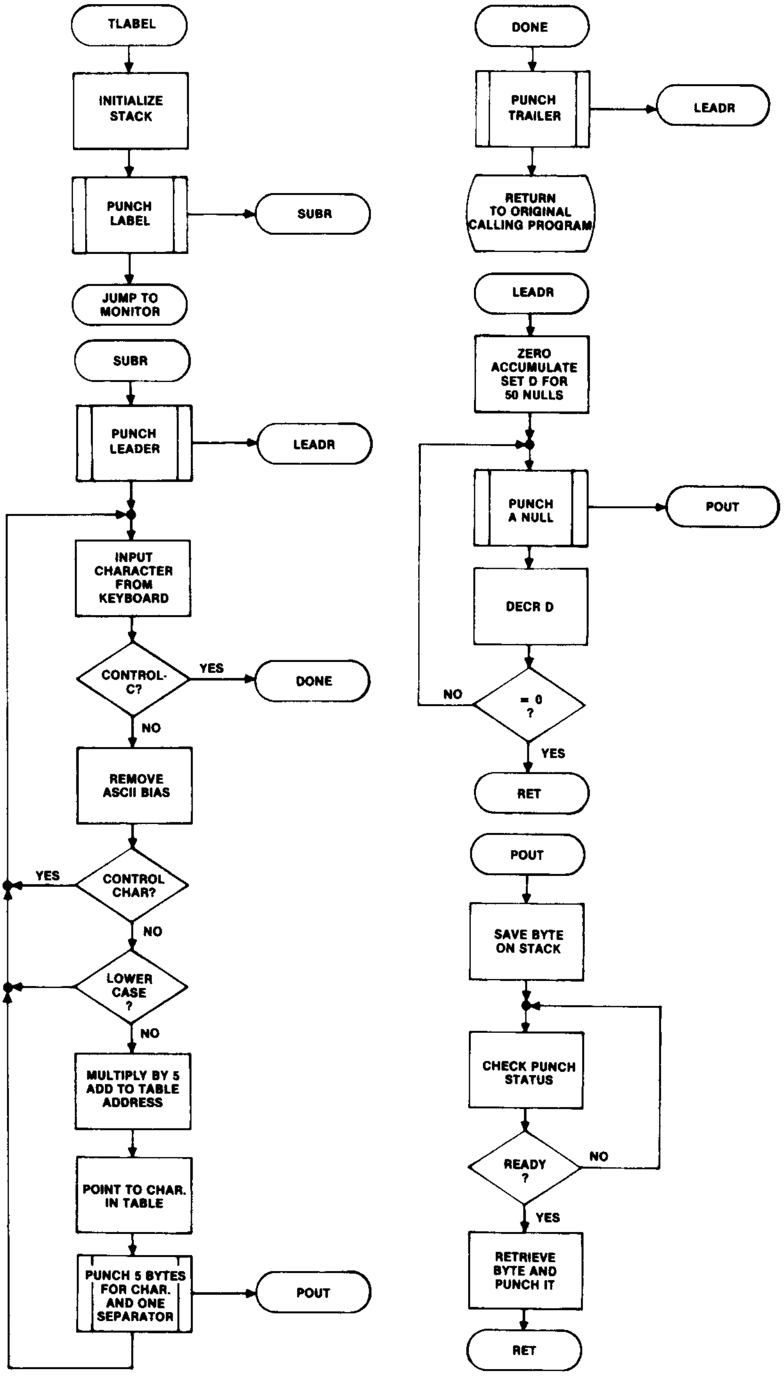
TLABEL requires 421 bytes including 8 bytes of stack. The HEX checksummed listing is assembled for the address range 5E00 to 5FA5. Keyboard input is at address 10/11 HEX (20/21 octal); in stand-alone mode, the separate punch is addressed to 12/13 HEX (22/23 octal). The keyboard-input address and the punch address can be the same (e.g. if a teletypewriter is used as the only peripheral) since the keyboard input is not echoed. The locations in Figure 1 may need to be changed for your system.

	Source Program Variable	Address (HEX)	Data (HEX)
Your monitor	MONIT	5E07,8	0000
Keyboard status addr	TYSTAT	5E0D	10
Keyboard data addr	TYDATA	5E14	11
Input-ready mask	INMSK	5E0F	01
Jump zero		5E10	CA
Punch status addr	PSTAT	5E57	12
Punch data addr	PDATA	5E5F	13
Output-ready mask	PMASK	5E59	02
Jump zero		5E5A	CA

Figure 1

PROGRAM LISTING

```
                ; TLABEL: PRØGRAM TØ MAKE HUMAN-READABLE
                ;         LABELS ØN ØN PAPER TAPE

                ; PRØGRAMMED FØR AN 8080 MICRØPRØCESSØR
                ; BY ALAN R. MILLER
                ; NEW MEXICØ TECH, SØCØRRØ, NM 87801
                ;  505-835-5619
                ; INTERFACE AGE, JANUARY 1979, PAGE 134

                        TITLE   'TAPE LABEL'

5E00            ØRG     5E00H
F800 =          MØNIT   EQU     0F800H  ;JMP TØ MØNITØR ØN ↑Z
0010 =          TYSTAT  EQU     10H     ;CØNSØLE STATUS
0011 =          TYDATA  EQU     TYSTAT+1 ;CØNSØLE DATA
0001 =          INMSK   EQU     1       ;INPUT MASK
0012 =          PSTAT   EQU     12H     ;PUNCH STATUS
0013 =          PDATA   EQU     PSTAT+1 ;PUNCH DATA
0002 =          PMASK   EQU     2       ;PUNCH-READY MASK

                ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

                ; ENTER AT THIS PØINT WHEN USED AS MAIN PRØGRAM

5E00 31A25F     START:  LXI     SP,STACK  ;STACK AT END
5E03 CD095E             CALL    SUBR
5E06 C300F8             JMP     MØNIT   ;WHEN DØNE
                ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

                ; CALL AT THIS PØINT FØR USE AS A SUBRØUTINE

5E09 CD485E     SUBR:   CALL    LEADR   ;PUNCH A LEADER

                ; INPUT A CHARACTER FRØM THE KEYBØARD

5E0C DB10       READ:   IN      TYSTAT  ;GET STATUS
5E0E E601               ANI     INMSK   ;MASK UNWANTED BITS
5E10 CA0C5E             JZ      READ    ;LØØP UNTIL READY
5E13 DB11               IN      TYDATA  ;GET DATA
5E15 E67F               ANI     7FH     ;STRIP PARITY
5E17 FE1A               CPI     1AH     ;QUIT ØN CØNTRØL-Z
5E19 CA485E             JZ      DØNE
5E1C DE20               SBI     20H     ;REMØVE ASCII BIAS
```


TLABEL
INITIALIZE STACK
PUNCH LABEL
SUBR
JUMP TO MONITOR
SUBR
PUNCH LEADER
LEADR
INPUT CHARACTER FROM KEYBOARD
CONTROL-C?
YES
DONE
NO
REMOVE ASCII BIAS
YES
CONTROL CHAR?
NO
LOWER CASE ?
NO
MULTIPLY BY 5 ADD TO TABLE ADDRESS
POINT TO CHAR. IN TABLE
PUNCH 5 BYTES FOR CHAR. AND ONE SEPARATOR
POUT
DONE
PUNCH TRAILER
LEADR
RETURN TO ORIGINAL CALLING PROGRAM
LEADR
ZERO ACCUMULATE SET D FOR 50 NULLS
PUNCH A NULL
POUT
DECR D
NO
= 0 ?
YES
RET
POUT
SAVE BYTE ON STACK
CHECK PUNCH STATUS
READY ?
NO
YES
RETRIEVE BYTE AND PUNCH IT
RET

```
5E1E DA0C5E              JC      READ     ;SKIP CØNTRØL CHARACTER
5E21 FE40                CPI     40H
5E23 DA285E              JC      RD2      ;UPPER CASE
5E26 DE20                SBI     20H      ;MAKE UPPER CASE
5E28 6F         RD2:     MØV     L,A      ;SAVE CHARACTER IN L
5E29 5F                  MØV     E,A      ; AND E
5E2A 2600                MVI     H,0      ;ZERØ H
5E2C 1600                MVI     D,0      ; AND D

                ; FIND THE TABLE ØFFSET BY MULTIPLYING THE
                ; CHARACTER VALUE BY FIVE (5 PUNCHES PER
                ; CHARACTER) AND ADDING IT TØ THE TABLE ADDRESS

5E2E 29                  DAD     H        ;DØUBLE THE CHAR. VALUE
5E2F 29                  DAD     H        ;THEN QUADRUPLE IT
5E30 19                  DAD     D        ;NØW TIMES FIVE
5E31 EB                  XCHG             ;SAVE IT IN D/E
5E32 215F5E              LXI     H,TABLE  ;PØINT TØ TABLE
5E35 19                  DAD     D        ;ADD ØFFSET
5E36 1E05                MVI     E,5      ;5 PUNCHES PER CHAR.
5E38 7E         NEXTC:   MØV     A,M      ;FETCH PUNCH CØDE
5E39 CD535E              CALL    PØUT     ;PUNCH IT
5E3C 23                  INX     H        ;INCREMENT PØINTER TØ NEXT
5E3D 1D                  DCR     E        ;DECREMENT CØUNT
5E3E C2385E              JNZ     NEXTC
5E41 AF                  XRA     A        ;PUNCH A BLANK
5E42 CD535E              CALL    PØUT     ; BETWEEN EACH CHARACTER
5E45 C30C5E              JMP     READ     ;NEXT CHARACTER

                ; FINISHED, PUNCH TRAILER
                ; AND RETURN TØ CALLING PRØGRAM

                DØNE:

                ; SUBRØUTINE TØ PUNCH A LEADER ØN TAPE

5E48 AF         LEADR:   XRA     A        ;SET FØR NULL
5E49 1632                MVI     D,50     ;NUMBER ØF NULLS
5E4B CD535E     NLDR:    CALL    PØUT     ;PUNCH A BLANK
5E4E 15                  DCR     D
5E4F C24B5E              JNZ     NLDR
5E52 C9                  RET
                ;
                ; SUBRØUTINE TØ ØUTPUT DATA TØ PUNCH
                ;
5E53 F5         PØUT:    PUSH    PSW
5E54 DB12       PØUTW:   IN      PSTAT
5E56 E602                ANI     PMASK    ;MASK UNWANTED BITS
5E58 CA545E              JZ      PØUTW    ;LØØP UNTIL READY
5E5B F1                  PØP     PSW
5E5C D313                ØUT     PDATA    ;PUNCH BYTE
5E5E C9                  RET
                ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

                ; TABLE ØF PUNCH SYMBØLS FØR THE CHARACTER SET

5E5F 0000000000TABLE:    DB      0,0,0,0,0          ; BLANK
5E64 0000CFCF00          DB      0,0,207,207,0 ; EXCLAIM
5E69 0007000700          DB      0,7,0,7,0, 40,254,40,254,40 ; ",#
5E73 4689FF8972          DB      70,137,255,137,114 ; $
5E78 462610C8C4          DE      70,38,16,200,196 ; %
5E7D 6C92AC40A0          DB      108,146,172,64,160, 0,4,3,3,0 ; & '
5E87 003C428100          DB      0,60,66,129,0, 0,129,66,60,0 ; ( )
5E91 8850F85088          DB      136,80,248,80,136, 8,8,126,8,8 ; *,+
5E9B 0080703000          DB      0,128,112,48,0, 8,8,8,8,8 ; , -
5EA5 00C0C00000          DB      0,192,192,0,0, 64,32,16,8,4 ; . /
5EAF 7EA189857E          DB      126,161,137,133,126 ; 0
5EB4 8482FF8080          DB      132,130,255,128,128 ; 1
5EB9 C2A1918986          DE      194,161,145,137,134 ; 2
5EBE 4289898976          DB      66,137,137,137,118 ; 3
```

```
5EC3 0C0A89FF88          DB      12,10,137,255,136 ;  4
5EC8 6789898971          DB      103,137,137,137,113 ;  5
5ECD 7E89898972          DB      126,137,137,137,114      ;  6
5ED2 0101F90503          DB      1,1,249,5,3 ;7
5ED7 7689898976          DB      118,137,137,137,118 ;  8
5EDC 468989897E          DB      70,137,137,137,126 ;  9
5EE1 00D8D80000          DB      0,216,216,0,0,  0,128,118,54,0 ;  : ;
5EEB 1028448200          DB      16,40,68,130,0,  40,40,40,40,40 ;  < =
5EF5 8244281000          DB      130,68,40,16,0,  6,1,185,9,6 ;  > ?
5EFF 7E819D910E          DB      126,129,157,145,14 ;  @
5F04 FE090909FE          DB      254,9,9,9,254 ;  A
5F09 81FF898976          DB      129,255,137,137,118 ;  B
5F0E 7E81818142          DB      126,129,129,129,66 ;  C
5F13 81FF81817E          DB      129,255,129,129,126 ;  D
5F18 FF89898989          DB      255,137,137,137,137 ;E
5F1D FF09090901          DB      255,9,9,9,1 ;F
5F22 7E81919172          DB      126,129,145,145,114 ;  G
5F27 FF080808FF          DB      255,8,8,8,255 ;H
5F2C 0081FF8100          DB      0,129,255,129,0 ;I
5F31 6080817F01          DB      96,128,129,127,1 ;J
5F36 FF081422C1          DB      255,8,20,34,193 ;  K
5F3B FF80808080          DB      255,128,128,128,128 ;  L
5F40 FF020C02FF          DB      255,2,12,2,255,  255,2,60,64,255 ;  M,N
5F4A FF818181FF          DB      255,129,129,129,255,  255,9,9,9,6 ;  O,P
5F54 7E81A141BE          DB      126,129,161,65,190 ;  P,Q
5F59 FF19294986          DB      255,25,41,73,134 ;  R
5F5E 4689898972          DB      70,137,137,137,114 ;  S
5F63 0101FF0101          DB      1,1,255,1,1,  127,128,128,128,127 ;  T,U
5F6D 0F30C0300F          DB      15,48,192,48,15 ;  V
5F72 7F8070807F          DB      127,128,112,128,127 ;  W
5F77 C3241824C3          DB      195,36,24,36,195,  3,4,248,4,3 ;  X,Y
5F81 C1A1918987          DB      193,161,145,137,135 ;  Z
5F86 00FF818181          DB      0,255,129,129,129,  4,8,16,32,64 ;[ \
5F90 818181FF00          DB      129,129,129,255,0,  12,2,1,2,12 ;] ↑

5F9A                     DS      8          ;STACK SPACE
5FA2 =          STACK    EQU     $

5FA2                     END
```

Chapter 4

TAPEMON: An 8080 Binary Tape Monitor

By Alan R. Miller

TAPEMON Is an 8080 Assembly Language program that can be used to dump, load, and verify checksummed tapes. The Intel HEX checksum format is commonly used to save relatively short programs on paper tape because the resulting tapes can be read directly when fed into a teletype. For longer programs, especially those saved on magnetic tape or disc, a binary format is more suitable. While the HEX format requires two types on tape to represent each byte of memory (only the characters 0-9 and A-F are used), the binary format needs only one byte on tape for each memory byte. As a consequence binary tapes load in half the time required for HEX format tape. For example, a 12K BASIC interpreter on paper tape can be loaded with a teletype in 20 minutes if the object tape is in a binary format. The same program would require 40 minutes to load if punched in the HEX format.

There appears to be no standard binary format currently in use. Some methods use a separate checksum for the record address and another for the data. Others, such as the program presented here, use only one checksum for each record. Files produced with TAPEMON consist of a leader, a file header record, one or more data records, an end-of-file record, and a trailer. The format is:

FILEHEADER	55H	filename , comment CR			
SHORT RECORD	3CH	rec len addr L,H	data	checksum	
LONG RECORD	77H	rec len L,H	addr L,H	data	checksum
END-OF-FILE	74H	autostart addr L,H	checksum		

The 55 HEX (125 OCTAL) byte at the beginning of the file header signals the beginning of the file. An optional file name of up to eight characters may follow. This file name may contain any ASCII print-

able character except a comma. The file name can be optionally followed by a comma and a comment of up to eight ASCII Printable characters. A carriage return terminates the header record.

Data records consist of a record header byte, a 1- or 2-byte record length, a 2-byte record address, the data byte and the checksum byte. There are two types of data records: short and long. Short records contain less than 256 data bytes; long records contain 256 or more data bytes. Data records start either with a 3C HEX (74 OCTAL) byte for short records, or with a 77 HEX (167 OCTAL) byte for the long records. The recorder-header byte is followed by a 1- or 2-byte record length, which gives the number of data bytes in the record. A single byte is used for short records and two bytes (least significant byte first) are used for long records.

The record address follows the record length. It consists of two bytes (least significant byte first) and gives the location where the first data byte of the record is to be stored. The data bytes appear next in binary form, one byte of record for each data byte. A checksum byte, obtained by adding without carry the record addresses (two bytes) and the data bytes, terminates the record. The record length is not included in the checksum. This is not necessary since if the record-length byte is incorrectly read, the byte which is incorrectly calculated to be the checksum byte will be the wrong one.

A 4-byte end-of-file record appears after the last data record. The first byte of this record is a 74 HEX (164 OCTAL), the next two bytes are the autostart address (least significant byte first), the address where the program is to begin. The fourth byte is a checksum for the two-byte autostart address.

The user initializes the HEXMON by starting at the beginning of the program (the label START). This produces the statement:

HEX OR OCTAL INPUT?

Type an "O" and a carriage return if you want to enter addresses and data in octal format. HEX-input mode is selected by typing an "H" and a carriage return. HEX-input mode will also be selected by default if just a carriage return is typed. The program then prints accordingly OCTAL INPUT or HEX INPUT and then asks:

RECORD LENGTH?

Enter a 2-byte record length in the previously selected HEX or OCTAL mode (6 OCTAL digits or 4 HEX digits) and a carriage return. Typing just a carriage return will select the default record length of 255 bytes (377 OCTAL, FF HEX). In this latter case, the computer responds with the statement RECORD LENGTH 255.

The printing of the prompt ">:" indicates that the main portion of the program starting at the label RESTRT has been reached. The valid commands are "M", "R", "N", "G", "D", "L", "E", "V", "O", and "C". If an error is made while entering the task or the addresses that follow, type a Control-X and this portion of the program will be restarted. At the completion of any of these tasks, control will return to this point with a reprinting of the prompt">:". The current mode (OCTAL or HEX input) and the record length can be determined by typing an "M" (MODE). The input mode and record length can be reset by typing an "R" (RESET).

Typing an "N" and one HEX or OCTAL byte will reset the number of NULLS that precede and follow a dump. The default value is one which is satisfactory for magnetic tape. The value should, however, be reset to 48 HEX (110 OCTAL) for paper tape. This will produce a blank 6-inch leader and trailer.

Enter a "G", a 2-byte address, and a carriage return to go somewhere else, for example to your regular monitor. Typing a Control-X during input will cancel the line and restart this program. A "WHAT?" will be printed for improper input (e.g., HEX characters when in OCTAL Mode).

A portion of memory can be dumped to tape by typing a "D" (for DUMP), the start address, the stop address, and autostart address (two bytes for each address), and a carriage return. An optional file name of up to eight characters can be typed after the autostart address and before the carriage return. Any printable ASCII character except a comma can be used. If a file name is entered, then an optional comment of up to eight characters can also be used. The comment can contain any printable ASCII character, and is separated from the file name by a comma. For example:

```
>:f800:F91F:F803:PROM4,VER 4.1<CR>
```

will dump from F800 to F91F with an autostart address of F803. The header will carry the file name/comment PROM,VER4.1. (The colons and > symbol are printed by the program).

If an error is made during entry of the file name or comment, type a DEL (RUB OUT) and then the correct character. A back arrow is echoed when the DEL key is pressed.

A tape can be loaded by typing an "L" and a carriage return. The keyboard bell will ring and the front panel lights (if you have them) will change when the file header (55 HEX) is found. This feature requires six additional bytes and an additional tape-input routine, one is used to look for the file header and the other is used for everything else. I feel that the additional complexity is worth it since I don't have to wait until a tape has been played through to find out that I have the left channel plugged into the computer, but the program I want is on the right channel.

When the tape has successfully loaded, the autostart address will be printed in both HEX and OCTAL. The prompt ">:" will be printed indicating that the program is ready for the next task. Another way to load a tape is to type the file name after the "L" command, e.g.

```
>:LPROM4<CR>
```

can be used to load the tape made in the above example. The DEL key can be used to correct errors made while entering the file name. If the file name in the command line does not match the file name on the tape, the task is terminated. The program prints an error statement followed by the actual file name and comment, e.g.

```
WRONG FILE NAME, TRY: PROM4,VER 4.1
```

would be printed if the incorrect file name LPROM3 were given. If the program loads correctly, the file name, comments and an autostart address are printed. For example:

```
PROM4,VER 4.1 STARTS AT F803:370003
```

would appear in this sample.

A program can be loaded and executed by typing an "E", optionally a file name and a carriage return. When the program has been loaded, the program counter will jump to the autostart address.

A tape can be loaded at other than its normal address by typing an "O" (for offset), a 2-byte offset address (in the current OCTAL or HEX mode), optionally the file name and a carriage return. An offset of 0400 HEX (004000 OCTAL) will load the program 1K bytes (1024 bytes) higher than the normal address. The program can be loaded 4K bytes lower than the normal address with an offset of either F000 HEX (360000 OCTAL) or −1000 HEX (−020000 OCTAL). If for example an offset of F000 is added to the program address of 3000, the double register add gives an address of 2000. A negative offset value is first subtracted from zero and then added to the program address. During the offset load, the original address is added to the checksum so that it is properly calculated. Of course the jumps and calls are not altered, so that the program will not run at the new location if there are jumps and calls addressed for original location.

MITS software such as BASIC and the Software Package II assembler is provided on paper or magnetic tape in a binary format that is compatible with TAPEMON. This software, however, also contains a reverse-loaded checksum loader ahead of the main program. All MITS programs can be loaded with the command "C" (for checksum) and a carriage return. The checksum loader at the beginning of the tape is scanned for the disable interrupt command (DI). This DI command is the last byte of the checksum loader and now represents the file header. The record header byte of 3C HEX following the checksum loader is then searched for. On the other hand, absolute tapes made with the MITS Software Package II monitor itself are fully compatible with TAPEMON and can be loaded with the "L", "E", or "O" commands. MITS does not use a checksum on the end-of-file record, but since the record header is a 78 HEX (170 OCTAL) rather than a 74 HEX, TAPEMON can distinguish between the two types of EOF records.

Tapes loaded with TAPEMON do not have to be verified since they are checksummed. If the load operation was completed, the tape was loaded correctly. Tapes dumped with TAPEMON of course should be verified to be certain that they were properly recorded. A defective spot on the tape for example may give an error. Tapes can be verified by playing back the tape, typing "V", optionally the file name, and a carriage return.

For all of the above load and verify operations, the two record-address bytes and the data bytes are summed and compared to the checksum value at the end of each record. If the two do not match, the operation is terminated, and the message:

CHECKSUM ERROR AT F801:370008

is printed giving the value of the H,L register pair at the checksum. This will not usually be the location of the error, but can be useful in deciding whether the error is in the tape or in the interface circuitry. If a second load or verify gives the same address, it is likely that the tape is the problem, whereas if the address is different the next time, the fault may lie with the hardware. During the load operations, each memory location is immediately read back after a deposit to be certain that the value in memory is correct. Attempting to load into non-existent, protected, or defective memory will terminate the load and error message:

MEMORY ERROR AT F820:370040

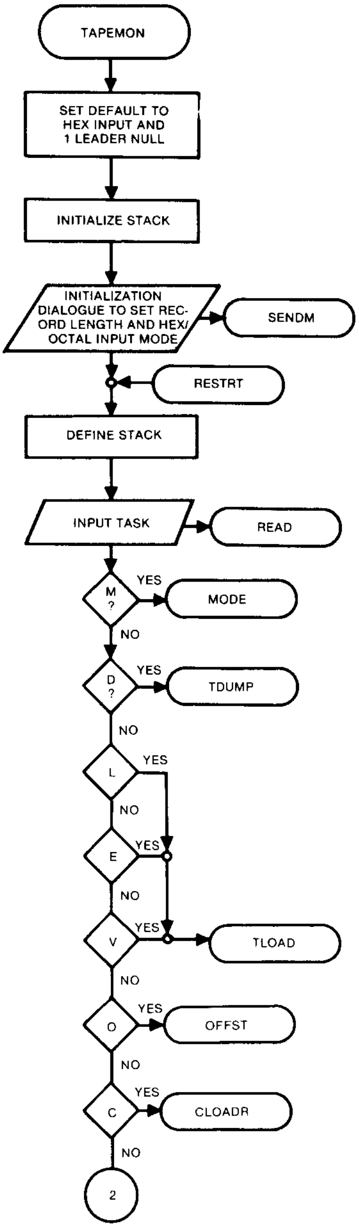
will be printed.

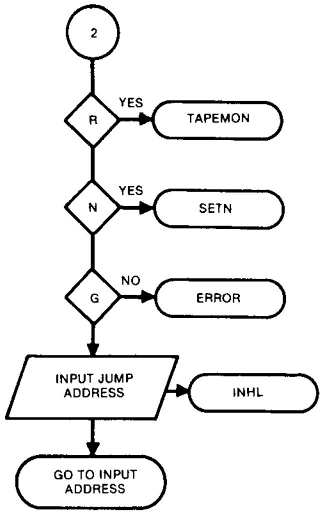
TAPEMON can be used to punch binary paper tapes. The resulting garbage on the printer during the dump will of course be meaningless and the printer will make funny noises. When the tape is read back, however, all is quiet, since the input is not echoed. The NULL command should be used to set 48 HEX (110 OCTAL) nulls to provide a leader and trailer of six inches. If paper tape is the usual medium, change the default option to provide 64 nulls during initialization.

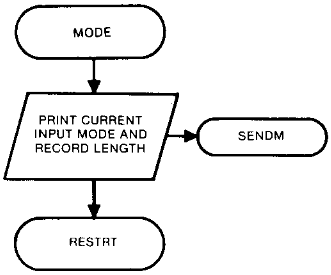
TAPEMON requires 1523 bytes of memory, including twelve levels (24 bytes) of stack. The program is written for the standard MITS configuration of a 2SIO serial port addressed to 10/11 HEX (20/21 OCTAL) and a tape recorder interface addressed to 6/7. The following table gives the locations and parameters that may need to be changed for your system.

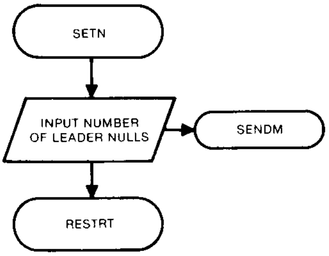
	SOURCE PROGRAM VARIABLE	**ADDRESS (HEX)**	**DATA (HEX)**
Define stack	STACK	580D,5894	5DEA
Keyboard status	TYSTAT	592D,595D	10
Keyboard data	TYDATA	5934,5965	11
Mask for data avail.	INMASK	592F	01
Mask for output	OUTMSK	595F	02
Jump zero		5930,5960	CA
Tape status	TAPES	5B1F,5C44,5C55	06
Tape data	TAPED	5B26,5C4B,5C5D	07
Mask for data avail.	ACINM	5B21,5C46	01
Mask for tape output	ACOM	5C57	80
Jump not zero		5B22,5C47,5C58	C2
Default record len	RLEN	587A	00FF
Default leader nulls	SNUL	5808	01

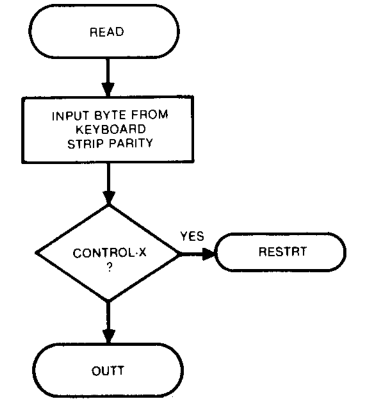
FLOWCHARTS

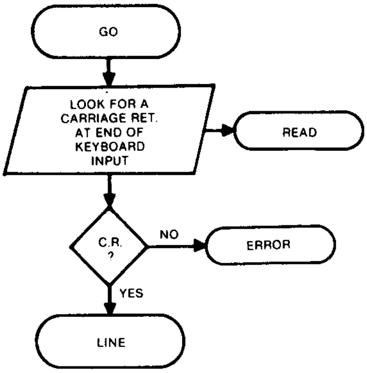

TAPEMON
SET DEFAULT TO HEX INPUT AND 1 LEADER NULL
INITIALIZE STACK
INITIALIZATION DIALOGUE TO SET RECORD LENGTH AND HEX/OCTAL INPUT MODE
SENDM
RESTRT
DEFINE STACK
INPUT TASK
READ
M ?
YES
MODE
NO
D ?
YES
TDUMP
NO
L
YES
NO
E
YES
NO
V
YES
TLOAD
NO
O
YES
OFFST
NO
C
YES
CLOADR
NO
2

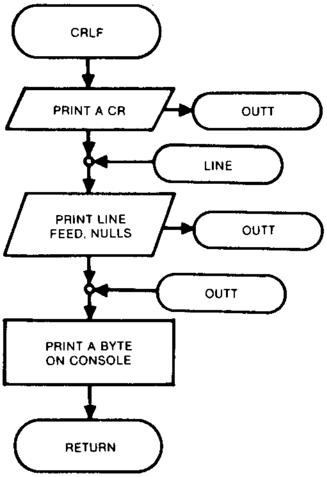

2
R
YES
TAPEMON
N
YES
SETN
G
NO
ERROR
INPUT JUMP ADDRESS
INHL
GO TO INPUT ADDRESS

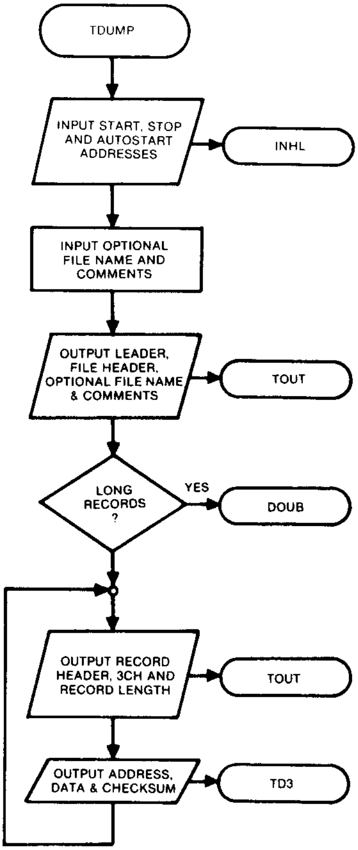

MODE
PRINT CURRENT INPUT MODE AND RECORD LENGTH
SENDM
RESTRT

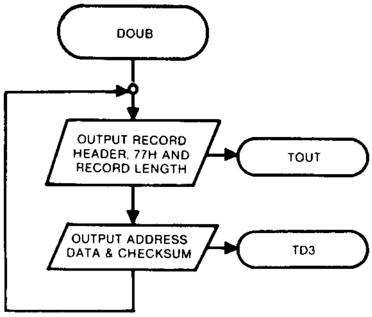

SETN
INPUT NUMBER OF LEADER NULLS
SENDM
RESTRT

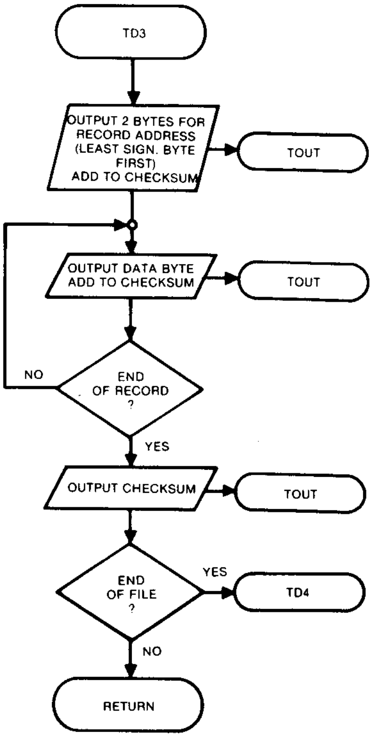

READ
INPUT BYTE FROM KEYBOARD STRIP PARITY
CONTROL-X ?
YES
RESTRT
OUTT

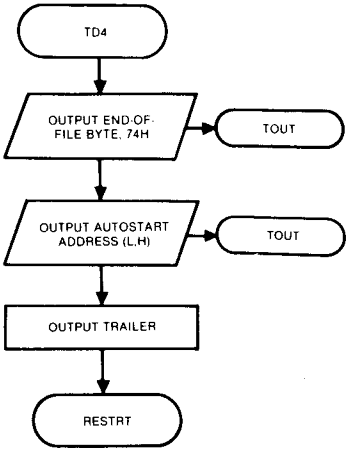

GO
LOOK FOR A CARRIAGE RET. AT END OF KEYBOARD INPUT
READ
C.R. ?
NO
ERROR
YES
LINE

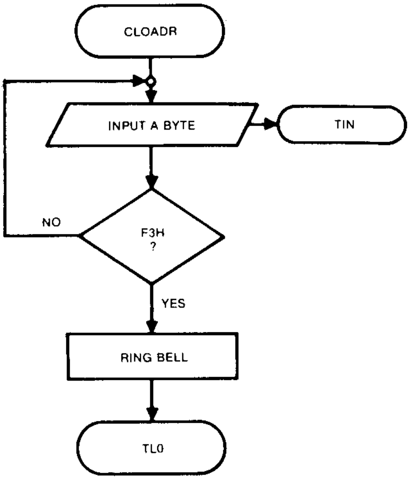

CRLF
PRINT A CR
OUTT
LINE
PRINT LINE FEED, NULLS
OUTT
OUTT
PRINT A BYTE ON CONSOLE
RETURN

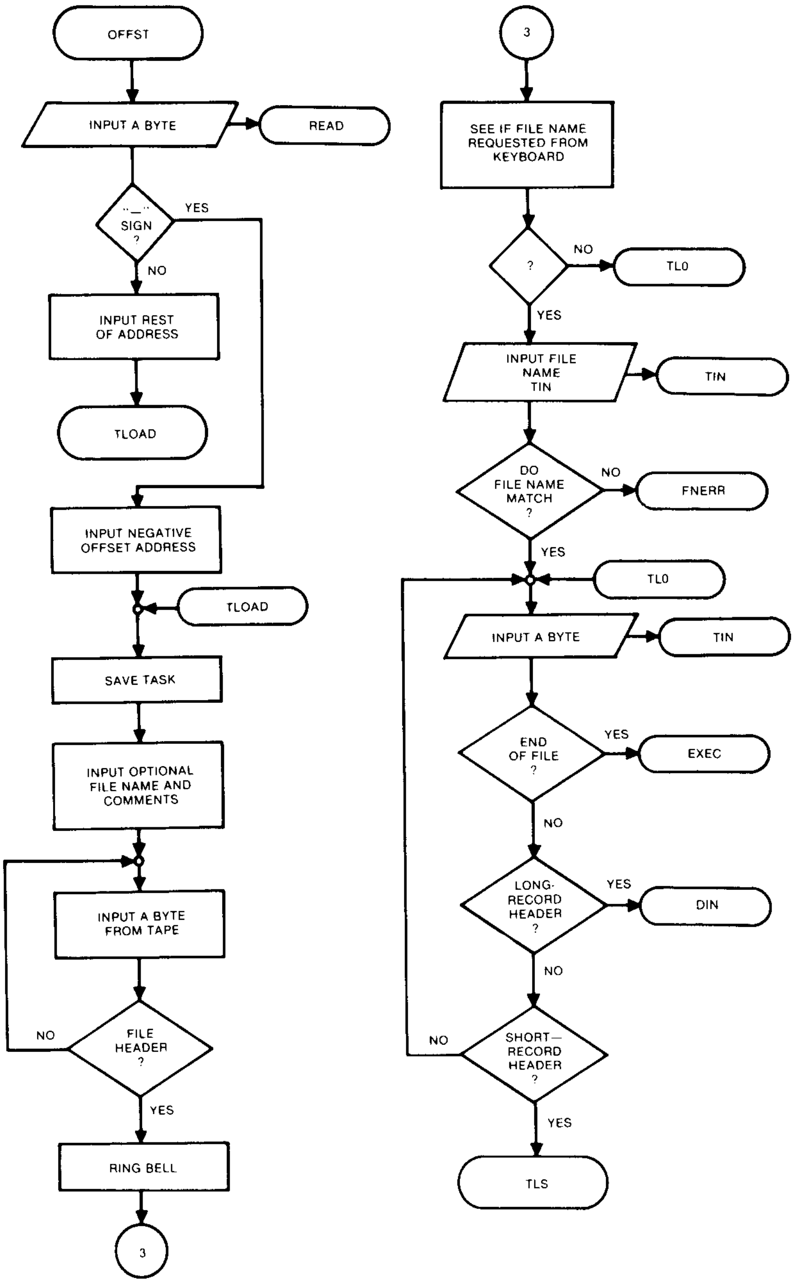

TDUMP
INPUT START, STOP AND AUTOSTART ADDRESSES
INHL
INPUT OPTIONAL FILE NAME AND COMMENTS
OUTPUT LEADER, FILE HEADER, OPTIONAL FILE NAME & COMMENTS
TOUT
LONG RECORDS ?
YES
DOUB
OUTPUT RECORD HEADER, 3CH AND RECORD LENGTH
TOUT
OUTPUT ADDRESS, DATA & CHECKSUM
TD3

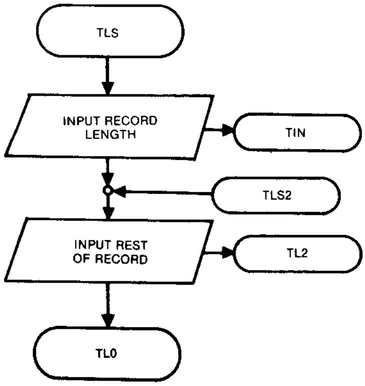

DOUB
OUTPUT RECORD HEADER, 77H AND RECORD LENGTH
TOUT
OUTPUT ADDRESS DATA & CHECKSUM
TD3

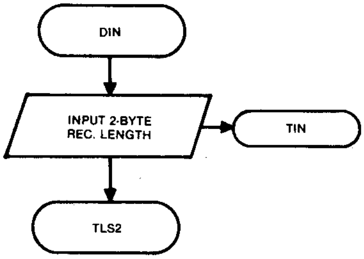

TD3
OUTPUT 2 BYTES FOR RECORD ADDRESS (LEAST SIGN. BYTE FIRST) ADD TO CHECKSUM
TOUT
OUTPUT DATA BYTE ADD TO CHECKSUM
TOUT
NO
END OF RECORD ?
YES
OUTPUT CHECKSUM
TOUT
END OF FILE ?
YES
TD4
NO
RETURN

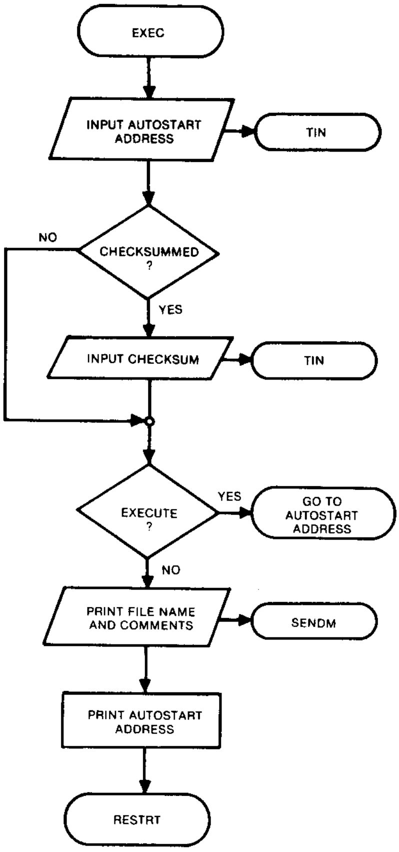

TD4
OUTPUT END-OF-FILE BYTE, 74H
TOUT
OUTPUT AUTOSTART ADDRESS (L,H)
TOUT
OUTPUT TRAILER
RESTRT

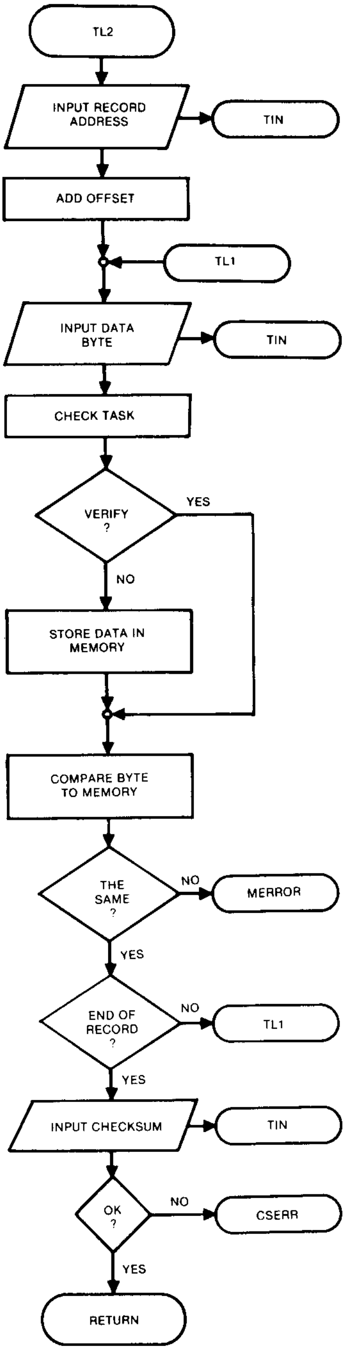

CLOADR
INPUT A BYTE
TIN
NO
F3H ?
YES
RING BELL
TLO

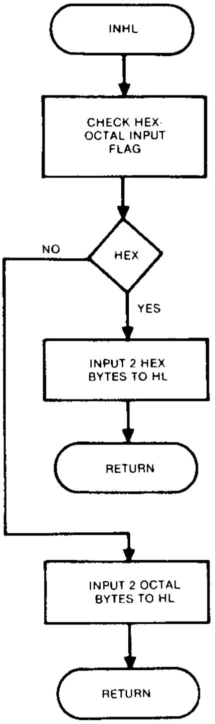

OFFST
INPUT A BYTE
READ
"–" SIGN ?
YES
NO
INPUT REST OF ADDRESS
TLOAD
INPUT NEGATIVE OFFSET ADDRESS
TLOAD
SAVE TASK
INPUT OPTIONAL FILE NAME AND COMMENTS
INPUT A BYTE FROM TAPE
FILE HEADER ?
NO
YES
RING BELL
3
3
SEE IF FILE NAME REQUESTED FROM KEYBOARD
?
NO
TL0
YES
INPUT FILE NAME TIN
TIN
DO FILE NAME MATCH ?
NO
FNERR
YES
TL0
INPUT A BYTE
TIN
END OF FILE ?
YES
EXEC
NO
LONG-RECORD HEADER ?
YES
DIN
NO
SHORT—RECORD HEADER ?
NO
YES
TLS

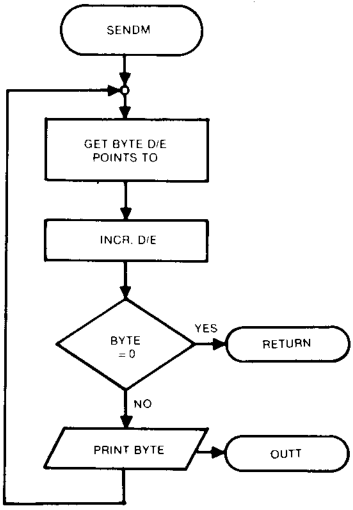

TLS
INPUT RECORD LENGTH
TIN
TLS2
INPUT REST OF RECORD
TL2
TL0

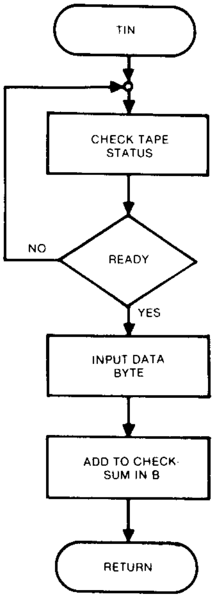

DIN
INPUT 2-BYTE REC. LENGTH
TIN
TLS2

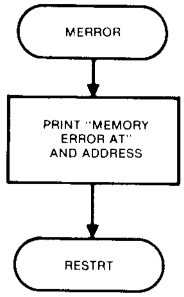

EXEC
INPUT AUTOSTART ADDRESS
TIN
CHECKSUMMED ?
NO
YES
INPUT CHECKSUM
TIN
EXECUTE ?
YES
GO TO AUTOSTART ADDRESS
NO
PRINT FILE NAME AND COMMENTS
SENDM
PRINT AUTOSTART ADDRESS
RESTRT

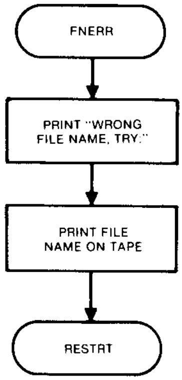

TL2
INPUT RECORD ADDRESS
TIN
ADD OFFSET
TL1
INPUT DATA BYTE
TIN
CHECK TASK
VERIFY ?
YES
NO
STORE DATA IN MEMORY
COMPARE BYTE TO MEMORY
THE SAME ?
NO
MERROR
YES
END OF RECORD ?
NO
TL1
YES
INPUT CHECKSUM
TIN
OK ?
NO
CSERR
YES
RETURN

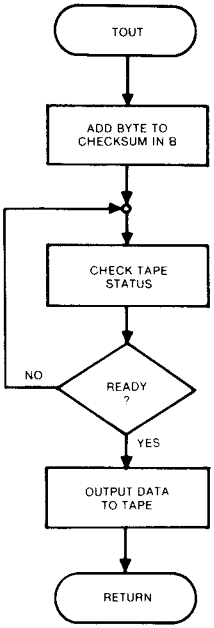

INHL
CHECK HEX-OCTAL INPUT FLAG
NO
HEX
YES
INPUT 2 HEX BYTES TO HL
RETURN
INPUT 2 OCTAL BYTES TO HL
RETURN

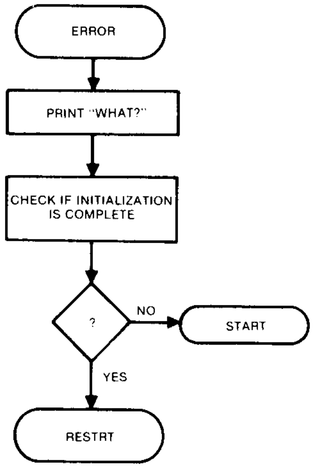

SENDM
GET BYTE D/E POINTS TO
INCR. D/E
BYTE = 0
YES
RETURN
NO
PRINT BYTE
OUTT

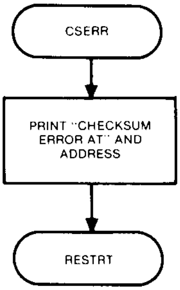

TIN
CHECK TAPE STATUS
READY
NO
YES
INPUT DATA BYTE
ADD TO CHECK-SUM IN B
RETURN


MERROR
PRINT "MEMORY ERROR AT" AND ADDRESS
RESTRT


FNERR
PRINT "WRONG FILE NAME, TRY:"
PRINT FILE NAME ON TAPE
RESTRT


TOUT
ADD BYTE TO CHECKSUM IN B
CHECK TAPE STATUS
READY ?
NO
YES
OUTPUT DATA TO TAPE
RETURN


ERROR
PRINT "WHAT?"
CHECK IF INITIALIZATION IS COMPLETE
?
NO
START
YES
RESTRT



PROGRAM LISTING

```
; TAPEMØN:  PRØGRAM TØ LØAD, DUMP AND VERIFY BINARY
;           CHECKSUMMED TAPES WITH AUTØSTART AND
;           WITH A CHØICE ØF HEX ØR ØCTAL INPUT
;
; PRØGRAMMED FØR AN 8080 MICRØPRØCESSØR
; BY ALAN R. MILLER
; NEW MEXICØ TECH, SØCØRRØ, NM   87801
; 505-835-5619
; INTERFACE AGE, FEBRUARY 1978, PAGE 144
;
        TITLE   'BINARY TAPE MØNITØR'
;
; KEYBØARD ADDRESS IS 10/11HEX (20/21 ØCTAL)
; TAPE ADDRESS IS 6/7.  TAPE AND KEYBØARD CAN HAVE
; THE SAME ADDRESS SØ THAT A TELETYPE TAPE CAN BE
; PUNCHED.
;
; WHEN STARTED AT 'START', PRØGRAM PRINTS:
;
;       'HEX ØR ØCTAL INPUT?'
;
; TYPE AN 'H' FØR HEX MØDE ØR AN 'Ø' FØR ØCTAL MØDE
; AND A CARRIAGE RETURN.  (A CARRIAGE RETURN WITHØUT
; AN 'H' ØR 'Ø' DEFAULTS TØ HEX MØDE.)  THE PRØGRAM
; THEN PRINTS:
;
;       'RECØRD LENGTH?'
;
; TYPE A TWØ-BYTE HEX ØR ØCTAL RECØRD LENGTH AND A
; CARRIAGE RETURN.  (A CARRIAGE RETURN WITH NØ ØTHER
; INPUT DEFAULTS TØ A RECØRD LENGTH ØF 255.)   A PRØMPT
; ØF '>:' IS THEN PRINTED.  THE VALID CØMMANDS ARE:
; 'M', 'R', 'G', 'D', 'L', 'E', 'V', 'Ø', 'N', AND 'C'.
;
; TYPE AN 'M' TØ DETERMINE THE RECØRD LENGTH AND
; WHETHER THE INPUT MØDE IS ØCTAL ØR HEX.
;
; TYPE AN 'R' TØ REINITIALIZE THE SYSTEM SØ THAT THE
; INPUT MØDE AND RECØRD LENGTH CAN BE CHANGED.
;
; TØ DUMP A PØRTIØN ØF MEMØRY TØ TAPE TYPE 'D', THE
; START ADDRESS (MØST SIGNIFICANT BYTE FIRST), THE
; STØP ADDRESS, THE EXECUTIØN (AUTØSTART) ADDRESS AND
; ØPTIØNALLY A FILE NAME AND CØMMENTS, THEN A CARRIAGE
; RETURN.  THE FILE NAME MAY HAVE 1 TØ 8 CHARACTERS,
; FØLLØWED ØPTIØNALLY BY A CØMMA AND A CØMMENT (E.G.,
; VERSIØN) CØNTAINING UP TØ 8 CHARACTERS.  ERRØRS
; MADE DURING ENTRY ØF FILE NAME ØR CØMMENT CAN BE
; CØRRECTED BY PRESSING THE DEL (RUB ØUT) KEY.
; THE TAPE FØRMAT IS:
```

```
;         SYNC BYTE (FILE HEADER) (55 HEX, 125 ØCTAL)
;
;         RECØRD-HEADER BYTE
;             (3C HEX, 74 ØCTAL FØR RECØRDS < 256 BYTES)
;             (77 HEX, 167 ØCTAL FØR RECØRDS >255 BYTES)
;         RECØRD-LENGTH  (NUMBER ØF DATA BYTES)
;             ØNE BYTE FØR RECØRDS < 256 LØNG
;             TWØ BYTES FØR RECØRDS > 255 LØNG
;             (LEAST-SIGNIFICANT BYTE FIRST)
;         2-BYTE RECØRD ADDRESS (LØW/HIGH)
;         DATA BYTES
;         CHECKSUM BYTE (SUM ØF RECØRD ADDRESS AND DATA)
;
;         END-ØF-FILE BYTE (74 HEX, 164 ØCTAL)
;         2-BYTE AUTØSTART ADDRESS (LØW/HIGH)
;         CHECKSUM BYTE ØN THE AUTØSTART ADDRESS
;
; TØ LØAD A TAPE, TYPE 'L', ØPTIØNALLY THE FILE NAME,
; AND A CARRIAGE RETURN.  IF A FILE NAME IS ENTERED
; THAT DØES NØT MATCH THE ØNE ØN THE TAPE, AN ERRØR
; MESSAGE IS PRINTED ALØNG WITH THE CØRRECT FILE NAME
; ØN THE TAPE.
;
; TØ LØAD A TAPE AT ØTHER THAN ITS NØRMAL ADDRESS, TYPE
; 'Ø', A TWØ-BYTE ØFFSET ADDRESS TØ BE ADDED TØ H,L,
; ØPTIØNALLY THE FILE NAME, AND A CARRIAGE RETURN.
; AS THE TAPE LØADS AT THE NEW ADDRESS, THE CHECKSUM
; WILL BE PRØPERLY CALCULATED.  AN ØFFSET ØF 0400 (HEX)
; WILL LØAD THE PRØGRAM 1K HIGHER, AN ØFFSET ØF F000
; ØR -1000 WILL LØAD THE PRØGRAM 4K LØWER.
;
; TØ LØAD ANY MITS CHECKSUMMED TAPE (WHICH HAS A
; CHECKSUM LØADER  AT THE BEGINNING), TYPE A 'C' (FØR
; CHECKSUM) AND CR.  THE PRØGRAM SEARCHES FØR THE
; DISABLE-INTERRUPT (DI) INSTRUCTIØN AT THE END ØF THE
; CHECKSUM LØADER, THEREBY SKIPPING ØVER IT.
;
; TØ VERIFY A TAPE AGAINST MEMØRY, TYPE 'V', AN
; ØPTIØNAL FILE NAME AND A CARRIAGE RETURN.
;
; FØR THE ABØVE THREE CASES, THE AUTØSTART ADDRESS IS
; PRINTED AND THIS PRØGRAM IS RESTARTED.  IF A FILE NAME
; IS ENTERED THE FILE NAME AND ANY CØMMENTS ARE PRINTED.
;
; TØ LØAD AND EXECUTE A TAPE, TYPE E, ØPTIØNALLY A
; FILE NAME, AND A CARRIAGE RETURN.  THE PRØGRAM
; CØUNTER WILL JUMP TØ THE EXECUTE ADDRESS AFTER
; THE TAPE HAS BEEN LØADED.
;
; TYPE A 'N' TØ CHANGE THE LEADER AND TRAILER LENGTH.
; ANSWER THE QUESTIØN 'LEADER LENGTH' WITH THE NUMBER
; ØF DESIRED NULLS (IN HEX ØR ØCTAL DEPENDING ØN THE
; MØDE).  ØNE IS GØØD FØR MAGNETIC TAPE.  40H WILL
; GIVE A 6-IN LEADER ØN PAPER TAPE.  THE DEFAULT IS 1.
;
; ENTER A 'G', AN ADDRESS, AND A CARRIAGE RETURN
; TØ GØ SØMEWHERE ELSE, E.G. TØ RETURN TØ YØUR
; REGULAR MØNITØR.
;
; A CØNTRØL-X ØN INPUT WILL RESTART THIS PRØGRAM.
; IF A CHECKSUM ERRØR ØCCURS DURING LØAD ØR VERIFY,
; AN ERRØR MESSAGE AND THE ADDRESS WILL BE PRINTED.
; A MEMØRY ERRØR (LØADING INTØ PRØTECTED, DEFECTIVE,
; ØR NØN-EXISTENT MEMØRY) WILL PRINT AN ERRØR MESSAGE
; AND THE ADDRESS.  A 'WHAT?' WILL THEN BE PRINTED AND
; THIS PRØGRAM WILL BE RESTARTED ØN IMPRØPER INPUT.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
5800               ØRG      5800H
                   ;
0000 =             FALSE    EQU      0
FFFF =             TRUE     EQU      -1
                   ;
FFFF =             JMPZ     EQU      TRUE     ;JUMP ØN ZERØ
                   ;
                   ; EQUATES
```

```
00FF =          RLEN     EQU     255       ;DEFAULT RECØRD LENGTH
003C =          SNUL     EQU     60        ;DEFAULT # ØF LEADER NULLS
0055 =          SBYTE    EQU     55H       ;SYNC BYTE (FILE HEADER)
003C =          RHEAD    EQU     3CH       ;RECØRD-HEADER FØR SHØRT RECØRDS
0077 =          LHEAD    EQU     77H       ;RECØRD-HEADER FØR LØNG RECØRDS
0074 =          EØFC     EQU     74H       ;END-ØF-FILE HEADER
0078 =          EØF      EQU     78H       ;MITS EØF HEADER
0012 =          TAPES    EQU     12H       ;TAPE STATUS ADDRESS
0013 =          TAPED    EQU     TAPES+1   ;TAPE DATA ADDRESS
0001 =          ACINM    EQU     1         ;TAPE INPUT-READY MASK
0002 =          ACØM     EQU     2         ;TAPE ØUTPUT-READY MASK
0010 =          TYSTAT   EQU     10H       ;KEYBØARD STATUS ADDRESS
0011 =          TYDATA   EQU     TYSTAT+1  ;KEYBØARD DATA ADDRESS
0001 =          INMASK   EQU     1         ;KEYBØARD INPUT-READY MASK
0002 =          ØUTMSK   EQU     2         ;KEYBØARD ØUTPUT-READY MASK
0012 =          BUFL     EQU     18        ;INPUT-BUFFER LENGTH
000D =          CR       EQU     0DH       ;CARRIAGE RETURN
000A =          LF       EQU     0AH       ;LINE FEED
007F =          DEL      EQU     7FH       ;DELETE CHARACTER
0008 =          BACKUP   EQU     8         ;BACKUP CHARACTER
                ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                ;
5800 AF         START:   XRA     A         ;GET A ZERØ
5801 320A5D              STA     HEXFL     ;RESET FLAG FØR HEX INPUT
5804 320B5D              STA     SFLAG     ;SET FØR INITIALIZATIØN
5807 3E3C                MVI     A,SNUL    ;DEFAULT NUMBER ØF NULLS
5809 32145D              STA     NNUL      ;SET LEADER NULLS TØ DEFAULT
580C 31EA5D              LXI     SP,STACK
580F CD4A59              CALL    CRLF
5812 11155D              LXI     D,MESG    ;PØINT TØ FIRST MESSAGE
5815 CD005D              CALL    SENDM     ;SEND IT
5818 CD2C59              CALL    READ      ;INPUT HEX/ØCTAL MØDE
581B FE0D                CPI     CR        ;CARRIAGE RETURN FØR HEX
581D C22658              JNZ     INIT0
5820 CD4F59              CALL    LINE      ;ØUTPUT LINE FEED FØR CR
5823 C32E58              JMP     INIT1
5826 FE48       INIT0:   CPI     'H'       ;HEX INPUT?
5828 C23458              JNZ     INIT3     ;JUMP IF NØT
582B CD4A59              CALL    CRLF      ;ØUTPUT CR AND LF
582E CD005D     INIT1:   CALL    SENDM     ;PRINT 'HEX'
5831 C34558              JMP     INIT5
5834 FE4F       INIT3:   CPI     'Ø'       ;Ø FØR ØCTAL INPUT
5836 C20058              JNZ     START     ;ERRØR, TRY AGAIN
5839 320A5D              STA     HEXFL     ;STØRE 'Ø' IN HEX FLAG
583C CD4A59              CALL    CRLF      ;ØUTPUT CR AND LF
583F 112F5D              LXI     D,MES2    ;PRINT 'ØCTAL'
5842 CD005D              CALL    SENDM     ;SEND IT
5845 11365D     INIT5:   LXI     D,MES3    ;PØINT TØ 'INPUT', ETC
5848 CD005D              CALL    SENDM     ;PRINT MESSAGE
584B CD005D              CALL    SENDM     ;PRINT '?'
584E 3A0A5D              LDA     HEXFL     ;FETCH HEX/ØCTAL FLAG
5851 B7                  ØRA     A         ;IS IT ZERØ?
5852 CA7F58              JZ      INITH     ;JUMP IF HEX MØDE
5855 CD6458              CALL    RDCR      ;SEE IF FIRST BYTE IS CR
5858 CDF55C              CALL    ØCTI2     ;SECØND ØCTAL BYTE
585B CDDB5C              CALL    RØCT2
585E CDB55C              CALL    RHLØ2
5861 C38B58              JMP     INIT6
5864 CD2C59     RDCR:    CALL    READ      ;FIRST BYTE ØF RECØRD LENGTH
5867 FE0D                CPI     CR        ;IS IT A CARRIAGE RETURN?
5869 C0                  RNZ               ;RETURN IF NØT
586A CD4F59              CALL    LINE      ;ØUTPUT LINE FEED
586D 11415D              LXI     D,MES4    ;PØINT TØ 'RECØRD LENGTH'
5870 CD005D              CALL    SENDM     ;PRINT IT
5873 11525D              LXI     D,MESR    ;PØINT TØ '?'
5876 CD005D              CALL    SENDM     ;PRINT IT
5879 21FF00              LXI     H,RLEN    ;SET STANDARD RECØRD LENGTH
587C C38B58              JMP     INIT6
587F CD6458     INITH:   CALL    RDCR      ;SEE IF FIRST BYTE IS A CR
5882 CD895C              CALL    HEX22     ;SECØND HEX BYTE
5885 CD7B5C              CALL    RDHX2
5888 CDA85C              CALL    RHL2
588B 220F5D     INIT6:   SHLD    RECLN     ;STØRE STANDARD RECØRD LENGTH
588E 3EFF                MVI     A,255
5890 320B5D              STA     SFLAG     ;SET INITIALIZATIØN FLAG
```

```
5893 31EA5D   RESTRT: LXI    SP,STACK
5896 210000           LXI    H,0      ;ZERØ H,L
5899 220D5D           SHLD   ØFSET    ;ZERØ THE LØAD-ØFFSET VECTØR
589C AF               XRA    A        ;GET A ZERØ
589D 32135D           STA    LFLAG    ;RESET LØAD-ERRØR FLAG
58A0 CD4A59           CALL   CRLF
58A3 3E3E             MVI    A,'>'    ;PRINT '>:' FØR
58A5 CD5B59           CALL   ØUTT     ; A PRØMPT
58A8 CDAD5C           CALL   CØLØN    ;THEN A CØLØN
58AB CD2C59           CALL   READ     ;INPUT THE TASK
58AE FE4D             CPI    'M'      ;PRINT MØDE AND RECØRD LENGTH
58B0 CAE758           JZ     MØDE
58B3 FE44             CPI    'D'      ;DUMP TØ TAPE
58B5 CA9D59           JZ     TDUMP
58B8 FE4C             CPI    'L'      ;LØAD
58BA CAFF5A           JZ     TLØAD
58BD FE45             CPI    'E'      ;LØAD AND EXECUTE
58BF CAFF5A           JZ     TLØAD
58C2 FE56             CPI    'V'      ;VERIFY
58C4 CAFF5A           JZ     TLØAD
58C7 FE4F             CPI    'Ø'      ;LØAD TAPE AT AN ØFFSET
58C9 CABB5A           JZ     ØFFST
58CC FE43             CPI    'C'      ;SKIP ØVER MITS CHECKSUM LØADER
58CE CA225C           JZ     CLØADR
58D1 FE52             CPI    'R'      ;RESET HEX/ØCTAL MØDE
58D3 CA0058           JZ     START    ; AND RECØRD LENGTH
58D6 FE4E             CPI    'N'      ;SET NUMBER ØF LEADER NULLS
58D8 CA0C59           JZ     SETN
58DB FE47             CPI    'G'      ;GØ SØMEWHERE
58DD C28D59           JNZ    ERRØR
              ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
              ;
              ; RØUTINE TØ JUMP TØ ANØTHER PRØGRAM

58E0 CD9E5C                   CALL   INHL     ;GET H,L ADDRESS
58E3 CD3F59   GØ2:    CALL   GØ       ;LØØK FØR CARRIAGE RETURN
58E6 E9       JPCHL:  PCHL            ;ØK, GØØDBYE
              ;
              ; SUBRØUTINE TØ PRINT CURRENT MØDE (HEX ØR ØCTAL)
              ; AND RECØRD LENGTH
              ;
58E7 3A0A5D   MØDE:   LDA    HEXFL    ;FETCH HEX/ØCTAL MØDE FLAG
58EA B7               ØRA    A        ;IS IT ZERØ?
58EB CAF758           JZ     MØDE1    ;HEX INPUT IF ZERØ
58EE 112F5D           LXI    D,MES2   ;PØINT TØ 'ØCTAL '
58F1 CD005D           CALL   SENDM    ;PRINT MESSAGE
58F4 C3FD58           JMP    MØDE2
58F7 112A5D   MØDE1:  LXI    D,MES1   ;PØINT TØ 'HEX'
58FA CD005D           CALL   SENDM    ;SEND MESSAGE
58FD 11365D   MØDE2:  LXI    D,MES3   ;PØINT TØ 'INPUT'
5900 CD005D           CALL   SENDM    ;PRINT MESSAGE
5903 CD6759           CALL   BLANK    ;PRINT A BLANK
5906 2A0F5D           LHLD   RECLN    ;FETCH STANDARD RECØRD LEN
5909 C37259           JMP    TERR3    ;PRINT H,L IN HEX AND ØCTAL
              ;
              ; SUBRØUTINE TØ SET NUMBER ØF NULLS ØN TAPE
              ; LEADER AND TRAILER
              ;
590C 11705D   SETN:   LXI    D,MESN   ;PØINT TØ MESSAGE
590F CD005D           CALL   SENDM    ;PRINT IT
5912 3A0A5D           LDA    HEXFL    ;FETCH HEX/ØCTAL FLAG
5915 B7               ØRA    A        ;IS IT ZERØ
5916 CA1F59           JZ     SETN2    ;JUMP IF ZERØ
5919 CDD85C           CALL   RDØCT    ;ØCTAL INPUT
591C C32259           JMP    SETN3
591F CD785C   SETN2:  CALL   RDHEX    ;HEX INPUT
5922 78       SETN3:  MØV    A,B      ;PUT IN A
5923 32145D           STA    NNUL     ;STØRE IN MEMØRY
5926 CD4A59           CALL   CRLF
5929 C39358           JMP    RESTRT
              ;
              ; SUBRØUTINE TØ INPUT A BYTE FRØM KEYBØARD
              ;
592C DB10     READ:   IN     TYSTAT   ;CHECK STATUS
592E E601             ANI    INMASK   ;MASK UNWANTED BITS
5930 CA2C59           JZ     READ     ;LØØP UNTIL READY
```

```
5933 DB11              IN      TYDATA  ;READ CHARACTER
5935 E67F              ANI     7FH     ;STRIP PARITY
5937 FE18              CPI     24      ;RESTART ØN
5939 CA9358            JZ      RESTRT  ; ØN CØNTRØL-X
593C C35B59            JMP     ØUTT    ;ECHØ INPUT
                ;
                ; SUBRØUTINE TØ LØØK FØR A CARRIAGE RETURN
                ; AT THE END ØF KEYBØARD-INPUT LINE
                ;
593F CD2C59     CØ:    CALL    READ    ;INPUT CHARACTER
5942 FE0D              CPI     CR      ;A CARRIAGE RETURN?
5944 C28D59            JNZ     ERRØR   ;NØ, RESTART
5947 C34F59            JMP     LINE    ;LINE FEED AND NULLS
                ;
                ; CARRIAGE RETURN, LINE FEED AND NULLS
                ;
594A 3E0D       CRLF:  MVI     A,CR    ;CARRIAGE RETURN
594C CD5B59            CALL    ØUTT
594F 3E0A       LINE:  MVI     A,LF    ;LINE FEED
5951 CD5B59            CALL    ØUTT
5954 AF                XRA     A       ;GET A ZERØ
5955 CD5B59            CALL    ØUTT    ;ØUTPUT THREE NULLS
5958 CD5B59            CALL    ØUTT    ;TØ PRINTER
                ;
                ; SUBRØUTINE TØ ØUTPUT A CHARACTER FRØM KEYBØARD
                ;
595B F5         ØUTT:  PUSH    PSW
595C DB10       WAITØ: IN      TYSTAT  ;CHECK STATUS
595E E602              ANI     ØUTMSK  ;ØUTPUT READY?
5960 CA5C59            JZ      WAITØ   ;NØ, LØØP UNTIL READY
5963 F1                PØP     PSW     ;YES
5964 D311              ØUT     TYDATA  ;ØUTPUT A BYTE TØ KEYBØARD
5966 C9                RET
                ;
5967 3E20       BLANK: MVI     A,' '   ;LØAD A BLANK
5969 C35B59            JMP     ØUTT    ;PRINT IT
                ;
                ; ERRØR MESSAGES
                ;
596C 115F5D     MERRØR: LXI    D,MESM  ;MEMØRY ERRØR
596F CD005D     TERR2: CALL    SENDM   ;SEND MESSAGE
5972 CD5F5C     TERR3: CALL    ØUTHL   ;PRINT H/L IN HEX
5975 CDAD5C            CALL    CØLØN
5978 CDBC5C            CALL    ØUTHLØ  ;PRINT H/L IN ØCTAL
597B C39358            JMP     RESTRT
597E 11A05D     FNERR: LXI     D,MESF  ;PØINT TØ INPUT FILE NAME
5981 CD005D            CALL    SENDM   ;PRINT IT
5984 11C05D            LXI     D,IBUF  ;PØINT TØ FILE NAME ØN TAPE
5987 CD005D            CALL    SENDM   ;PRINT IT
598A C39358            JMP     RESTRT
598D 11575D     ERRØR: LXI     D,MESW  ;PØINT TØ 'WHAT?'
5990 CD005D            CALL    SENDM   ;PRINT IT
5993 3A0B5D            LDA     SFLAG   ;FETCH INITIALIZATIØN FLAG
5996 B7                ØRA     A       ;IS IT ZERØ?
5997 CA0058            JZ      START   ;START ØVER IF SØ
599A C39358            JMP     RESTRT  ;ØTHERWISE RESTART
                ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                ;
                ; ENTRY FØR DUMP TØ TAPE
                ;
599D CD9E5C     TDUMP: CALL    INHL    ;INPUT START ADDRESS (HEX)
59A0 EB                XCHG
59A1 CD9E5C            CALL    INHL    ;INPUT STØP ADDRESS
59A4 EB                XCHG            ;START TØ H/L, STØP TØ D/E
59A5 13                INX     D       ;INCREMENT STØP ADDRESS
59A6 E5                PUSH    H       ;PUSH H/L ØNTØ STACK
59A7 CD9E5C            CALL    INHL    ;INPUT AUTØSTART ADDRESS
59AA E3                XTHL            ;EXCHANGE STACK FØR H/L
59AB E5                PUSH    H
59AC 0E09              MVI     C,9     ;FILE-NAME CØUNT PLUS ØNE
59AE 21C05D            LXI     H,IBUF  ;PØINT TØ INPUT BUFFER
59B1 CD9D5A     TDMP3: CALL    RFILE   ;INPUT FILE-NAME CHAR.
59B4 77                MØV     M,A     ;PUT CHARACTER IN BUFFER
59B5 23                INX     H       ;INCREMENT BUFFER PØINTER
59B6 FE0D              CPI     CR      ;LØØK FØR CARRIAG RETURN
59B8 CAD759            JZ      TDMP5   ; AT END ØF FILE NAME
```

```
59BB FE2C                CPI      ','        ;CØMMA AT END ØF FILE NAME
59BD CAC759              JZ       TDMP6      ;CØMMENTS CØME NEXT
59C0 0D                  DCR      C          ;DECREMENT FILE-NAME CØUNT
59C1 CA8D59              JZ       ERRØR      ;QUIT IF TØØ MANY CHARACTERS
59C4 C3B159              JMP      TDMP3      ;NEXT CHARACTER
59C7 0E0A      TDMP6:    MVI      C,10       ;8 CØMMENT CHARACTERS
59C9 0D        TDMP7:    DCR      C          ;DECREMENT CØUNT
59CA CA8D59              JZ       ERRØR
59CD CD9D5A    TDMP8:    CALL     RFILE      ;INPUT CØMMENT
59D0 77                  MØV      M,A        ;STØRE IN MEMØRY
59D1 23                  INX      H          ;INCREMENT PØINTER
59D2 FE0D                CPI      CR         ;LØØK FØR CARRIAGE RETURN
59D4 C2C959              JNZ      TDMP7      ; AT END ØF CØMMENT
59D7 CD4F59    TDMP5:    CALL     LINE       ;PRINT A LINE FEED
59DA CD365C              CALL     LEADR      ;ØUTPUT A LEADER ØF NULLS
59DD 3E55                MVI      A,SBYTE    ;SYNC BYTE
59DF CD515C              CALL     TØUT       ;ØUTPUT FILE HEADER
59E2 21C05D              LXI      H,IBUF     ;PØINT TØ INPUT BUFFER
59E5 7E        TDMP4:    MØV      A,M        ;FETCH FILE NAME
59E6 CD515C              CALL     TØUT       ;ØUTPUT FILE NAME
59E9 23                  INX      H
59EA FE0D                CPI      CR         ;CARRIAGE RETURN MARKS
59EC C2E559              JNZ      TDMP4      ; END ØF FILE NAME/CØMMENTS
59EF E1                  PØP      H
59F0 3A105D              LDA      RECL2      ;FETCH HIGH HALF ØF REC. LEN.
59F3 B7                  ØRA      A          ;EQUAL TØ ZERØ?
59F4 C2685A              JNZ      DØUB       ;NØ, RECØRD LENGTH > 255
               ;
               ; RØUTINE TØ DUMP RECØRDS LESS THAN 256 BYTES LØNG
               ;
59F7 3E3C      TD0:      MVI      A,RHEAD    ;RECØRD-HEADER BYTE
59F9 CD515C              CALL     TØUT       ;ØUTPUT RECØRD HEADER
59FC AF                  XRA      A          ;ZERØ ACCUMULATØR
59FD 32115D              STA      RECA       ;ZERØ HIGH BYTE ØF REC LENGTH
5A00 CD905A              CALL     CIND       ;HØW FAR TØ END?
5A03 3A0F5D              LDA      RECLN      ;SET FØR FULL RECØRD
5A06 C20E5A              JNZ      NEW2       ;USE FULL RECØRD LENGTH (D>H)
5A09 B9                  CMP      C          ;CØMPARE TØ E - L
5A0A DA0E5A              JC       NEW2       ;USE FULL RECØRD LENGTH
5A0D 79                  MØV      A,C        ;SHØRT RECØRD
5A0E 4F        NEW2:     MØV      C,A        ;PUT RECØRD LENGTH IN C
5A0F CD515C              CALL     TØUT       ;ØUTPUT RECØRD LENGTH
5A12 CD185A              CALL     TD3        ;ØUTPUT H,L, DATA, CHECKSUM
5A15 C3F759              JMP      TD0        ;START NEXT RECØRD
               ;
               ; ØUTPUT RECØRD ADDRESS, DATA, AND CHECKSUM
               ; TEST FØR END ØF FILE AND RETURN IF NØT EØF
               ;
5A18 7D        TD3:      MØV      A,L        ;ØUTPUT LØW BYTE
5A19 CD515C              CALL     TØUT       ; ØF RECØRD ADDRESS
5A1C 45                  MØV      B,L        ;START CHECKSUM WITH L
5A1D 7C                  MØV      A,H        ;ØUTPUT HIGH BYTE
5A1E CD515C              CALL     TØUT       ; ØF RECØRD ADDRESS
5A21 7E        TD1:      MØV      A,M        ;FETCH DATA BYTE
5A22 CD515C              CALL     TØUT       ;ØUTPUT IT
5A25 23                  INX      H          ;INCREMENT PØINTER
5A26 79                  MØV      A,C        ;GET RECØRD CØUNT (LØW)
5A27 D601                SUI      1          ;DECREMENT IT
5A29 4F                  MØV      C,A        ;SAVE IT BACK IN C
5A2A CA3E5A              JZ       TD5        ;JUMP IF C IS ZERØ
5A2D D2215A              JNC      TD1        ;CØNTINUE IF NØT 255
5A30 3A115D              LDA      RECA       ;FETCH RECØRD CØUNT (HIGH)
5A33 D601                SUI      1          ;DECREMENT IT
5A35 32115D              STA      RECA       ;SAVE IT
5A38 DA455A              JC       TD2        ;END ØF RECØRD IF 255
5A3B C3215A              JMP      TD1        ;NEXT BYTE
5A3E 3A115D    TD5:      LDA      RECA       ;FETCH RECØRD CØUNT (HIGH)
5A41 B1                  ØRA      C          ;SEE IF BØTH HIGH AND LØW = 0
5A42 C2215A              JNZ      TD1        ;CØNTINUE IF NØT
               ;
               ; END ØF RECØRD
               ; PRØCESS CHECKSUM AND SEE IF END ØF FILE
               ;
5A45 78        TD2:      MØV      A,B        ;FETCH CHECKSUM
5A46 CD515C              CALL     TØUT       ;ØUTPUT IT
```

```
5A49 CD905A           CALL    CEND     ;HØW MUCH IS LEFT?
5A4C B1               ØRA     C        ;ZERØ?
5A4D C0               RNZ              ;START NEXT RECØRD
                ;
                ; END ØF FILE, ØUTPUT EØF BYTE AND AUTØSTART ADDRESS
                ;
5A4E F1         TD4:  PØP     PSW      ;RAISE STACK
5A4F 3E74             MVI     A,EØFC   ;END-ØF-FILE MARK
5A51 CD515C           CALL    TØUT     ;ØUTPUT IT
5A54 E1               PØP     H        ;FETCH AUTØSTART ADDRESS
5A55 7D               MØV     A,L
5A56 CD515C           CALL    TØUT     ;ØUTPUT LØW HALF
5A59 45               MØV     B,L      ;START CHECKSUM WITH L
5A5A 7C               MØV     A,H
5A5B CD515C           CALL    TØUT     ;ØUTPUT HIGH HALF
5A5E 78               MØV     A,B      ;FETCH CHECKSUM
5A5F CD515C           CALL    TØUT     ;ØUTPUT IT
5A62 CD365C           CALL    LEADR    ;ØUTPUT A TRAILER ØF NULLS
5A65 C39358           JMP     RESTRT   ;NEXT TASK
                ;
                ; RØUTINE TØ DUMP RECØRDS LØNGER THAN 255 BYTES
                ;
5A68 3E77       DØUB: MVI     A,LHEAD  ;LØNG RECØRD-HEADER BYTE
5A6A CD515C           CALL    TØUT     ;ØUTPUT RECØRD HEADER
5A6D CD905A           CALL    CEND     ;HØW FAR TØ END?
5A70 E5               PUSH    H        ;SAVE H,L ØN STACK
5A71 2A0F5D           LHLD    RECLN    ;FETCH FULL RECØRD LENGTH
5A74 7D               MØV     A,L      ;SUBTRACT REMAINING
5A75 91               SUB     C        ; FRØM END ØF
5A76 7C               MØV     A,H      ; FILE
5A77 98               SBB     B        ;CARRY SET IF FULL REC LENGTH
5A78 D27D5A           JNC     DØUBF    ; LØNGER THAN REMAINING BYTES
5A7B 4D               MØV     C,L      ;CØPY FULL RECØRD LENGTH
5A7C 44               MØV     B,H      ; FRØM H,L TØ B,C
5A7D 60         DØUBF: MØV    H,B
5A7E 79               MØV     A,C
5A7F CD515C           CALL    TØUT     ;ØUTPUT REC LEN (LØW BYTE)
5A82 7C               MØV     A,H      ;FETCH HIGH BYTE
5A83 32115D           STA     RECA     ;STØRE HIGH HALF ØF REC LEN
5A86 CD515C           CALL    TØUT     ;ØUTPUT REC LEN (HIGH BYTE)
5A89 E1               PØP     H        ;RESTØRE PØINTER
5A8A CD185A           CALL    TD3      ;ØUTPUT H,L, DATA, CHECKSUM
5A8D C3685A           JMP     DØUB     ;START NEXT RECØRD
                ;
                ; SUBRØUTINE TØ FIND THE DIFFERENCE BETWEEN
                ; D,E AND H,L AND PUT THE DIFFERENCE IN B,C
                ; IF START ADDRESS > STØP ADDRESS PRINT 'WHAT?'
                ;
5A90 7B         CEND: MØV     A,E      ;CØMPARE LØW STØP
5A91 95               SUB     L        ; TØ LØW PØINTER
5A92 4F               MØV     C,A      ;SAVE DIFFERENCE IN C
5A93 7A               MØV     A,D      ;CØMPARE HIGH STØP
5A94 9C               SBB     H        ; TØ HIGH PØINTER
5A95 47               MØV     B,A      ;SAVE DIFFERENCE IN B
5A96 D0               RNC              ;ØK IF D,E > H,L
5A97 7A               MØV     A,D      ;SEE IF D,E
5A98 B3               ØRA     E        ; IS ZERØ
5A99 C28D59           JNZ     ERRØR    ;IMPRØPER INPUT, H,L > D,E
5A9C C9               RET              ;UPPER LIMIT IS FFFF HEX
                ;
                ; SUBRØUTINE TØ INPUT A FILE-NAME ØR CØMMENT CHARACTER
                ; FRØM THE KEYBØARD.  DEL (RUB ØUT) DELETES PRIØR
                ; CHARACTER.
                ;
5A9D CD2C59     RFILE: CALL   READ     ;INPUT FRØM KEYBØARD
5AA0 FE0D             CPI     CR       ;CARRIAGE RETURN
5AA2 C8               RZ               ;YES, RETURN
5AA3 FE20             CPI     ' '      ;CHECK FØR CØNTRØL CHARACTER
5AA5 DA9D5A           JC      RFILE    ;REJECT CØNTRØL CHARACTER
5AA8 FE7F             CPI     DEL      ;DELETE (RUBØUT) CHARACTER
5AAA C0               RNZ
5AAB 79               MØV     A,C      ;FETCH CHARACTER CØUNT
5AAC FE09             CPI     9        ;PØINTER AT START ØF BUFFER?
5AAE CA9D5A           JZ      RFILE    ;YES, IGNØRE DEL
5AB1 2B               DCX     H        ;DECREMENT PØINTER
```

```
5AB2 0C                 INR     C          ;INCREMENT COUNT
5AB3 3E08               MVI     A,BACKUP   ;BACKUP CHARACTER
5AB5 CD5B59             CALL    ØUTT       ;PRINT IT
5AB8 C39D5A             JMP     RFILE      ;NEXT CHARACTER
               ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
               ;
               ; LØAD A TAPE AT ØTHER THAN ITS NØRMAL ADDRESS
               ; ØFFSET VECTØR IS ADDED TØ THE NØRMAL H,L ADDRESS
               ; ØR SUBTRACTED IF PRECEEDED BY A MINUS SIGN
               ;
5ABB 57        ØFFST:   MØV     D,A        ;SAVE 'Ø' CØMMAND IN D
5ABC 3A0A5D             LDA     HEXFL      ;FETCH HEX/ØCTAL INDICATØR
5ABF B7                 ØRA     A          ;IS IT ZERØ?
5AC0 CAEA5A             JZ      ØFF1       ;JUMP IF HEX INPUT
5AC3 CD2C59             CALL    READ       ;INPUT A BYTE
5AC6 FE2D               CPI     '-'        ;CHECK FØR NEGATIVE ØFFSET
5AC8 CAD75A             JZ      ØFF4       ;JUMP IF NEGATIVE
5ACB CDF55C             CALL    ØCTI2      ;CØNTINUE WITH ØCTAL ADDRESS
5ACE CDDB5C             CALL    RØCT2
5AD1 CDB55C             CALL    RHLØ2
5AD4 C3FB5A             JMP     ØFF2
5AD7 CDB25C    ØFF4:    CALL    RDHLØ      ;INPUT NEGATIVE ØCTAL ØFFSET
5ADA C3E05A             JMP     ØFF5
5ADD CDA55C    ØFF3:    CALL    READHL     ;INPUT NEGATIVE HEX ØFFSET
5AE0 AF        ØFF5:    XRA     A          ;GET A ZERØ
5AE1 95                 SUB     L          ;INVERT L
5AE2 6F                 MØV     L,A        ;SAVE IT
5AE3 3E00               MVI     A,0        ;ZERØ A WITHØUT RESETING CARRY
5AE5 9C                 SBB     H          ;INVERT H
5AE6 67                 MØV     H,A        ;SAVE IT
5AE7 C3FB5A             JMP     ØFF2
5AEA CD2C59    ØFF1:    CALL    READ       ;INPUT A BYTE
5AED FE2D               CPI     '-'        ;CHECK FØR NEGATIVE ØFFSET
5AEF CADD5A             JZ      ØFF3       ;JUMP IF NEGATIVE
5AF2 CD895C             CALL    HEX22      ;CØNTINUE WITH HEX ADDRESS
5AF5 CD7B5C             CALL    RDHX2
5AF8 CDA85C             CALL    RHL2
5AFB 220D5D    ØFF2:    SHLD    ØFSET      ;SAVE ØFFSET IN MEMØRY
5AFE 7A                 MØV     A,D        ;MØVE TASK TØ A
               ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
               ;
               ; ENTRY FØR LØAD, EXECUTE, AND VERIFY
               ;
5AFF 320C5D    TLØAD:   STA     TASK       ;SAVE TASK IN MEMØRY
               ;
               ; CHECK FØR INPUT ØF  FILE NAME (UP TØ EIGHT
               ; CHARACTERS) FRØM KEYBØARD
               ;
5B02 21B75D             LXI     H,FBUF     ;PØINT TØ FILE-NAME BUFFER
5B05 0E09               MVI     C,9        ;8-CHARACTER FILE NAME
5B07 CD9D5A    TLD1:    CALL    RFILE      ;INPUT FILE-NAME CHARACTER
5B0A 77                 MØV     M,A        ;PUT IN BUFFER
5B0B 23                 INX     H          ;INCREMENT BUFFER PØINTER
5B0C FE0D               CPI     CR         ;<CR> AT END ØF FILE NAME
5B0E CA185B             JZ      TLD5       ;END ØF FILE NAME
5B11 0D                 DCR     C          ;DECREMENT FILE-NAME CØUNT
5B12 CA8D59             JZ      ERRØR      ;TØØ MANY CHARACTERS
5B15 C3075B             JMP     TLD1       ;NEXT CHARACTER
5B18 32C05D    TLD5:    STA     IBUF       ;PUT CARR RET IN BUFFER
5B1B CD4F59             CALL    LINE       ;ØUTPUT LINE FEED
               ;
               ; LØØK FØR FILE HEADER AT BEGINNING ØF TAPE
               ;
5B1E DB12      TINN:    IN      TAPES      ;CHECK STATUS
5B20 E601               ANI     ACINM      ;TAPE-INPUT MASK
                        IF      JMPZ       ;LØØP ØN ZERØ
5B22 CA1E5B             JZ      TINN
                        ELSE               ;LØØP NØT ZERØ
                        JNZ     TINN       ;LØØP UNTIL READY
                        ENDIF
5B25 DB13               IN      TAPED      ;INPUT A BYTE
5B27 FE55               CPI     SBYTE      ;IS IT A FILE HEADER?
5B29 C21E5B             JNZ     TINN       ;LØØP UNTIL IT IS
5B2C CD315C             CALL    BELL       ;RING BELL AT START
5B2F 21B75D             LXI     H,FBUF     ;PØINT TØ FILE-NAME BUFFER
5B32 7E                 MØV     A,M        ;FETCH FIRST CHAR ØF FILE NAME
```

```
5B33 FE0D             CPI     CR       ;IS IT A CARRIAGE RETURN?
5B35 CA6B5B           JZ      TL0      ;SKIP ØVER FILE NAME (IF ANY)
5B38 11C05D           LXI     D,IBUF   ;PØINT TØ INPUT BUFFER
                 ;
                 ; INPUT FILE NAME AND CØMMENTS FRØM TAPE
                 ;
5B3B CD435C      TLD2:  CALL    TIN      ;INPUT CHARACTER FRØM TAPE
5B3E 12               STAX    D        ;STØRE IN INPUT BUFFER
5B3F 13               INX     D        ;INCREMENT BUFFER
5B40 FE0D             CPI     CR       ;CARR RET AT END ØF FILE NAME
5B42 CA615B           JZ      TLD4     ;END ØF FILE NAME
5B45 FE2C             CPI     ','      ;A CØMMA SEPARATES FILE NAME
5B47 CA575B           JZ      TLD3     ; AND CØMMENTS
5B4A BE               CMP     M        ;SEE IF FILE NAMES MATCH
5B4B 23               INX     H        ;INCREMENT FILE-NAME PØINTER
5B4C CA3B5B           JZ      TLD2     ;NEXT CHARACTER
5B4F 3EFF             MVI     A,255    ;ERRØR, FILE NAMES DØN'T MATCH
5B51 32135D           STA     LFLAG    ;SET ERRØR FLAG
5B54 C33B5B           JMP     TLD2     ;CØNTINUE INPUTTING FILE NAME
5B57 CD435C      TLD3:  CALL    TIN      ;INPUT CØMMENT CHARACTER
5B5A 12               STAX    D        ;STØRE IN INPUT BUFFER
5B5B 13               INX     D        ;INCREMENT BUFFER PØINTER
5B5C FE0D             CPI     CR       ;CARRIAGE RETURN ENDS CØMMENT
5B5E C2575B           JNZ     TLD3     ;NEXT CØMMENT
5B61 AF          TLD4:  XRA     A        ;GET A ZERØ
5B62 1B               DCX     D        ;INCR INPUT BUFFER PØINTER
5B63 12               STAX    D        ;PUT ZERØ AT END ØF BUFFER
5B64 3A135D           LDA     LFLAG    ;FETCH LØAD-ERRØR FLAG
5B67 B7               ØRA     A        ;IS IT ZERØ?
5B68 C27E59           JNZ     FNERR    ;ERRØR IF NØT ZERØ
                 ;
                 ; LØØK FØR RECØRD HEADER ØR END-ØF-FILE BYTE
                 ;
5B6B CD435C      TL0:   CALL    TIN      ;INPUT A BYTE
5B6E FE78             CPI     EØF      ;END ØF FILE?
5B70 CAEF5B           JZ      EXEC     ;YES
5B73 FE74             CPI     EØFC     ;EØF WITH CHECKSUM?
5B75 CAEF5B           JZ      EXEC
5B78 FE77             CPI     LHEAD    ;LØNG RECØRDS?
5B7A CAE15B           JZ      DIN      ;YES
5B7D FE3C             CPI     RHEAD    ;BEGINNING ØF RECØRD?
5B7F C26B5B           JNZ     TL0      ;NØ, TRY AGAIN
                 ;
                 ; RØUTINE TØ INPUT RECØRDS SHØRTER THAN 256 BYTES
                 ;
5B82 CD435C      TLS:   CALL    TIN      ;INPUT RECØRD LENGTH
5B85 4F               MØV     C,A      ;SAVE IT IN C
5B86 B7               ØRA     A        ;RECØRD LENGTH ZERØ?
5B87 CAEB5B           JZ      TL4      ;YES, MITS USES ZERØ FØR 256
5B8A AF               XRA     A        ;GET A ZERØ
5B8B 32125D      TLS2:  STA     RECI     ;ZERØ HIGH BYTE ØF REC LENGTH
5B8E CD945B           CALL    TL2      ;INPUT REST ØF RECØRD
5B91 C36B5B           JMP     TL0      ;NEXT RECØRD
                 ;
                 ; RØUTINE TØ INPUT RECØRD ADDRESS, DATA, AND
                 ; CHECKSUM, AND TEST FØR EØF
                 ;
5B94 2A0D5D      TL2:   LHLD    ØFSET    ;PUT ØFFSET IN H,L
5B97 CD435C           CALL    TIN      ;GET LØW BYTE ØF RECØRD ADDR
5B9A 5F               MØV     E,A      ;SAVE IT IN E
5B9B 47               MØV     B,A      ;START CHECKSUM WITH IT
5B9C CD435C           CALL    TIN      ;GET HIGH BYTE ØF RECØRD ADDR
5B9F 57               MØV     D,A      ;SAVE IT IN D
5BA0 19               DAD     D        ;ADD ØFFSET TØ H,L ADDRESS
5BA1 3A0C5D           LDA     TASK     ;FETCH TASK
5BA4 57               MØV     D,A      ;SAVE IT IN D
5BA5 CD435C      TL1:   CALL    TIN      ;INPUT DATA BYTE
5BA8 5F               MØV     E,A      ;SAVE BYTE
5BA9 7A               MØV     A,D      ;CHECK TASK
5BAA FE56             CPI     'V'      ;SEE IF VERIFYING
5BAC 7B               MØV     A,E      ;RESTØRE DATA BYTE
5BAD CAB15B           JZ      SKIP     ;JUMP IF VERIFYING
5BB0 77               MØV     M,A      ;STØRE DATA IN MEMØRY
5BB1 BE          SKIP:  CMP     M        ;CHECK MEMØRY
5BB2 C26C59           JNZ     MERRØR   ;BAD MEMØRY
5BB5 23               INX     H        ;INCREMENT MEMØRY PØINTER
```

```
5BB6 79                 MØV     A,C     ;GET RECØRD CØUNT (LØW)
5BB7 D601               SUI     1       ;DECREMENT IT
5BB9 4F                 MØV     C,A     ;SAVE IT
5BBA CACE5B             JZ      TL5     ;IF ZERØ, CHECK HIGH HALF
5BBD D2A55B             JNC     TL1     ;CØNTINUE IF NØT 255
5BC0 3A125D             LDA     RECI    ;FETCH RECØRD CØUNT (HIGH)
5BC3 D601               SUI     1       ;DECREMENT IT
5BC5 32125D             STA     RECI    ;SAVE IT
5BC8 DAD55B             JC      TL3     ;END ØF RECØRD IF 255
5BCB C3A55B             JMP     TL1     ;NEXT BYTE
5BCE 3A125D     TL5:    LDA     RECI    ;FETCH RECØRD CØUNT (HIGH)
5BD1 B1                 ØRA     C       ;SEE IF RECØRD CØUNT IS ZERØ
5BD2 C2A55B             JNZ     TL1     ;CØNTIUNE IF NØT
5BD5 48         TL3:    MØV     C,B     ;MØVE SUM TØ C
5BD6 CD435C             CALL    TIN     ;INPUT CHEKSUM
5BD9 B9                 CMP     C       ;CØMPARE TØ SUM
5BDA C8                 RZ              ;RETURN IF ØK
                ;
                ; PRINT A 'C' FØR CHECKSUM ERRØR
                ;
5BDB 11825D     CSERR:  LXI     D,MESC  ;CHECKSUM ERRØR
5BDE C36F59             JMP     TERR2   ;PRINT ERRØR MESSAGE
                ;
                ; RØUTINE TØ INPUT RECØRDS LØNGER THAN 255 BYTES
                ;
5BE1 CD435C     DIN:    CALL    TIN     ;INPUT REC LENGTH (LØW)
5BE4 4F                 MØV     C,A     ;SAVE IT IN C
5BE5 CD435C             CALL    TIN     ;INPUT REC LENGTH (HIGH)
5BE8 C38B5B             JMP     TLS2
                ;
                ; MITS USES A RECØRD LENGTH ØF ZERØ FØR A RECØRD
                ; LENGTH ØF 256.  THIS SUBRØUTINE PUTS A ØNE IN
                ; RECI SØ THAT SUCH TAPES ARE PRØPERLY READ.
                ;
5BEB 3C         TL4:    INR     A       ;INCREMENT RECØRD LENGTH TØ 1
5BEC C38B5B             JMP     TLS2    ;STØRE IN RECI
                ;
                ; END ØF FILE, INPUT AUTØSTART ADDRESS
                ;
5BEF 4F         EXEC:   MØV     C,A     ;SAVE END-ØF-FILE HEADER
5BF0 CD435C             CALL    TIN     ;INPUT LØW BYTE ØF  ADDR.
5BF3 6F                 MØV     L,A     ;PUT INTØ L
5BF4 47                 MØV     B,A     ;START CHECKSUM WITH L
5BF5 CD435C             CALL    TIN     ;GET HIGH BYTE ØF AUTØSTART
5BF8 67                 MØV     H,A     ;PUT INTØ H
5BF9 79                 MØV     A,C     ;GET END-ØF-FILE HEADER
5BFA FE74               CPI     EØFC    ;CHECKSUMMED?
5BFC C2075C             JNZ     EXEC3   ;JUMP IF NØ CHECKSUM
5BFF 48                 MØV     C,B     ;PUT CHECKSUM IN C
5C00 CD435C             CALL    TIN     ;INPUT CHECKSUM BYTE
5C03 B9                 CMP     C       ;CØMPARE TØ SUM ØF H AND L
5C04 C2DB5B             JNZ     CSERR   ;JUMP IF ERRØR
5C07 7A         EXEC3:  MØV     A,D     ;CHECK TASK
5C08 FE45               CPI     'E'     ;SEE IF EXECUTING
5C0A CAE658             JZ      JPCHL   ;YES, GØ TØ H/L
                ;                        NØT EXECUTING
5C0D 11C05D             LXI     D,IBUF  ;PØINT TØ FILE NAME
5C10 1A                 LDAX    D       ;FETCH FIRST CHARACTER
5C11 FE0D               CPI     CR      ;IS IT A CARRIAGE RETURN?
5C13 CA1C5C             JZ      EXEC2   ;SKIP FILE NAME IF SØ
5C16 CD005D             CALL    SENDM   ;PRINT FILE NAME, CØMMENTS
5C19 CD6759             CALL    BLANK   ;PRINT A BLANK
5C1C 11955D     EXEC2:  LXI     D,MESE  ;PØINT TØ 'STARTS AT'
5C1F C36F59             JMP     TERR2   ;PRINT IT

                ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                ;
                ; ENTRY TØ SKIP ØVER MITS CHECKSUM LØADER
                ; AT BEGINNNING ØF TAPE
                ;
5C22 57         CLØADR: MØV     D,A     ;SAVE TASK
5C23 CD435C     CLD2:   CALL    TIN     ;INPUT A BYTE
5C26 FEF3               CPI     0F3H    ;CHECK FØR DI ØPCØDE
5C28 C2235C             JNZ     CLD2    ;NØT YET, NEXT BYTE
5C2B CD315C             CALL    BELL    ;RING BELL TØ INDICATE
```

```
                                                ;END ØF CHECKSUM LØADER
5C2E C36B5B                 JMP      TLO        ;START MAIN PRØGRAM
                  ;
                  ; RØUTINE TØ RING THE BELL
                  ;
5C31 3E07         BELL:     MVI      A,7
5C33 C35B59                 JMP      ØUTT
                  ;
                  ; SUBRØUTINE TØ ØUTPUT A LEADER ØF NULLS FØR
                  ; A LEADER AND A TRAILER
                  ;
5C36 3A145D       LEADR:    LDA      NNUL       ;FETCH NUMBER ØF NULLS
5C39 4F                     MØV      C,A        ;PUT IN C
5C3A AF                     XRA      A          ;GET A NULL
5C3B CD515C       LEAD2:    CALL     TØUT       ;ØUTPUT A NULL
5C3E 0D                     DCR      C          ;DECREMENT CØUNT
5C3F C23B5C                 JNZ      LEAD2      ;NEXT NULL
5C42 C9                     RET
                  ;
                  ; SUBRØUTINE TØ INPUT A BYTE FRØM TAPE
                  ; AND ADD TØ CHECKSUM
                  ;
5C43 DB12         TIN:      IN       TAPES      ;CHECK STATUS
5C45 E601                   ANI      ACINM      ;TAPE-INPUT MASK
                            IF       JMPZ       ;LØØP ØN ZERØ
5C47 CA435C                 JZ       TIN
                            ELSE                ;LØØP NØT ZERØ
                            JNZ      TIN        ;LØØP IF NØT READY
                            ENDIF
5C4A DB13                   IN       TAPED      ;INPUT A BYTE
5C4C F5                     PUSH     PSW
5C4D 80                     ADD      B          ;ADD BYTE TØ CHECKSUM
5C4E 47                     MØV      B,A        ;SAVE CHECKSUM IN B
5C4F F1                     PØP      PSW
5C50 C9                     RET
                  ;
                  ; ØUTPUT A BYTE TØ TAPE AND ADD TØ CHECKSUM
                  ;
5C51 F5           TØUT:     PUSH     PSW
5C52 80                     ADD      B          ;ADD TØ CHECKSUM
5C53 47                     MØV      B,A        ;SAVE IT IN B
5C54 DB12         TØUT1:    IN       TAPES      ;CHECK STATUS
5C56 E602                   ANI      ACØM       ;TAPE-ØUTPUT MASK
                            IF       JMPZ       ;LØØP ØN ZERØ
5C58 CA545C                 JZ       TØUT1
                            ELSE                ;LØØP NØT ZERØ
                            JNZ      TØUT1      ;LØØP IF NØT READY
                            ENDIF
5C5B F1                     PØP      PSW
5C5C D313                   ØUT      TAPED      ;ØUTPUT A BYTE
5C5E C9                     RET
                  ;
                  ; PRINT THE H/L REGISTER PAIR IN HEX
                  ;
5C5F 4C           ØUTHL:    MØV      C,H        ;FETCH H
5C60 CD645C                 CALL     ØUTHEX     ;PRINT IT
5C63 4D                     MØV      C,L        ;FETCH L, PRINT IT
                  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                  ;
                  ; SUBRØUTINE TØ CØNVERT A BINARY NUMBER IN C
                  ; TØ TWØ ASCII HEX CHARACTERS, AND PRINT THEM
                  ;
5C64 79           ØUTHEX:   MØV      A,C
5C65 1F                     RAR                 ;RØTATE UPPER
5C66 1F                     RAR
5C67 1F                     RAR                 ;CHARACTER
5C68 1F                     RAR                 ;TØ LØWER
5C69 CD6D5C                 CALL     HEX1       ;ØUTPUT UPPER CHARACTER
5C6C 79                     MØV      A,C        ;ØUTPUT LØWER CHARACTER
                  ;
                  ; SUBRØUTINE TØ ØUTPUT A HEX CHARACTER
                  ; FRØM THE LØWER FØUR BITS
                  ;
5C6D E60F         HEX1:     ANI      0FH        ;MASK UPPER 4 BITS
5C6F C690                   ADI      90H
```

```
5C71 27                 DAA               ;INTEL DAA TRICK
5C72 CE40               ACI      40H
5C74 27                 DAA               ;ØNCE AGAIN
5C75 C35B59             JMP      ØUTT
                 ;
                 ; SUBRØUTINE TØ CØNVERT TWØ KEYBØARD
                 ; HEX CHARACTERS TØ ØNE BINARY BYTE IN B
                 ;
5C78 CD865C      RDHEX:  CALL     HEX2      ;INPUT UPPER CHARACTER
5C7B 07          RDHX2:  RLC               ;RØTATE TØ
5C7C 07                 RLC
5C7D 07                 RLC               ;UPPER HALF
5C7E 07                 RLC
5C7F 47                 MØV      B,A      ;SAVE IT IN B
5C80 CD865C             CALL     HEX2      ;INPUT LØWER CHARACTER
5C83 80                 ADD      B        ;CØMBINE BØTH PARTS
5C84 47                 MØV      B,A      ;SAVE BØTH IN B
5C85 C9                 RET
5C86 CD2C59      HEX2:   CALL     READ      ;INPUT FRØM KEYBØARD
5C89 D630        HEX22:  SUI      '0'      ;SUBTRACT ASCII BIAS
5C8B DA8D59             JC       ERRØR     ;ERRØR, LESS THAN '0'
5C8E FE17               CPI      23
5C90 D28D59             JNC      ERRØR     ;ERRØR, GREATER THAN 'F'
5C93 FE0A               CPI      10
5C95 D8                 RC                ;A NUMBER 0-9
5C96 D607               SUI      7
5C98 FE0A               CPI      10
5C9A DA8D59             JC       ERRØR     ;ERRØR, BETWEEN 9-A
5C9D C9                 RET               ;A CHARACTER A-F
                 ;
                 ; SUBRØUTINE TØ CHECK HEX/ØCTAL FLAG AND JUMP
                 ; TØ PRØPER INPUT RØUTINE
                 ;
5C9E 3A0A5D      INHL:   LDA      HEXFL     ;FETCH HEX/ØCTAL FLAG
5CA1 B7                 ØRA      A        ;CHECK FØR ZERØ
5CA2 C2B25C             JNZ      RDHLØ     ;ØCTAL INPUT
                 ;
                 ; SUBRØUTINE TØ INPUT H,L FRØM KEYBØARD (HEX FØRMAT)
                 ;
5CA5 CD785C      READHL: CALL     RDHEX     ;READ HIGH BYTE
5CA8 60          RHL2:   MØV      H,B      ;PUT IT IN H
5CA9 CD785C             CALL     RDHEX     ;INPUT LØW BYTE
5CAC 68          RDHL2:  MØV      L,B      ;PUT IT IN L
                 ;
5CAD 3E3A        CØLØN:  MVI      A,':'    ;ØUTPUT A CØLØN TØ
5CAF C35B59             JMP      ØUTT      ; PRINTER
                 ;
                 ; SUBRØUTINE TØ INPUT H/L FRØM KEYBØARD (ØCTAL)
                 ;
5CB2 CDD85C      RDHLØ:  CALL     RDØCT     ;INPUT HIGH HALF ØF ADDRESS
5CB5 60          RHLØ2:  MØV      H,B      ;PUT INTØ H
5CB6 CDD85C             CALL     RDØCT     ;INPUT LØW HALF ØF ADDRESS
5CB9 C3AC5C             JMP      RDHL2     ;CØNTINUE IN HEX RØUTINE
                 ;
                 ; SUBRØUTINE TØ PRINT THE H/L REGISTER PAIR IN ØCTAL
                 ;
5CBC 4C          ØUTHLØ: MØV      C,H      ;FETCH H
5CBD CDC15C             CALL     ØUTØCT    ;PRINT IT
5CC0 4D                 MØV      C,L      ;FETCH L
                 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                 ;
                 ; SUBRØUTINE TØ CØNVERT A BINARY NUMBER IN C
                 ; TØ THREE ASCII ØCTAL CHARACTERS AND PRINT THEM
                 ;
5CC1 79          ØUTØCT: MØV      A,C
5CC2 07                 RLC               ;RØTATE LEFT TWØ BITS
5CC3 07                 RLC               ; TØ BØTTØM
5CC4 E603               ANI      3        ;SELECT BØTTØM TWØ BITS
5CC6 CDD35C             CALL     ØUT60     ;ØUTPUT LEFT CHARACTER
5CC9 79                 MØV      A,C
5CCA 0F                 RRC               ;RØTATE MIDDLE
5CCB 0F                 RRC               ; BITS TØ
5CCC 0F                 RRC               ; BØTTTØM
5CCD CDD15C             CALL     ØUTØ      ;ØUTPUT CENTER CHARACTER
5CD0 79                 MØV      A,C      ;ØUTPUT RIGHT CHARACTER
```

```
5CD1 E607      ØUTØ:    ANI      7         ;SELECT RIGHT THREE BITS
5CD3 C630      ØUT60:   ADI      60Q       ;ADD ASCII BIAS
5CD5 C35B59             JMP      ØUTT      ;PRINT CHARACTER
               ;
               ; SUBRØUTINE TØ CØNVERT THREE KEYBØARD ASCII
               ; ØCTAL DIGITS TØ ØNE BINARY BYTE IN E
               ;
5CD8 CDF25C    RDØCT:   CALL     ØCTIN     ;INPUT FIRST CHARACTER
5CDB FE04      RØCT2:   CPI      4         ;>4?
5CDD D28D59             JNC      ERRØR     ;YES, ERRØR
5CE0 87                 ADD      A         ;SHIFT
5CE1 87                 ADD      A         ; TØ THE LEFT
5CE2 87                 ADD      A         ; THREE BITS
5CE3 47                 MØV      B,A       ;SAVE FIRST PART IN B
5CE4 CDF25C             CALL     ØCTIN     ;INPUT SECØND CHARACTER
5CE7 B0                 ØRA      B         ;CØMBINE WITH FIRST PART
5CE8 87                 ADD      A         ;SHIFT
5CE9 87                 ADD      A         ; THREE BITS
5CEA 87                 ADD      A         ; LEFT
5CEB 47                 MØV      E,A       ;SAVE IN B
5CEC CDF25C             CALL     ØCTIN     ;INPUT THIRD CHARACTER
5CEF B0                 ØRA      E         ;CØMBINE ALL THREE
5CF0 47                 MØV      B,A       ;SAVE BYTE IN B
5CF1 C9                 RET
               ;
               ; SUBRØUTINE TØ INPUT AN ØCTAL CHARACTER TØ A
               ;
5CF2 CD2C59    ØCTIN:   CALL     READ      ;INPUT FRØM KEYBØARD
5CF5 D630      ØCTI2:   SUI      '0'       ;SUBTRACT ASCII BIAS
5CF7 DA8D59             JC       ERRØR     ;ERRØR, LESS THAN 'Ø'
5CFA FE08               CPI      8         ;CØMPARE TØ 8
5CFC D28D59             JNC      ERRØR     ;ERRØR, GREATER THAN 7
5CFF C9                 RET
               ;
               ; SUBRØUTINE TØ PRINT AN ASCII MESSAGE PØINTED TØ
               ; BY D,E.  STØPS WHEN A BINARY ZERØ IS FØUND.
               ;


5D00 1A        SENDM:   LDAX     D         ;FETCH CHARACTER
5D01 13                 INX      D         ;INCREMENT PØINTER
5D02 B7                 ØRA      A         ;IS CHAR A BINARY ZERØ?
5D03 C8                 RZ                 ;RETURN IF IT IS
5D04 CD5B59             CALL     ØUTT      ;ØTHERWISE PRINT IT
5D07 C3005D             JMP      SENDM     ;NEXT CHARACTER
               ;
5D0A 00        HEXFL:   DB       0         ;HEX/ØCTAL MØDE FLAG, 0 = HEX
5D0B 00        SFLAG:   DB       0         ;INITIALIZATIØN FLAG
5D0C           TASK:    DS       1         ;SAVE TASK HERE
5D0D 0000      ØFSET:   DW       0         ;ØFFSET VECTØR FØR LØAD
5D0F FF        RECLN:   DB       RLEN      ;RECØRD LENGTH (LØW BYTE)
5D10 00        RECL2:   DB       0         ;RECØRD LENGTH (HIGH BYTE)
5D11 00        RECA:    DB       0         ;RECØRD-LENGTH CØUNT (INPUT)
5D12 00        RECI:    DB       0         ;INPUT REC-LENGTH CØUNT (HIGH)
5D13 00        LFLAG:   DB       0         ;LØAD-ERRØR FLAG
5D14 01        NNUL:    DB       1         ;NUMBER ØF NULLS ØN LEADER
5D15 484558204FMESG:    DB       'HEX ØR ØCTAL INPUT? ',0
5D2A 2048455800MES1:    DB       ' HEX',0
5D2F 204F435441MES2:    DB       ' ØCTAL',0
5D36 20494E5055MES3:    DB       ' INPUT'
5D3C 0D0A020202         DB       CR,LF,2,2,2
5D41 5245434F52MES4:    DB       'RECØRD LENGTH',0
5D4F 3F2000             DB       '? ',0
5D52 2032353500MESR:    DB       ' 255',0
5D57 2020574841MESW:    DB       '  WHAT? ',0  ;ERRØR MESSAGE
5D5F 4D454D4F52MESM:    DB       'MEMØRY ERRØR AT ',0
5D70 20204C4541MESN:    DB       '  LEADER LENGTH? ',0
5D82 434845434BMESC:    DB       'CHECKSUM ERRØR AT ',0
5D95 5354415254MESE:    DB       'STARTS AT ',0
5DA0 57524F4E47MESF:    DB       'WRØNG FILE NAME, TRY: ',0
5DB7           FBUF:    DS       9         ;FILE-NAME BUFFER
5DC0           IBUF:    DS       BUFL      ;INPUT BUFFER
5DD2                    DS       24        ;SPACE FØR STACK
5DEA =         STACK    EQU      $         ;TØP ØF STACK
5DEA                    END
```

```
CP/M MACRØ ASSEM 2.0     #019     BINARY TAPE MØNITØR

0001 ACINM       0002 ACØM        0008 BACKUP      5C31 BELL
5967 BLANK       0012 BUFL        5A90 CEND        5C23 CLD2
5C22 CLØADR      5CAD CØLØN       000D CR          594A CRLF
5BDB CSERR       007F DEL         5BE1 DIN         5A68 DØUB
5A7D DØUBF       0074 EØFC        0078 EØF         598D ERRØR
5BEF EXEC        5C1C EXEC2       5C07 EXEC3       0000 FALSE
5DB7 FBUF        597E FNERR       58E3 GØ2         593F GØ
5C6D HEX1        5C89 HEX22       5C86 HEX2        5D0A HEXFL
5DC0 IBUF        5C9E INHL        5826 INIT0       582E INIT1
5834 INIT3       5845 INIT5       588B INIT6       587F INITH
0001 INMASK      FFFF JMPZ        58E6 JPCHL       5C3B LEAD2
5C36 LEADR       000A LF          5D13 LFLAC       0077 LHEAD
594F LINE        596C MERRØR      5D2A MES1        5D2F MES2
5D36 MES3        5D41 MES4        5D82 MESC        5D95 MESE
5DA0 MESF        5D15 MESG        5D5F MESM        5D70 MESN
5D52 MESR        5D57 MESW        58E7 MØDE        58F7 MØDE1
58FD MØDE2       5A0E NEW2        5D14 NNUL        5CF5 ØCTI2
5CF2 ØCTIN       5AEA ØFF1        5AFB ØFF2        5ADD ØFF3
5AD7 ØFF4        5AE0 ØFF5        5ABB ØFFST       5D0D ØFSET
5CD3 ØUT60       5C64 ØUTHEX      5C5F ØUTHL       5CBC ØUTHLØ
0002 ØUTMSK      5CD1 ØUTØ        5CC1 ØUTØCT      595B ØUTT
5864 RDCR        5C78 RDHEX       5CAC RDHL2       5CB2 RDHLØ
5C7B RDHX2       5CD8 RDØCT       592C READ        5CA5 READHL
5D11 RECA        5D12 RECI        5D10 RECL2       5D0F RECLN
5893 RESTRT      5A9D RFILE       003C RHEAD       5CA8 RHL2
5CB5 RHLØ2       00FF RLEN        5CDB RØCT2       0055 SBYTE
5D00 SENDM       590C SETN        591F SETN2       5922 SETN3
5D0B SFLAG       5BB1 SKIP        003C SNUL        5DEA STACK
5800 START       0013 TAPED       0012 TAPES       5D0C TASK
59F7 TD0         5A21 TD1         5A45 TD2         5A18 TD3
5A4E TD4         5A3E TD5         59B1 TDMP3       59E5 TDMP4
59D7 TDMP5       59C7 TDMP6       59C9 TDMP7       59CD TDMP8
599D TDUMP       596F TERR2       5972 TERR3       5C43 TIN
5B1E TINN        5B6B TL0         5BA5 TL1         5B94 TL2
5BD5 TL3         5BEB TL4         5BCE TL5         5B07 TLD1
5B3B TLD2        5B57 TLD3        5B61 TLD4        5B18 TLD5
5AFF TLØAD       5B82 TLS         5B8B TLS2        5C51 TØUT
5C54 TØUT1       FFFF TRUE        0011 TYDATA      0010 TYSTAT
595C WAITØ
```

Chapter 5

Complete Data Base Management System

By Peter Reece

INTRODUCTION

One of the most common uses of a computer is the manipulation of large amounts of data in a utilitarian task-determined manner. That is, by selective manipulation or scanning of knowledge bases, the computer can yield rapid summary information which is representative of the complete data base. This manipulation, commonly known as data base management, is unfortunately usually relegated to the large computer only. The small system user who wishes to organize office inventory, book lists, mailing labels, and the like, is usually left out in the cold.

The IDMAS (Interactive Data Base Manipulator And Summarizer) system is a remedy to the problems of the small user who wishes large system features. IDMAS allows the user to selectively scan, summarize, total, count, change, enter, delete, and encode data from a data base without any knowledge of the internal workings of the computer or program. Commands are interpreted through an English language parser which can be modified or augmented very easily to enable the user to utilize the English language subset he prefers. There may be as many data bases on the system as the user's facility allows. In addition, while the program is included here in direct access disk mode (available in most micro-floppy software) modification to a sequential access file structure is straightforward.

IDMAS is written in PDP-10 BASIC—a BASIC dialect which is widely accepted (see IDMAS listing). If the error message and ASCII code in the program is minimized for a given application, the entire source will fit into about 12K of core on the average small computer. Further, since the various commands are modular, they may be removed at will if not required in a given application, in order to drain core even less.

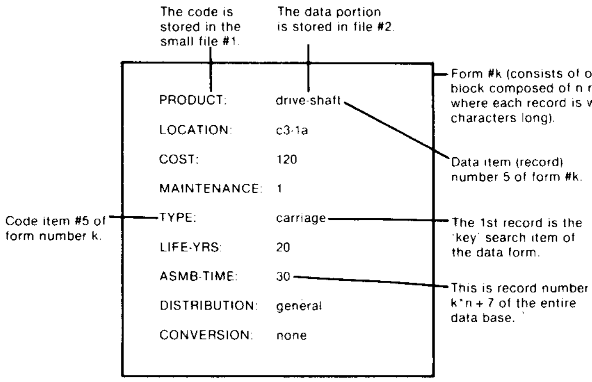

The code is stored in the small file #1.
The data portion is stored in file #2.
PRODUCT: drive-shaft
LOCATION: c3-1a
COST: 120
MAINTENANCE: 1
TYPE: carriage
LIFE-YRS: 20
ASMB-TIME: 30
DISTRIBUTION: general
CONVERSION: none
Code item #5 of form number k.
Form #k (consists of one block composed of n records, where each record is w2 characters long).
Data item (record) number 5 of form #k.
The 1st record is the 'key' search item of the data form.
This is record number k*n + 7 of the entire data base.

Figure 1. A small annotated sample form from an assembly line data base.

FILE STRUCTURE

There are two data files for every data base used by IDMAS. The first is a small file containing the codes or "names" assigned by the user to the various items in his data base. The second file contains the actual data which corresponds to these names. This is the larger of the two files. For example, suppose that the user is concerned with the length of time it takes to assemble some automotive components and wishes to store this information in a data base. As Figure 1 shows, the time, say 30 minutes, would be stored as data in file two while the name of that data—asmb-time—would be stored in file one.

Each item of data and its name are stored in a single record in their respective files. These records are always the same distance from the start of this block of data. Thus, in our example, asmb-time is stored at the seventh record of the block, and "30" is stored at the seventh record of the block of file two.

A block consists of all records which correspond to a given item in the data which is of particular importance. This is the "key" item, and is usually the main item of interest in the data base. If we think of a data base consisting of mailing lists, the key item might be postal zone since it is the most general designation. The next most important item might be state, then county, city, street, and so on. The key item would be the first item in the hierarchy. All records following the key will in some way be tied to the key. (Note that IDMAS does not require the key item be the main item in a block, but from a user standpoint, and simply by convention, one item is usually designated as the key.)

A "form" consists of the key plus its related records. There may be one and only one code form per data base in IDMAS. This is because all data in file two is assigned names through the code form of file one. More code forms would lead to confusion, and the program automatically prevents the occurrence of two code forms. The number of different codes per data base, however, is unlimited. A "form," then relates to the total structure enclosed in the box in Figure 1, while a "block" refers to the physical grouping of the records in the data file. (That is, "block" takes on the common meaning of data block on a disk or tape file.) The user may choose the length of each record within a block by adjusting the parameter "w2." Hence, if w2 = 80, each record in file two, the data file, will be 80 characters in length. (Blanks are added if all 80 are not used.)

In performing a search, the program computes the length of a form from the number of names in the list of codes in file one and adds to this the distance of the user selected record from the key item. For example, to search the data base for all assembly times, as in Figure 1, the program would read records 7, 16, 25, etc. In this way, only records relevant to the search are read, thereby saving considerable read time.

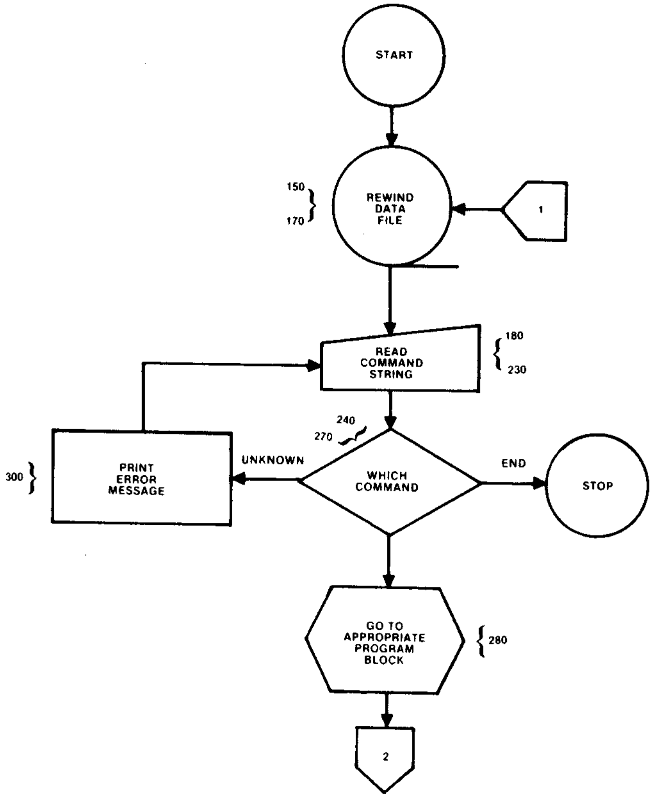


Figure 2. The initializing loop of the program (numbers in single brackets indicate line numbers in the program).

THE SEARCH

The actual mechanics of this search is illustrated in flow form in Figure 3. After deciding that a search has been requested (see Figure 2) the program reads the code table to compute the validity of the names requested by the user input string. Such an input string is illustrated in the example in the first three lines of Figure 6.

Once it has been determined that the input string contains valid requests, the search is begun. Each search item of the input string is compared to the information residing in the appropriate record as computed by the method already outlined. For example, as illustrated in Figure 6, three records would be read per form, and the information in the seventh record, for example, would be checked to see if it is greater than fifteen. Totals, detailed counts, etc. (see command list outlined below) would be performed once it had been determined that all of the requested information in this form had matched the user's input specifications (e.g., that there was not a cost of 10 in the form of the example in Figure 6).

In this manner, the search would continue until each form in the data file had been read. A summary according to the user's request and previous commands would then be printed, and the program would await the next command.

THE PARSER

IDMAS contains a table driven parser which is capable of encoding English language user input into a form usable by the rest of the program. The table consists of verbs and verb phrases, as well as "noise words"—articles and adjectives which are used in English but are of no use in the search. First the parser scans the input string searching for a match between elements in the string and the table. When such a match is found, the matched characters in the string are replaced by the appropriate verb code from the table. For example, in line 510 of the program, it can be seen that the phrase "is not" is replaced by the code ".not.". That is, the table is composed of pairs—the first word in the pair is the match item, the second is the replacement item. If the replacement item is a "9", the parser automatically replaces the match item in the input string with blanks.

Note that the table may contain any verbs or phrases which the user deems appropriate to his task. Hence, the input string has considerable flexibility. The table may also be as long or short as the user wishes.

The parser's next step is to remove the blanks from the input string. At this point, the input string will consist only of subjects, verbs, objects, and connectors. For example, if the input string is:

```
FIND AN ANSMB-TIME WHICH IS MORE THAN 15*t AND
WHOSE TYPE IS NOT A CARRIAGE AND WHICH SHOULD
HAVE A COST OF 100.
```

then at this point in the parsing procedure, the input string would appear as:

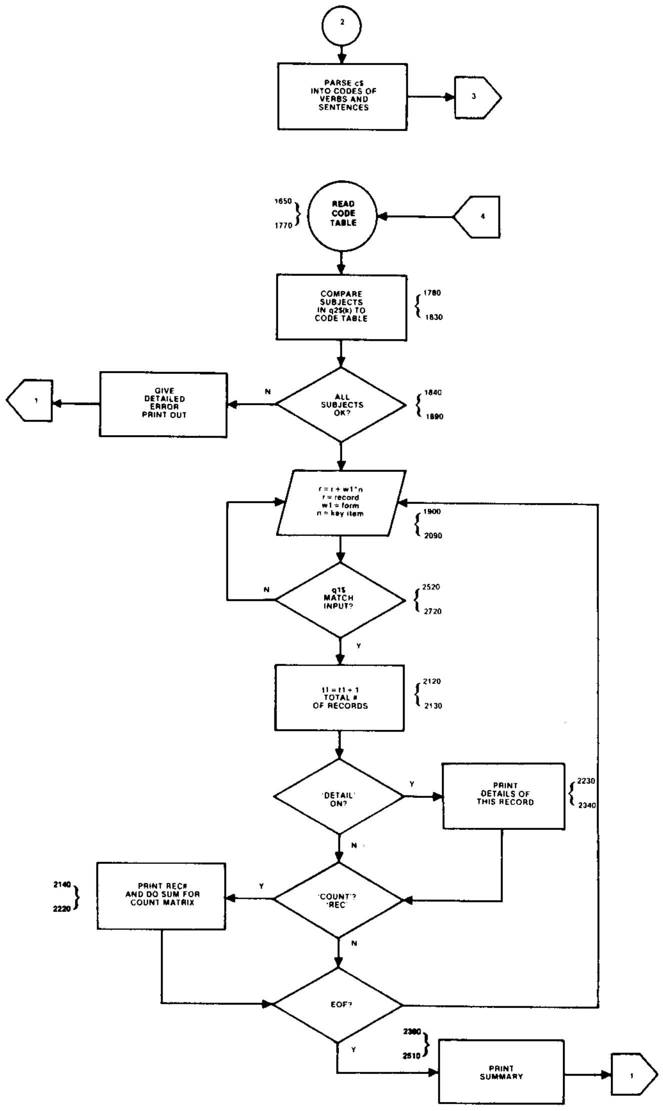


Figure 3. The search loop.

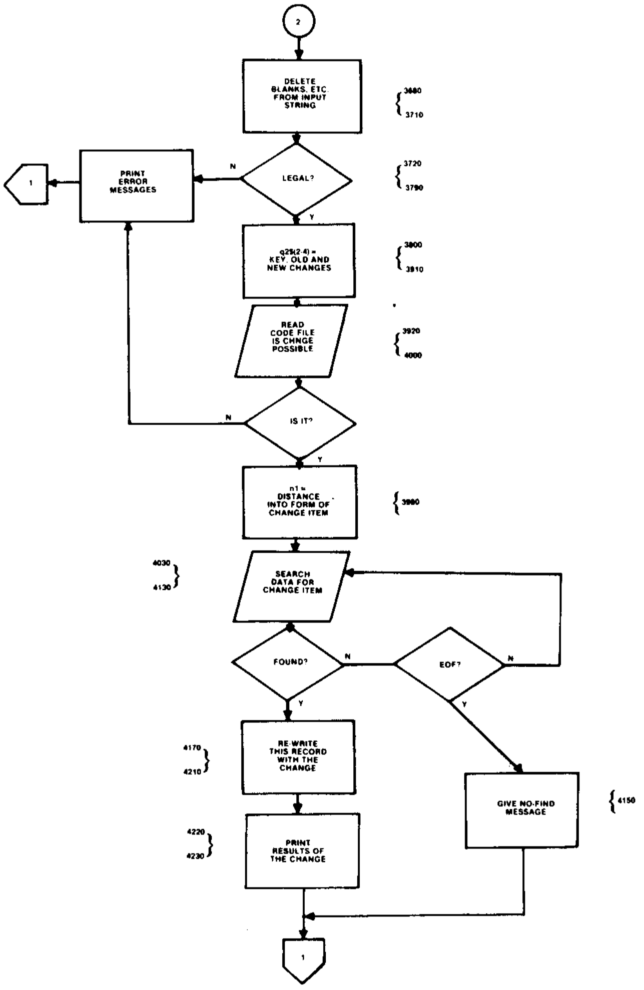

2
DELETE BLANKS, ETC. FROM INPUT STRING
3680 3710
LEGAL?
N
Y
3720 3790
PRINT ERROR MESSAGES
1
q2$(2-4) = KEY, OLD AND NEW CHANGES
3800 3910
READ CODE FILE IS CHNGE POSSIBLE
3920 4000
IS IT?
N
Y
n1 = DISTANCE INTO FORM OF CHANGE ITEM
3990
4030 4130
SEARCH DATA FOR CHANGE ITEM
FOUND?
N
Y
EOF?
N
Y
4170 4210
RE-WRITE THIS RECORD WITH THE CHANGE
GIVE NO-FIND MESSAGE
4150
4220 4230
PRINT RESULTS OF THE CHANGE
1

Figure 4. The 'change' command.

ASMB-TIME.GT.15*tANDTYPE.NOT.CARRIAGEAND
COST.IS.100

The parser now creates a number of matrices which categorize the information in the above string. First the matrix q$(k) is created, where each "k" contains one of the sentences in the input string. The matrix q1(k) is then created and consists of numbers which represent the verbs of the input string. The objects of each sentence—that is, the sought items in the data—are contained in the matrix q1$(k). Finally,

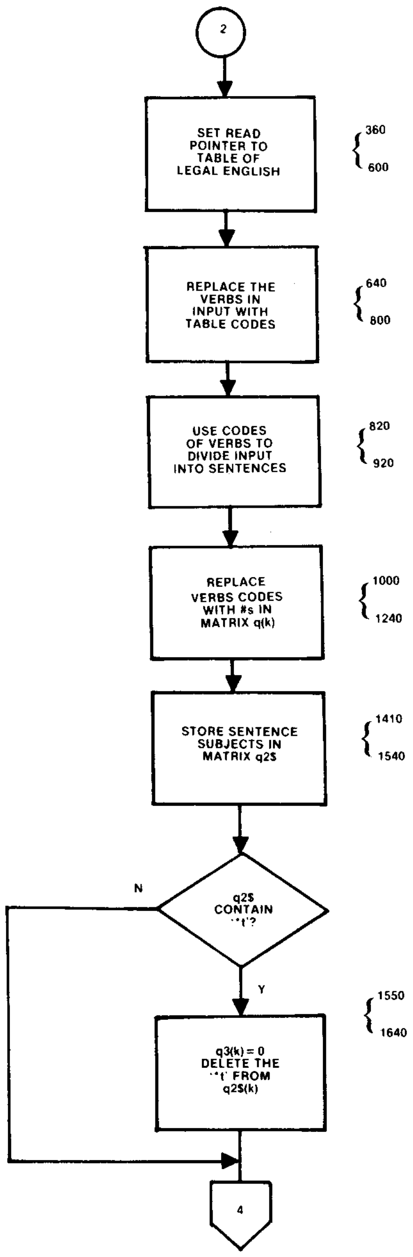
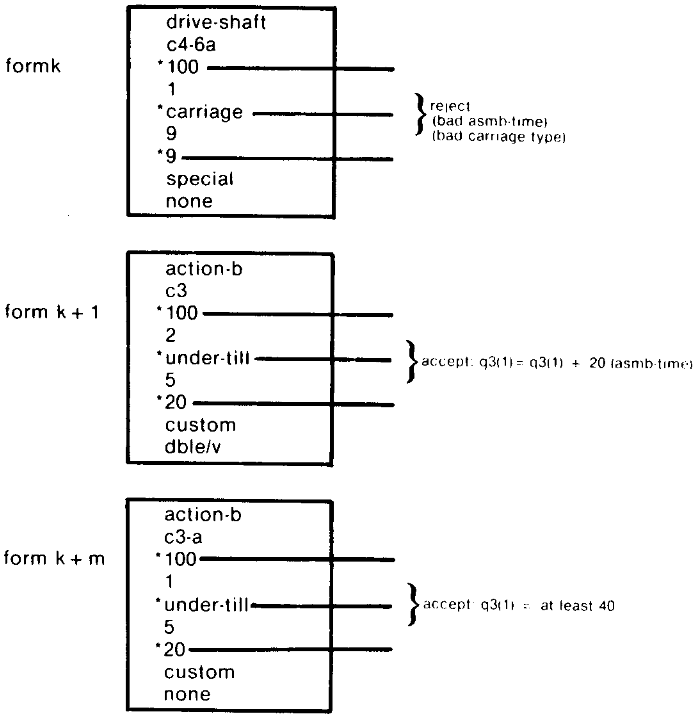

2
SET READ POINTER TO TABLE OF LEGAL ENGLISH
360
600
REPLACE THE VERBS IN INPUT WITH TABLE CODES
640
800
USE CODES OF VERBS TO DIVIDE INPUT INTO SENTENCES
820
920
REPLACE VERBS CODES WITH #s IN MATRIX q(k)
1000
1240
STORE SENTENCE SUBJECTS IN MATRIX q2$
1410
1540
q2$ CONTAIN "t'?
N
Y
q3(k) = 0 DELETE THE "t' FROM q2$(k)
1550
1640
4

Figure 5. Flow of the parser.

FIND AN ASMB-TIME WHICH IS MORE THAN 15*t AND WHOSE TYPE IS NOT A CARRIAGE AND WHICH SHOULD HAVE A COST OF 100.



*The program computes which records are of use and reads *only* those records — thereby speeding search time considerably.

The minimum output (i.e. if no additional forms where accepted, and no special program switches where in effect) would be:

```
I HAVE FOUND 2 ITEMS.
TOTAL ASMB-TIME = 40.
```

Figure 6. Example of a simple search through data.

q2$(k) is loaded with the subjects of the sentences. To illustrate all of this, using the above example, we might have the following:

	q1(k)	q1$(k)	q2$(k)	q$(k)
K=1	3	15	assmb-time	assmb-time.gt.15
K=2	2	carriage	type	type.not.carriage
K=3	1	100	cost	cost.is.100

The final task of the parser is to assign a subject to sentences which lack one. This is done by assuming that the subject of the previous sentence of the input string also applies to the present sentence. If no previous message existed, an error message is generated.

At this time too, the matrix q3(k) is loaded with a one or a zero per k according to whether or not a total is desired for objects in q1$(k). For example, in the input string illustrated above, q3(1) = 0, q3(2) = 1, q3(3) = 1, since a total of assembly times only was requested.

The general flow of the parser, as well as the program line numbers which correspond to the various tasks, is illustrated in Figure 5.

COMMANDS

Below is a listing of command words accepted by IDMAS, with a description of each. Each command may be abbreviated to the first three letters if desired.

HELP: prints a list of commands.

DELETE: Deletes a particular form from the data base. Deletions are always by key item. For example, using the illustration in Figure 6, typing "DELETE ACTION-B" would delete form number k + 1. To delete form K + m, "DELETE ACTION—B" would have to be typed again. This is a safety feature in the event that duplicte key items exist (it is up to the user whether or not his key items are unique).

Suppose that form k + 1 only had been deleted. The next time the "ADD" command was issued by the user (see below) the space left empty by form k + 1 would be filled. In this way IDMAS Prevents the existence of holes in the data base, minimizing storage cost.

ADD: To add a form to the data base, the user need type only the word "add" plus the name of the key item plus the data for the key item. For example, to add the product "mud" to the data base, type "ADD PRODUCT MUD". IDMAS would then prompt the user for the rest of the items in the form, e.g. the value of "location", "cost", and so on.

A switch exists in the program (see "add" subroutine) which allows the user to have multiple key items of identical value should he wish to do so. Otherwise the program will automatically produce an error message if the user attempts to create forms with non-unique keys. This feature therefore allows maximum versatility in the use of non- or pure hierarchical data base form structures.

DONE: This command will terminate the action of any other command which is waiting for input. For example, if halfway through "add"ing a form the user decides that he doesn't wish to "add" this form after all, typing "done" will terminate the add command and return the user to his initial state prior to the "add" command.

CREATE: To create a new data base, simply type "create". IDMAS will then prompt for names of code items and internally assigned files one and two for future use by this data base.

CODE: This command allows the user to scan raw data as it actually exists in his data file by naming the specific records he wishes to see. For example, if the user wishes to scan the contents of records 200-204, he need simply type "code". IDMAS will then prompt for the first and last record desired (200-204, respectively), then type the contents of these records in their actual stored state.

RECORD: Typing "record" activates a program switch which causes the record numbers of all key items activated during a search (assuming the form matches the search specifications) to be printed.

NORECORD: Disables the "record" switch.

SHOW: This causes the chosen form to be printed as in Figure 1. For example, to print the form whose key item is "mud", type "SHOW PRODUCT MUD".

CHANGE: Figure 4 illustrates the flow of the change command. Suppose, for example, that the user wished to change the "type" in Figure 1 from "carriage" to "unit/a". To do this he need simply type "CHANGE TYPE IN PRODUCT = DRIVE – SHAFT TO UNIT/A". The program would then search the data base for the key item named, then perform the change. If the search and change were successful, IDMAS would type:

> "TYPE CHANGED FROM CARRIAGE TO UNIT/A in PRODUCT DRIVE—SHAFT".

TOTAL: This enables the facility whereby user designated *numeric* items may be summed together during a search (assuming the conditions of a search are met). The user specifies which items he wishes IDMAS to sum by adding "*t" following the object(s) of a sentence or sentences. Figure 6 illustrates an example of the use of the "total" command. As many objects as the user wishes may be totaled per input string.

REASSIGN: This allows the user to end (i.e., close) the data base he is presently working with and proceed to a different data base. IDMAS Prompts the user for the appropriate information.

COUNT: This command causes the total number of items found during a search to be printed, regardless of the status of any other commands.

NOCOUNT: Disables the "count" command.

DISPLAY: If the user wishes to see the actual values of the items which meet the conditions of a search, he must type "display" prior to conducting the search.

NODISPLAY: Cancels the "display" switch.

WORD: Prints the first word of the word pairs in the English translation table used by the parser. In other words, this command lists the words and phrases which are accepted by the parser.

LENGTH: This command causes the total number of records in the data file of the active data base to be printed.

SPECIAL: The "special" command allows the user to call a program which he has written as though it were a subroutine of the IDMAS Program.

Typing "special" causes the program to print a description of the use of the special command. Typing "special progx" causes IDMAS to execute the user program "progx" as a subroutine.

FIND: The general format of the command to enable a data base search is the word "find", "suggest", "can", or "match" followed by sentences joined by the Boolian connective "and".

A sample format might appear as follows:

> FIND subject1 verb object1 AND article subject2 verb phrase object 2. Or, using the form of Figure 1: FIND type is carriage AND a cost which isn't 120.

Searches are performed, as explained when the parser was discussed above, by treating all the words between the "ands" as separate sentences. A search is successful if and only if a given form meets the conditions specified by *each* sentence in the input string. In other words, the "AND" is a logical, or Boolian "and".

All searches will print as a minimum the total number of forms that met all of the conditions of the search, if other switches have not been set.

CONCLUSION

The appendix gives several examples of the use of IDMAS with a very simple data base. It is hoped that by adapting IDMAS to your system, and taking advantage of its versatility and flexibility you will find that the task of data manipulation becomes easier, more useful and enjoyable.

PROGRAM LISTING

```
10 REM *******************************************************
12 REM *                                                     *
14 REM *        IDMAS: A DATA BASE MANAGEMENT SYSTEM         *
16 REM *                                                     *
40 REM *               BY PETER REECE                        *
42 REM *                                                     *
60 REM *******************************************************
70 DIM Q$(20),Q1(20),Q1$(20)
80 DIM Q2$(20),Q2(20)
90 DIM Q3(20)
110 GOTO 5380
120 REM
130 REM **** 'W2' IS THE NUMBER OF COLUMNS ALLOWED
       FOR A SEARCH ITEM ****
140 W2=20
150 PRINT" "
152 REM REWIND DATA FILES
160 RESTORE :1
170 RESTORE :2
180 PRINT "READY: ";
182 REM READ IN A COMMAND
190 INPUT C$
200 IF C$ = "END" THEN 3470
210 N = INSTR("HELCODDISNODENDEXIRAWRECNORFIN
       COUNOCTOTNOTSPE", LEFT$(C$,3))
220 IF N = 0 THEN 250
```

```
230 N = INT(N/3) +1
240 ON N THEN 2950,3480,4250,4280,3080,3080,320,5070,5100,
        350,4660,4690,5280,320,4710
250 N = INSTR("MATMAKSUGIS CANCREADDCHAUPDSHOWORLENASS",
        LEFT$(C$,3))
260 IF N = 0 THEN 290
262 REM CHOOSE A SUBSECTION, THEN GO THERE
270 N = INT(N/3) + 1
280 ON N THEN 350,350,350,350,350,2730,3090,3670,3670,
        4300,5160,5120,9000
290 GOSUB 3440
300 PRINT "*** I DO NOT RECOGNISE YOUR COMMAND ***"
310 GOTO 150
320 GOSUB 3440
322 REM REACH HERE IF NEW COMMANDS ARE BEING DEBUGGED
330 PRINT "COMMAND NOT YET IMPLIMENTED."
340 GOTO 150
350 REM
360 REM ************** FIND COMMAND **************
370 REM FIRST FORMALISE VERBS AND ELIMINATE NOISE WORDS
380 DATA "IS IT POSSIBLE TO HAVE","9"
390 DATA "IS THERE","9","CAN THERE","9"
400 DATA "FIND","9","MAKE","9","MATCH","9","IS IT POSSIBLE","
410 DATA "CAN","9","?","9"
420 DATA "AT",".AND."
430 DATA "WHO'S",".IS.","WHOSE","9"
440 DATA "AND","@","WAS NOT",".NOT."
450 DATA "SHOULD HAVE HAD",".IS.","SHOULD HAVE",".IS."
460 DATA "OF",".IS.","HAD",".IS.","HAS",".IS.","HAVE",".IS."
470 DATA "FOR",".IS.","WAS",".IS.","WILL BE",".IS.",
        "SHOULD BE",".IS."
480 DATA "IF","9","WHICH","9","THAT","9","WHO","9","THE","9"
490 DATA "IS GREATER THAN",".GT.","IS LESS THAN",".IT."
500 DATA "A","9","AN","9","IN","9","ON","9"
510 DATA "ARE",".IS.","IS NOT",".NOT.","ARE NOT",".NOT."
520 DATA "WITH","AND","<",".IT.",">",".GT.","ISN'T",".NOT."
530 DATA "WASN'T",".NOT.","HAVN'T",".NOT.","SHOULDN'T",".NOT."
540 DATA "IS",".IS.","&","AND","+","AND"
550 DATA "WITH",".AND.","TO","9"
560 DATA "SHOULD","9","BE",".IS.","YET","9",
        "IS MORE THAN",".GT."
570 DATA "999","9"
580 REM ********* PARSER BEGINS HERE **********
590 C7 = 0
600 RESTORE
610 N = INSTR(C$,"?")
620 IF N = 0 THEN 640
630 C7 = 444
640 READ A$,B$
650 IF A$ = "999" THEN 810
660 N = INSTR(C$,A$)
670 IF N = 0 THEN 640
680 L = LEN(A$)
690 N1 = N+1
700 IF MID$(C$,N1,1) = " " THEN 730
710 N = INSTR(N1,C$,A$)
720 GOTO 670
730 IF N - 1 > 1 THEN 760
740 C$=MID$(C$,1+2)
750 GOTO 660
760 IF B$ = "9" THEN 790
770 C$ =MID$(C$,1,N-1) + B$ +MID$(C$,N+1)
780 GOTO 660
790 C$ = MID$(C$,1,N-1) + MID$(C$,N+1)
800 GOTO 660
810 REM
820 REM NO ISOLATE THE INDIVIDUAL SENTENCES INTO Q$(*)
830 I = 0
840 I = I +1
850 K$ = "@"
860 L = LEN(K$)
870 N = INSTR(C$,K$)
880 IF N = 0 THEN 920
890 Q$(I) = MID$(C$,1,N-1)
900 C$ = MID$(C$,N+1)
```

```
910 GOTO 840
920 Q$(I) = C$
930 REM Q$(*) NOW CONTAINS THE SENTANCES TO BE 'ANDED'
940 REM NOW ELIMINATE ALL BLANKS FROM THE SENTENCE STRINGS.
950 FOR K = 1 TO I
960 C$ = Q$(K)
970 GOSUB 1290
980 Q$(K) = C$
990 NEXT K
1000 REM NOW ISOLATE THE VERBS
1010 MAT Q1 = ZER
1020 FOR K = 1 TO I
1030 N = INSTR(Q$(K),".IS.")
1040 IF N = 0 THEN 1080
1050 Q1(K) = 1
1060 Q1$(K) = MID$(Q$(K),N+4)
1070 GOTO 1240
1080 N = INSTR(Q$(K),".NOT.")
1090 IF N = 0 THEN 1130
1100 Q1(K) = 2
1110 Q1$(K) = MID$(Q$(K),N+5)
1120 GOTO 1240
1130 N = INSTR(Q$(K),".GT.")
1140 IF N = 0 THEN 1170
1150 Q1(K) = 3
1160 GOTO 1060
1170 N = INSTR(Q$(K),".IT.")
1180 IF N = 0 THEN 1210
1190 Q1(K) = 4
1200 GOTO 1060
1210 GOSUB 3440
1220 PRINT"** I DO NOT RECOGNIZE THE VERB IN '";
       Q$(K);"' **"
1230 GOTO 120
1240 NEXT K
1250 REM
1260 REM Q$(*) = SENTENCES, Q1(*) = VERB CODES,
       Q1$(*) = ITEMS SOUGHT
1270 REM I = # OF SENTENCES
1280 GOTO 1410
1290 REM ROUTINE TO ELIMINATE BLANKS FROM THE
       ENTER STRING 'C$'
1300 L1 = 0
1310 L = LEN(C$)
1320 L1 = L1+1
1330 IF L1 > 1 THEN 1400
1340 IF MID$(C$,L1,1) <>" " THEN 1310
1350 IF L1 <> 1 THEN 1380
1360 C$ = MID$(C$,2)
1370 GOTO 1310
1380 C$ = MID$(C$,1,L1-1) + MID$(C$,L1+1)
1390 GOTO 1310
1400 RETURN
1410 REM LOAD Q2$(*) WITH SENTENCE SUBJECTS FOR
       FURTHER MESSAGES.
1420 REM NOTE THAT IF A SENTENCE LACKS A SUBJECT
       THEN IT IS ASSUMED
1430 REM THAT THE VERB REFERS TO THE SUBJECT OF THE PRE-
       VIOUS SENTENCE.
1440 FOR K = 1 TO I
1450 N = INSTR(Q$(K),".")
1460 IF N = 0 THEN 1210
1470 IF N > 1 THEN 1510
1480 GOSUB 3440
1490 PRINT"** THERE IS NO SUBJECT IN THE SENTENCE '";
       Q$(K);"' **"
1500 GOTO 150
1510 Q2$(K) = MID$(Q$(K),1,N-1)
1520 IF Q2$(K) <>" " THEN 1540
1530 Q2$(K) = Q"$(K-1)
1540 NEXT K
1550 REM SEARCH SUBJECTS FOR 'TOTAL' COMMANDS
1560 Q3 = -1
1570 C8 = 0
1580 FOR K = 1 TO I
```

```
1590 N = INSTR(Q$(K),"*T")
1600 FI N = 0 THEN 1640
1610 Q2$(K) = MID$(Q2$(K),1,N-1)
1620 Q3(K) = 0
1630 C8 = 444
1640 NEXT K
1650 REM .................. PARSER ENDS HERE ...............
1660 REM NOW SEARCH A CODE TABLE FOR MATCHES TO THE
       SUBJECTS IN Q2$(*)
1670 REM AND STORE THE RECORD ADDRESSES.
1680 SET 2:1
1690 INPUT:2,C$
1700 W1 = VAL(C$)
1710 K = 0
1720 K = K+1
1730 IF K > I THEN 1890
1740 N = 0
1750 N = N+1
1760 IF N > W1 THEN 1840
1770 IF END:1 THEN 1840
1780 SET 1:N
1790 INPUT:1,C$
1800 IF INSTR(Q2$(K),C$) = 0 THEN 1750
1810 Q2(K) = N
1820 T1 = 0
1830 GOTO 1720
1840 GOSUB 3440
1850 PRINT " "
1860 PRINT"** '";Q2$(K);"' IS AN ILLEGAL SUBJECT
       FOR '";Q$(K);"' **"
1870 N1 = 999
1880 GOTO 1720
1890 IF N1 = 999 THEN 150
1900 REM ................. BEGIN SEARCH ................
1910 REM RECORD #1 OF THE DATA FILE CONTAINS 'W1'.  THIS IS
1920 REM A POINTER TO A KEY SEARCH ITEM.  THUS, IF THE PRESENT
1930 REM KEY ITEM IS A RECOR 'R', THE Nth KEY ITEM
       WILL BE AT RECORD 'R + W1*N'.
1940 T = 0
1950 R = 0
1960 SET 1:1
1970 INPUT:1,Q$
1980 SET 2:1
1990 IF END:2 THEN 2350
2000 INPUT:2,C$
2010 W1 = VAL(C$)
2020 R1 = 0
2030 IF END:2 THEN 2350
2040 FOR K = 1 TO I
2050 R1 = R + Q2(K)
2060 R1 = R1 + 1
2070 SET 2:R1
2080 IF END:2 THEN 2350
2090 INPUT:2,C$
2100 GOSUB 2520
2110 NEXT K
2120 IF T <> I THEN 2320
2130 T1 = T1 + 1
2140 REM 'RECORD' IN EFFECT ?
2150 IF C5 <> 444 THEN 2180
2160 PRINT"REC#";R;
2170 REM COUNT IN EFFECT ?
2180 IF C2 = 444 THEN 2230
2190 IF C1 <> 444 THEN 2320
2200 SET 2:R+2
2210 INPUT:2,C$
2220 PRINT G$; "=";C$;"     ";
2230 REM DETAIL IN EFFECT ?
2240 IF C1 <> 444 THEN 2320
2250 J = R
2260 FOR K = 1 TO I
2270 J1 = J + Q2(K) + 1
2280 SET 2:J1
2290 INPUT:2,C$
2300 PRINT Q2$(K); "=" ;C$;"    ";
```

```
2310 NEXT K
2320 T = 0
2330 R = R + W1
2340 GOTO 2030
2350 PRINT " "
2360 GOSUB 3440
2370 IF T1 > 1 THEN 2410
2380 IF T1 = 0 THEN 2410
2390 PRINT" I HAVE FOUND ONE FORM"
2400 GOTO 2460
2410 IF T1 <> 0 THEN 2450
2420 IF C' <> 444 THEN 2450
2430 PRINT" I HAVE FOUND NO FORMS."
2440 GOTO 2460
2450 PRINT"I HAVE FOUND ";T1;"FORMS."
2460 REM 'TOTAL' IN EFFECT ?
2470 IF C8 <> 444 THEN 2510
2480 FOR K = 1 TO I
2490 PRINT "TOTAL ";Q2$(K);" = ";Q3(K)
2500 NEXT K
2510 GOTO 150
2520 REM HERE THE ACTUAL COMPARISONS ARE PERFORMED
2530 REM THE OBJECTS OF THE INPUT SENTENCES (Q1$(*)) ARE
     COMPARED
2540 REM TO DATA ACCORDING TO THE VERBS IN Q1(*).
2550 P = T.
2560 N = INSTR(C$,Q1$(K))
2570 ON Q1(K) GOTO 2580,2610,2640,2670
2580 IF N <> 0 THEN 2690
2590 T = T + 1
2600 GOTO 2690
2610 IF N <> 0 THEN 2690
2620 T = T + 1
2630 GOTO 2690
2640 IF C$ < Q1$(K) THEN 2690
2650 T = T + 1
2660 GOTO 2690
2670 IF C$ > Q1$(K) THEN 2690
2680 T = T + 1
2690 IF P = T THEN 2720
2700 IF Q3(K) = -1 THEN 2720
2710 Q3(K) = VAL(C$) + Q3(K)
2720 RETURN
2730 REM ***************** CREATE COMMAND ****************
2740 N = 0
2750 P = 444
2760 SCRATCH:2
2770 SCRATCH:1
2780 K = 1
2790 GOSUB 3440
2800 PRINT"** WHEN YOU ARE FINISHED, TYPE 'DONE'. **"
2810 PRINT" "
2820 N = N+1
2830 PRINT"ITEM #";N;
2840 IF N <> 1 THEN 2860
2850 PRINT"(KEY) ";
2860 INPUT C$
2870 IF C$ = "DONE" THEN 2910
2880 SET 1:N
2890 PRINT: 1,C$
2900 GOTO 2820
2910 C$ = STR$(N-1)
2920 SET 2:1
2930 PRINT:1,C$
2940 GOTO 150
2950 REM ****************** HELP COMMAND ****************
2960 PRINT " "
2970 GOSUB 3440
2980 PRINT"** COMMANDS ARE AS FOLLOWS: **"
2990 PRINT"FIXED COMMANDS:"
3000 PRINT"HELP CODE DISPLAY NO DISPLAY END EXIT RAW "
3010 PRINT"RECORD NORECORD COUNT NOCOUNT TOTAL NOTOTAL "
3020 PRINT"ADD CREATE CHANGE SHOW WORD LENGTH ASSIGN"
3030 PRINT"VARIABLE COMMANDS:"
3040 PRINT"SPECIAL FIND MAKE IS IT POSSIBLE SUGGEST "
```

```
3050 PRINT"IS IT POSSIBLE TO HAVE IS IS POSSIBLE THAT"
3060 PRINT"MATCH IS THERE CAN THERE BE"
3070 GOTO 150
3080 STOP
3090 REM ***************  ADD COMMAND ******************
3100 C$=MID$(C$,4)
3110 IF C$<>""THEN 3150
3120 GOSUB 3440
3130 PRINT"ADD WHAT ? (EXAMPLE: ADD M.BROWN)"
3140 GOTO 150
3150 SET2:1
3160 INPUT:2,C1$
3170 W1=VAL(C1$)
3180 R = 2
3190 GOTO 3210
3200 R = R + W1
3210 SET 2:R
3220 IF END: 2 THEN 3280
3230 INPUT:2, C1$
3240 IF INSTR(C1$,C$) = 0 THEN 3200
3250 GOSUB 3440
3260 PRINT"** '";C1$;"' ALREADY EXISTS. **"
3270 GOTO 150
3280 R1 = 0
3290 IF P <> 444 THEN 3310
3300 R = 2
3310 R1 = R1 + 1
3320 SET 1:R1
3330 IF END:1 THEN 150
3340 FOR T = 2 TO W1 + 1
3350 INPUT:1,C$
3360 PRINT C$;
3370 INPUT K$
3380 IF K$ = "DONE" THEN 150
3390 SET 2:R+T-2
3400 PRINT :2,K$
3410 NEXT T
3420 P = 0
3430 GOTO 150
3440 REM MONKEY MESSAGE
3450 PRINT"COMPUTER MESSAGE: ";
3460 RETURN
3470 STOP
3480 REM ***************** CODE COMMAND ******************
3490 SET 2:1
3500 GOSUB 3440
3510 PRINT" AT WHAT RECORD NUMBER SHALL I BEGIN ";
3520 INPUT C$
3530 IF C$ = "DONE" THEN 150
3540 A = VAL(C$)
3550 GOSUB 3440
3560 PRINT"AT WHAT RECORD # SHALL I END ";
3570 INPUT C$
3580 IF C$ = "DONE" THEN 150
3590 B = VAL(C$)
3600 FOR I = A TO B
3610 IF END:2 THEN 150
3620 SET 2:1
3630 INPUT:2,C$
3640 PRINTI;". ";C$
3650 NEXT I
3660 GOTO 150
3670 REM ********* CHANGE COMMAND *********
3680 GOSUB 1290
3690 C$ = MID$(C$,8)
3700 IF C$ <>""THEN 3750
3710 GOSUB 3440
3720 PRINT"CHANGE WHAT ?";
3730 PRINT"(EXAMPLE: CHAGE DIET IN RECIPE = 1 TO NUTRICIOUS)."
3740 GOTO 150
3750 N = INSTR(C$,"IN")
3760 IF N<> 0 THEN 3800
3770 GOSUB 3440
3780 PRINT"ILLEGAL 'CHANGE' STRUCTURE: ";
3790 GOTO 3730
```

```
3800 Q2$(1) = MID$(C$,1,N-1)
3810 C$ = MID$(C$,N+2)
3820 N = INSTR(C$,"=")
3830 IF N = 0 THEN 3780
3840 Q2$(2) = MID$(C$,1,N-1)
3850 C$ = MID$(C$,N+1)
3860 N = INSTR(C$,"TO")
3870 IF N = 0 THEN 3780
3880 Q2$(3) = MID$(C$,1,N-1)
3890 Q2$(4) = MID$(C$,N+2)
3900 Q2$(2) = Q2$(3)
3910 Q2$(3) = Q2$(4)
3920 20R = 0
3930 R = R+1
3940 SET1:R
3950 IF END:1 THEN 4000
3960 INPUT:1,C$
3970 IF INSTR(C$,Q2$(1)) = 0 THEN 3930
3980 N1 = R
3990 GOTO 4030
4000 GOSUB 3440
4010 PRINT"'";Q2$(1);"' IS AN ILLEGAL ITEM FOR THIS
       DATABASE."
4020 GOTO 150
4030 REM HAVE SET N = POINTER WITHIN A DATA FIELD,
       FIND THAT DATA FIELD
4040 SET 2:1
4050 INPUT:2,C$
4060 W1 = VAL(C$)
4070 R = 2
4080 SET2:R
4090 IF END:2 THEN 4140
4100 INPUT :2,C$
4110 IF INSTR(C$,Q2$(2)) <> 0 THEN 4170
4120 R = R + W1
4130 GOTO 4080
4140 GOSUB 3440
4150 PRINT"I CANNOT FIND THE ENTRY'";Q2$(2);
       "' IN THE DATA."
4160 GOTO 150
4170 R = R + N1-1
4180 SET 2:R
4190 INPUT :2,C$
4200 SET 2:R
4210 PRINT :2,Q2$(3)
4220 GOSUB 3440
4230 PRINT"'";C$;"' CHANGED TO '";
       Q2$(3);"' IN ";Q2$(2)
4240 GOTO 150
4250 REM **** DETAIL COMMAND ****
4260 C1 = 444
4270 GOTO 150
4280 C1 = 0
4290 GOTO 150
4300 REM ***** SHOW COMMAND *****
4310 C4 = 0
4320 C$ = MID$(C$,5)
4330 IF C$ <> "" THEN 4370
4340 GOSUB 3440
4350 PRINT"SHOW WHAT ? (EG.: SHOW NAME = J.SMITH)."
4360 GOTO 150
4370 N = INSTR(C$,"=")
4380 IF N = 0 THEN 4350
4390 Q2$(1) = MID$(C$,1,N-1)
4400 Q2$(2) = MID$(C$,N+1)
4410 SET1:1
4420 INPUT:1,C$
4430 IF INSTR(Q2$(1);C$) <> 0 THEN 4470
4440 GOSUB 3440
4450 PRINT"'";Q2$(1);"' IS NOT A KEY ITEM."
4460 GOTO 150
4470 SET2:1
4480 INPUT:2,C$
4490 W1 = VAL(C$)
4500 R = 2-W1
```

```
4510 R = R + W1
4520 SET 2:R
4530 IF END: 2 THEN 4150
4540 INPUT:2, C$
4550 IF INSTR(C$,Q2$(2)) = 0 THEN 4510
4560 IF C4 = 444 THEN 4940
4570 FOR I = 1 TO W1
4580 SET 1:1
4590 INPUT:1,C$
4600 PRINT I;". ";C$;":  ";
4610 SET2:R + I-1
4620 INPUT:2,C$
4630 PRINT C$
4640 NEXT I
4650 GOTO 150
4660 REM ****** COUNT COMMAND *****
4670 C2 = 444
4680 GOTO 150
4690 C2 = 0
4700 GOTO 150
4710 REM ****** SPECIAL COMMAND ********
4720 C$ = MID$(C$,8)
4730 IF C$ = "" THEN 4760
4740 CHAIN C$
4750 GOTO 150
4760 GOSUB 3440
4770 PRINT " "
4780 PRINT TAB(5);"TO USE YOUR OWN COMMANDS, YOU
       MAY TYPE"
4790 PRINT"'SPECIAL PROGX, WHERE 'PROGX' IS THE NAME
       OF A PROGRAM"
4800 PRINT"WHICH YOU WISH TO CALL FROM WITHIN THIS
       SYSTEM."
4810 PRINT"THE SECOND LAST LINE OF 'PROGX' MUST BE"
4820 PRINT"'CHAIN PROGO'.  THE 'X' IN 'PROGX'"
4830 PRINT"MAY BE IN THE RANGE 1-99, i.e. YOU MAY HAVE UP"
4840 PRINT"TO ONE HUNDRED ROUTINES OF YOUR OWN."
4850 GOTO 150
4860 REM *********** DELETE COMMAND ***********
4870 C$ = MID$(C$,8)
4880 CR = 444
4890 IF C$ <> "" THEN 4370
4900 C4 = 0
4910 GOSUB 3440
4920 PRINT"DELETE WHAT ? (e.g. DELETE NAM = J.SMITH)"
4930 GOTO 150
4940 SET 2:R
4950 INPUT:2,C$
4960 GOSUB 3440
4970 PRINT"DELETE ";C$
4980 INPUT K$
4990 IF MID$(K$,1,1) = "N" THEN 150
5000 FOR I = 1 TO W1
5010 SET2: R + I-1
5020 PRINT: 2," "
5030 NEXT I
5040 GOSUB 3440
5050 PRINT"FORM DELETED FOR KEY ITEM; ";C$
5060 GOTO 150
5070 REM **************** RECORD COMMAND ***************
5080 C5 = 444
5090 GOTO 150
5100 C5 = 0
5110 GOTO 150
5120 REM *************** LENGTH COMMAND ***************
5130 GOSUB 3440
5140 PRINT"THE FILE CONTAINS ABOUT ";LOF(:2);" ITEMS."
5150 GOTO 150
5160 REM ****** WORD COMMAND ******
5170 RESTORE
5180 PRINT
5190 I = 0
5200 READ A$,B$
5210 IF A$ = "999" THEN 5260
5220 I = I + 1
```

```
5230 PRINT A$;" ";TAB(I*18);
5240 IF I > 3 THEN 5180
5250 GOTO 5200
5260 PRINT
5270 GOTO 150
5280 REM *********** TOTAL COMMAND **********
5290 GOSUB 3440
5300 PRINT" YOU MAY REQUEST TOTALS FOR NUMERIC
       ITEMS ONLY. FOR"
5310 PRINT"EXAMPLE: FIND A COST*T WHICH IS LESS THAN 50"
5320 PRINT"AND A CALORIE-COUT*T OF MORE THAN TEN AND"
5330 PRINT"AN AGE OF 15."
5340 PRINT"THIS WOULD PRESENT SUMS OF THE VALUE OF"
5350 PRINT"'CALORIE-COUNT' AND 'COST', BUT NOT OF AGE."
5360 GOTO 150
5370 REM *** A USER MAY HAVE UP TO TEN DATA BASES ***
5380 PRINT"PLEASE TYPE THE NUMBER OF YOUR DATABASE";
5390 INPUT N
5400 IF N < 1 THEN 5380
5410 IF N > 10 THEN 5380.
5420 ON N THEN 5430, 5450, 5470, 5490, 5510, 5530,
       5550,5570,5590,5610
5430 FILE:1,"CODE1.BAS$20",:2,"DATA1.BAS$80"
5440 GOTO 5620
5450 FILE:1,"CODE2.BAS$20",:2,"DATA2.BAS$80"
5460 GOTO 5620
5470 FILE:1,"CODE3.BAS$20",:2,"DATA3.BAS$80"
5480 GOTO 5620
5490 FILE:1,"CODE4.BAS$20",:2,"DATA4.BAS$80"
5500 GOTO 5620
5510 FILE:1,"CODE5.BAS$20",:2,"DATA5.BAS$80"
5520 GOTO 5620
5530 FILE:1,"CODE6.BAS$20",:2,"DATA6.BAS$80"
5540 GOTO 5620
5550 FILE:1,"CODE7.BAS$20",:2,"DATA7.BAS$80"
5560 GOTO 5620
5570 FILE:1,"CODE8.BAS$20",:2,"DATA8.BAS$80"
5580 GOTO 5620
5590 FILE:1,"CODE9.BAS$20",:2,"DATA9.BAS$80
5600 GOTO 5620
5610 FILE:1,"CODE10.BAS$20",:2,"DATA10.BAS$90"
5620 GOTO 130
9000 REM *********** REASSIGN COMMAND ***********
9002 GOTO 110
9999 END
```

APPENDIX: EXAMPLES OF IDMAS USE

```
READY? create
    *** WHEN FINISHED, TYPE 'DONE' ***
ITEM #1 (KEY)? product
ITEM #2? location
ITEM #3? cost
ITEM #4? maintenance
ITEM #5? type
ITEM #6? life-yrs
ITEM #7? asmb-time
ITEM #8? distribution?
ITEM #9? conversion
ITEM #10? done

READY? add drive-shaft
PRODUCT? drive-shaft
LOCATION? c3-1a
COST? 120
MAINTENANCE? 1
TYPE? carriage
LIFE—YRS? 20
```

```
 ASMB-TIME? 30
 DISTRIBUTION? general
 CONVERSION? none

 READY? add drive-shaft
    ** 'DRIVE-SHAFT' ALREADY EXISTS **


READY? add unit1
PRODUCT? unit1
LOCATION? 1a
COST? 120
MAINTENANCE? 1
TYPE? carriage
LIFE-YRS? 10
ASMB-TIME? 15
DISTRIBUTION? general
CONVERSION? custom

READY? is there an asmb-time which is less than 30 and
       a location that is in 1a?
** I HAVE FOUND ONE FORM **

READY? display

READY? find a cost which is less than 140*t and an asmb-time
       which is greater than 10*t and a location which isn't 1b.
PRODUCT=DRIVE-SHAFT   COST=120   ASMB-TIME=30   LOCATION=C3-1A
PRODUCT=UNIT1         COST=120   ASMB-TIME=15   LOCATION=1A
** I HAVE FOUND 2 FORMS **
TOTAL COST=240       TOTAL ASMB-TIME=45

READY? end
```

Chapter 6

The Computation of Direction

By Gene Szymanski

In the daily pursuit of our affairs, we do not find it necessary to have a knowledge of absolute direction, for we are able to find our way bout through a recognition of familiar sights and sounds. Even when it is necessary to travel beyond the conventional routes, there are available a multitude of guides to help us reach our destination.

On those rare occasions when we find that we are "lost", the feeling of disorientation quickly subsides once a familiar landmark comes into view, for then we quickly recover our sense of direction.

For our purposes, then, direction is thought of as it relates to some recognized object or prominent feature, such as a structure, highway intersection, or the skyline of a city. Sometimes we find it convenient to extend the scope of our reference by descriptives, such as "to the north, or east," and so on.

The surveyor, navigator, and astronomer require a more precise definition of direction in their work. They are concerned with the measurement of exact positions, often separated by great distances. For their purposes, the concept of direction is of fundamental importance.

THE MEASUREMENT OF DIRECTION

Direction is the angular difference measured in degrees from a reference. For most purposes, we are interested in "true direction" whose reference is the geograhic north pole. True direction is measured as an angle whose initial value is 0 degrees at north and which increases in a clockwise direction to 360 degrees. In order to measure true direction, then, it is first necessary to accurately determine the direction of the earth's geographic poles.

For centuries, the magnetic compass has served as the principal instrument for providing a knowledge of direction. Unfortunately, the compass indication of true north is subject to considerable error. The directive force on the compass needle is the result of two forces, one exerted by the earth's magnetic field and the second exerted by iron or steel which may be found in the vicinity of the compass.

The earth's magnetic field is irregular; furthermore, the position of the magnetic and geographic poles do not coincide. This gives rise to error called variation in the direction of north indicated by the compass needle. The amount of variation depends upon location and can be found by consulting a map or chart of the locality. A chart of Long Island Sound, for example, would show that the variation is 13 degrees west. This means that the compass needle is deflected 13 degrees to the west of true north so that a value of 13 degrees must be subtracted from the compass reading to obtain the true direction.

Obviously, it is a simple matter to cope with variation. All that is necessary is to determine its value from the chart, then apply it to the compass reading by addition or subtraction.

The second source of compass error, caused by the presence of iron and steel near the compass, is far more troublesome. The result of this type of error, called "deviation," must be carefully measured for each compass installation before the instrument can be used with confidence. Even then, after deviation errors have been measured and recorded for reference, they are subject to gradual change as the vehicle or ship in which the compass is installed is moved to other locations. Obviously, if the compass is to serve as a reliable instrument, its errors must be known under all conditions of use.

AZIMUTH OBSERVATIONS

Because of the regularity in which celestial bodies appear to move overhead, we are able to observe their positions in the sky and, from this, determine direction. In practicing this technique, we are said to be performing an "azimuth observation," following a procedure which is used throughout the world to establish direction.

The azimuth of a celestial body is simply its direction from the observer and is measured as a horizontal angle from the north clockwise to 360 degrees. In facing the body, we are also facing the point on the earth's surface directly beneath the body. This point is called the geographic position or "GP" of the celestial object, and, in a strict sense, the term "azimuth" refers to the direction of the GP.

If both the position of the observer and the GP of the celestial body are known, the azimuth of the body can be computed. An accurate direction is thus established which serves as an absolute reference for determining any other direction quickly and simply. Azimuth observations enable us to survey the wilderness, allign launching pads in the desert, and determine the error of the ship's compass at sea.

In practice, one sights a celestial body, preferably when it is low in the sky, using a suitable pointer. The pointer is then locked into position, and the exact time is recorded. A celestial "timetable" or almanac is then entered with the time and date of the observation to extract the geographic position of the observed object. Combining this with the position of the observation, the azimuth is computed. This azimuth is the exact direction in which the locked pointer is oriented. The direction of any other object is then found by measuring its angular displacement horizontally from the reference pointer.

METHODS FOR COMPUTING AZIMUTH

The computation of azimuth requires the application of spherical trigonometry. This is because we are dealing with a geometric figure lying on the earth's curved surface (see Figure 1). This figure is a triangle formed by connecting the geographic positions of the celestial body, the observer, and the north (or south) pole.

Although the equations for the computation of azimuth are well established, the solution is tedious and prone to human error. For this reason, many methods have been devised which attempt to ease the burden of computation. These range from tables of logarithms to volumes of "pre-computed" solutions, to which the user must apply a liberal amount of interpolation before arriving at the final result.

In dealing with logarithmic solutions, for example, it is necessary to perform the addition and subtraction of at least a dozen 6-digit numbers after they have been extracted from logarithmic tables. In addition, the "labels" of various angles must be examined during intermediate steps in order to determine how the arithmetic is to proceed.

The solution for azimuth is an ideal application for the small computer, and in this role it replaces pages of mathematical tables. The program can be designed to perform a multitude of preliminary calculations which are necessary to establish the known parts of the spherical triangle. The solution of the trigonometric equations then proceeds rapidly.

The azimuth program shown here is written in MITS 8K BASIC. The entering arguments consist of the observer's latitude and longitude, the Greenwich hour angle and declination of the observed celestial body. Greenwich hour angle (GHA) and declination (DEC) are the astronomical counterparts to longitude and latitude, respectively, and define the GP of the celestial body.

Both GHA and DEC for all the prominent celestial bodies are obtained from the "Nautical Alamanac," a publication prepared by the U.S. Naval Observatory and issued by the Government Printing Office and its agents. Because of the earth's motion, the GP of every celestial body is constantly changing so the Nautical Almanac must be entered with the exact date and time (to the nearest minute) of the observation.

The azimuth program solves the spherical triangle (Figure 1) for the angle AZ which is formed by the great circles connecting the observer, the GP, and the nearest geographic pole. Because of the apparent motion of point GP, this triangle expands, then contracts as the celestial body sets in the west, and eventually rises in the east.

The program first processes longitude and GHA to determine angle T. The latitude and declination are then examined to establish two

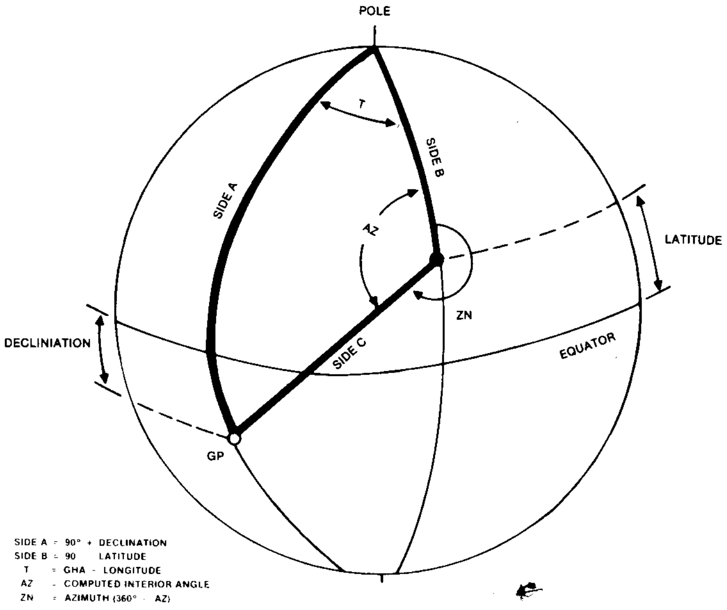


Figure 1. Geometry used by the Azimuth Program.

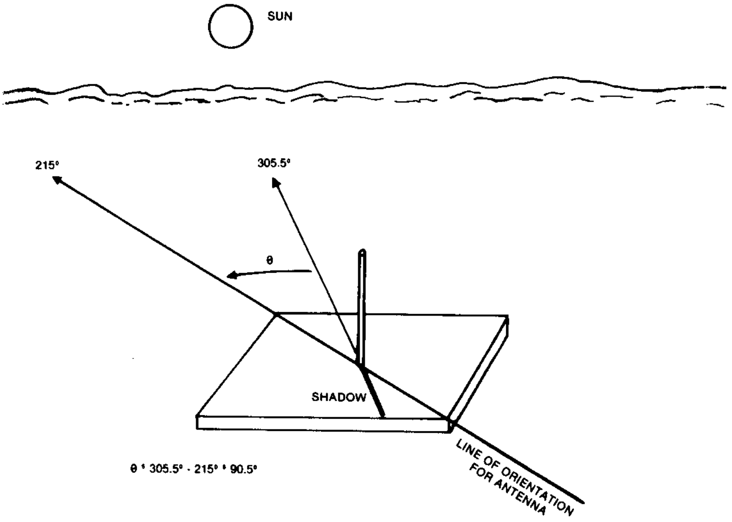


Figure 2. Azimuth indicator used in the example.

sides of the triangle; the interior angle AZ can then be computed.

Finally, the program converts AZ to angle ZN, the azimuth. This is of great advantage when done by the program since one of four different conversion rules must be selected. This is because the triangle may be referenced to either the north or south pole, depending upon which is nearest to the observer.

When started, the program prompts the user to enter input data in the following format: first degrees, next minutes, then the "label." Degrees are always entered as an integer, while minutes are to be entered to the nearest tenth. "Label" refers to the characteristic suffix east or west (for longitude) and north or south (for latitude and declination); there is no label for Greenwich hour angle, GHA.

A longitude, for example, whose value is 134 degrees, 15.7 minutes west would be input to the program in the following format:

(degrees)	134
(minutes)	15.7
(label)	W

At its conclusion, the program prints out the value of ZN. This is the azimuth, or true direction of the celestial body from the observer, valid for the instant of the observation.

MATHEMATICS USED FOR SOLUTION

First the input values, specified in degrees and fractional minutes of ARC, are converted into a decimal degrees format (fractional degrees).

"Local hour angle" (LHA) is now introduced as a parameter to expedite the calculation for meridian angle, T. LHA is found by adding east logitudes to and subtracting west longitudes from Greenwich hour angle, GHA.

Coefficient M has an absolute value of unity. Its sign is now made positive if latitude and declination are both north or both south; otherwise, the sign of M is to be negative.

Side C of the spherical triangle (see Figure 1) is computed by the following formula, derived from the Law of Cosines:

SIN C = SIN L*SIN D + M(COS L*COS D*COS T)

The intermediate value S is not computed as:

S = (C + L + 90 – M*D)/2

The "haversine" function of angle AZ is computed as follows:

HAV AZ = SIN(S-L)*SIN(S-C)/(COS C*COS L)

Interior angle AZ can now be found according to the following identity:

COS AZ = 1-2*HAV AZ

Finally, the value of the azimuth ZN is determined by noting the label specified for latitude and the label of angle T. Four different label combinations are possible and angle ZN is derived from angle AZ according to the following rule:

Label of L	Label of T	ZN
N	E	ZN = AZ
N	W	ZN = 360 – AZ
S	E	ZN = 180 – AZ
S	W	ZN = 180 + AZ

Note: Angle T is assigned the label "W" if LHA is less than 180 degrees; otherwise, the label "E" is assigned to angle T.

APPLICATIONS

The technique of celestial observation for azimuth can be employed wherever there is a need to establish direction. Some typical examples are as follows:

1) Orientation of structures, highways, property lines.
2) Surveys and construction of maps.
3) Calibration of the magnetic compass or gyrocompass.
4) Orientation of solar energy receptors, directional antennas, tracking devices, telescopes.
5) Predicting the position on the horizon at which the sun or any other celestial body will appear to rise or set.

At sea, azimuth observations are performed daily in order to detect any changes in magnetic compass deviations and to verify the accuracy of the ship's gyrocompass.

Professional compass adjusters, when calibrating a ship's compass for the first time, also rely on the azimuth technique for a directional reference. Their approach is to first compute the sun's azimuth for periodic time intervals, and from this draw a curve of azimuth values as a function of time. The ship is then placed on various headings and the sun's azimuth, as measured by the compass, is noted. By comparing the observed value of the azimuth with the precomputed value, the residual deviation error in the compass can be quickly determined.

Surveyors and mapmakers rely on azimuth observations to provide them with the geographic orientation vital in their work. Once they have obtained directional orientation, a baseline can be established whose length and direction are well defined. The baseline may then be used as a datum from which all other points of interest can be established by triangulation.

EXAMPLE

The following example indicates how the azimuth program would be applied to a practical situation:

A directional transmitting antenna is to be oriented such that it points exactly towards a receiving antenna located several hundred miles distant. Magnetic compass readings cannot be relied upon because of the presence of electrical machinery and a large steel cyclone fence surrounding the transmitter site.

As a first step, the required direction is determined. The transmitter and receiver positions are marked on a great circle chart and the line connecting them is found to have a direction of 215 degrees. A suitable azimuth indicator is now set up next to the transmitting antenna. A simple but effective indicator can consist of a flat sheet of cardboard placed on a level surface, pierced by a rigid, vertical pin.

At a convenient hour, when the sun is low in the sky, a mark is made on the cardboard to indicate the position of the shadow cast by the pin, and the exact time is recorded.

The Nautical Almanac is now entered with the date and recorded time of the observation, and the coordinates of the sun are found to be:

GHA = 81 degrees, 40.2 minutes.
DEC = 22 degrees, 03.2 minutes, north.

The chart indicates that the position of the transmitter is:

Longitude = 20 degrees, 40.2 minutes, west.
Latitude = 41 degrees, 00.0 minutes, south.

These values are entered into the computer azimuth program, and the resulting print-out indicates the azimuth to be exactly 305.5 degrees. In other words, this is the direction which the shadow described at the time of the observation.

A line drawn on the cardboard surface from the mark towards the position of the vertical pin, therefore, points in the exact direction of 305.5 degrees. A second line can now be drawn, offset from this reference "pointer" by an angle of 90.5 degrees to the left (305.5 – 215 = 90.5), to indicate the direction for the antenna.

PROGRAM LISTING

```
5 REM:PROGRAM"AZIMUTH BY CELESTIAL OBSERVATIONS",
6 REM:BY GENE SZYMANSKI, JAN 3,1978
9 CLEAR 100
10 REM:DATA INPUT MODULE
20 PRINT"ENTER LONGITUDE:"
22 INPUT"DEGREES";A(1):INPUT"MINUTES";B(1)
24 INPUT"LABEL(E OR W)";A$
25 PRINT:PRINT
30 PRINT"ENTER LATITUDE:"
32 INPUT"DEGREES";A(2):INPUT"MINUTES";B(2)
34 INPUT"LABEL(N OR S)";B$
35 PRINT:PRINT
40 PRINT"ENTER DECLINATION:"
42 INPUT"DEGREES";A(3):INPUT"MINUTES";B(3)
44 INPUT"LABEL(N OR S)";C$
45 PRINT:PRINT
50 PRINT"ENTER GHA:"
52 INPUT"DEGREES";A(4):INPUT"MINUTES";B(4)
100 REM:CONVERT INPUTS TO DECIMAL DEGREES
110 FORI=1TO4
120 B(I)=A(I)+B(I)/60
130 NEXT I
200 REM:COMPUTE LOCAL HOUR AND MERIDIAN ANGLES & M.
210 IF A$="W"THEN B(1)=-1*B(1)
```

```
220 LH=B(4)+B(1)
230 IF LH<180 GOTO 250
240 T=360-LH: TS=1: GOTO 260
250 T=LH: TS=-1
260 IF T>0 GOTO 280
270 T=-1*T:TS=-1*TS
280 T$="E"
290 IF TS<0 THEN T$="W"
291 LET M=-1
292 IF B$=C$ THEN M=1
300 REM:SOLVE FOR COMPUTED ALTITUDE
310 K=57.2958
320 A=SIN(B(2)/K)*SIN(B(3)/K)
321 A1=M*COS(B(2)/K)*COS(B(3)/K)*COS(T/K):A=A+A1
330 HC=(ATN(A/SQR(1-A↑2)))*K
340 HC=ABS(HC)
500 REM:COMPUTE INTERIOR ANGLE AZ
510 S=0.5*(HC+B(2)+90-M*B(3))
520 H1=SIN((S-B(2))/K)*SIN((S-HC)/K)
530 H2=H1/(COS(HC/K)*COS(B(2)/K))
540 H3=1-2*H2
550 H4=ATN(SQR(1-H3↑2)/H3)
560 AZ=K*H4
561 IF AZ<0 THEN AZ=180+AZ
600 REM:COMPUTE ZN
610 LET X$=B$+T$
620 IF X$="NE"THEN ZN=AZ
630 IF X$="NW" THEN ZN=360-AZ
640 IFX$="SE" THEN ZN=180-AZ
650 IF X$="SW" THEN ZN=180+AZ
651 ZN=INT(ZN*10+0.5)/10
652 PRINT:PRINT
660 PRINT "ZN=";ZN;"DEGREES"
670 PRINT "DONE"
680 END
OK


RUN
ENTER LONGITUDE:
DEGREES? 20
MINUTES? 40.2
LABEL(E OR W)? W


ENTER LATITUDE:
DEGREES? 41
MINUTES? 0
LABEL(N OR S)? S


ENTER DECLINATION:
DEGREES? 22
MINUTES? 3.2
LABEL(N OR S)? N


ENTER GHA:
DEGREES? 81
MINUTES? 40.2


ZN= 305.5 DEGREES
DONE
```

Chapter 7

Random Files Illustrated

by Frederick E. La Plante, Jr.

INTRODUCTION

In the recent series on General Business Software by Shamburger, I seem to detect an apology for not having used a truly random file approach in this design. This set me to thinking and try as I might, I could not recall having seen a single software article in the "hobbyist" literature which used random files. Since I had just recently finished a small software package to maintain a program for a membership file using random files with BASIC, the thought occurred to me that others might be interested in a practical example of the utility of such file structures.

DEFINITION OF FILES

Before we go any futher, we had better define just what sequential and random files are and how they differ.

A *sequential file* is typified by a magnetic tape. Typically such a file consists of a number of records end-to-end along the tape, usually in the order in which they are most frequently needed. When access is required to a particular record, the usual procedure is to rewind the tape to assure that it is at the start of the file, then read each record in turn, performing any necessary processing, and then read the next. If the program should need to read only one record, say the recipe for rhubarb pie, we must read through all of them until the desired record is found. If we wish to insert a new record, say a newly hired employee into a file ordered by employee number, and the new record must be placed anywhere other than at the end (always, in accordance with Murphy) then unless the file is small enough to fit into memory (never, same reason) we must copy tape #1 to tape #2, from the start until we reach the insertion point, write the new record on tape #2, then continue with the copy. Now, suppose instead of one new employee, we have five scattered at random throughout the existing range of employee numbers. We could simply process them one at a time at random, each time rewinding the most recent copy, and then copying and inserting as above. This would be slow, and rough on the tape as well. So, to do the job right, we first sort the insertions in employee number order and then read tape #1 to the point of the first insertion, write it, copy until reaching the place for the second, etc., until the updates are made. We then end up with a back-up tape of "yesterday's" file, a new updated file, and if we save it somewhere, a sorted list of the updates. The value of this file set will become obvious the first time the boss wishes to inspect the file after you have added the new employees, and you find the copy didn't take.

A *random file* is typified by a library book shelf where you can retrieve any book without disturbing any others on the shelf. Again, individual records are stored end to end, but the order may not be at all obvious if you do not recognize the key (catalog code). This key is probably some alphanumeric character sequence which the librarian (programmer) found easy to generate for each record and is *absolutely* unique. It may make no sense at all to anyone else but that doesn't matter. To find any record in the file, you must go to the index and look for the record in a sequence of key names (author, title, etc.) and get the corresponding record number (catalog code). You then go directly to that record and retrieve it. Physical devices providing that feature work much like normal computer memory in that you specify an address and are presented with data. (In fact, except for speed of access, it is frequently possible to treat them that way.) In order to write a new record, simply add the new record to the end of the file, and place its record number and keyword in the index in its proper place. There is no need to copy the file at all, and if any sorting is to be done, it will be the index which is usually very small compared to the file. Should you wish to modify a record, you simply read the old version into memory, change it, and write the new version over the old with no copying required. Thus, there is never more than one copy of the file and it is always current.

The accompanying program MARSBASE implements a comparatively crude database with 128 bytes allocated to each. Either or both of these limits can be increased up to the limits of disc space that the user wishes to commit. Since this is a random file, we can do directly to any specific member's record (assuming we know which one it is) so response time is not significantly affected by the size of the database. A little thought should result in a fairly large number of applications for such an approach to data storage. No longer does one have to read through most of the entire file of recipes in order to look at the one for rhubarb pie (or how about the contest log checking for radio amateurs). In fact, random file design allows one to get significantly closer to real-time access to a specific item in a voluminous data file.

MARSBASE PROGRAM

The program described here is written in a rather unusual form of BASIC. It is called BASIC-E and was written by Gordon E. Eubanks, Jr., of the Naval Post Graduate School. This BASIC runs under the CPM operating system written by Digital Research and takes advantage of its rather complete file editing system and I/O package. The particular version I used is that distributed by Imsai™ with their floppy disk.

Those familiar with BASIC will notice several peculiarities about the program described here. Perhaps the most obvious are the lack of line numbers and the absence of 2-character variable names. Less obvious is the IF–THEN–ELSE statements and the line-continuation symbol \. In writing MARSBASE it was decided to make liberal use of these features to determine if any significant improvements could be made in the readability and understandability of this program as compared to the usual BASIC program. In writing the program, I tried to follow the structured programming precepts of no "GOTO" statements and single entry and exit points from a block of code. I was not altogether successful, but still I think the understandability has been considerably enhanced.

One other aspect of BASIC-E has also affected the program and that is the fact that BASIC-E is a compiler/interpreter similar in some respects to the concepts of TINY BASIC. That is, the code you see here is pre-processed by a compiler into an intermediate language with all symbols reduced and all remarks, etc. removed. This intermediate language is then interpreted by the run-time software. While one has lost the rapid interactive features most beginners seem to find appealing, one gains the ability to be somewhat verbose in the source code while still retaining most of the advantages of compact code for the interpreter.

PROGRAM FEATURES

The program breaks down functionally into six major segments: The main program, four processing modules, and a set of support subroutines.

The main program defines variables, establishes array space, creates the database index, and allows the user to select from the functions available. The important thing to notice here is the index, for this is the heart of random file access. Whenever some part of the program wishes to access a particular member's data record, a search is made of the index to determine the record number and the program then asks the operating system for that specific record. The method used for searching the index is a simple sequential one of comparing each entry in turn. If the database were much larger than it is, a faster method of locating the key would be appropriate, but was not used in this case since the response time of the disc system seems to mask any search time.

Note that the index is nothing more than an array containing the "key" words, in this case the member's amateur call sign. However, the array is organized in exactly the same order as the database so that if the desired key is found in the 25th place in the array, then the desired record is number 25 in the file. While we are still performing a sequential search, we are now doing it in core at the maximum rate of the interpreter and also we only search through the keys. While smaller files of 10-15 records would barely show the difference between normal sequential and random files, the advantages become very obvious as the number of records increases and individual record size increases.

The functional modules perform as follows:

ADD—Get information from the operator concerning a new member, format it and insert it into the database. One search is made to insure that a duplicate entry is not being made and another to find the first empty record. If no empty record is found, the new record is added to the end of the database and a new end-of-file flag is written. When the user indicates there are no more additions to be made, he is advised of the current size of the database and returned to the main program.

CHANGE—Get a new item entry from the operator and insert it into the proper place in the database for the specified member. After the member to be altered is stated, the record is read into the core. The operator is asked to specify the item to change and its new value. Items may be changed in any order. When the operator has no further changes for that member, he is given a copy of the member's record as it appears on the disc with all changes made. When no further members are to be changed, control is passed back to the main program.

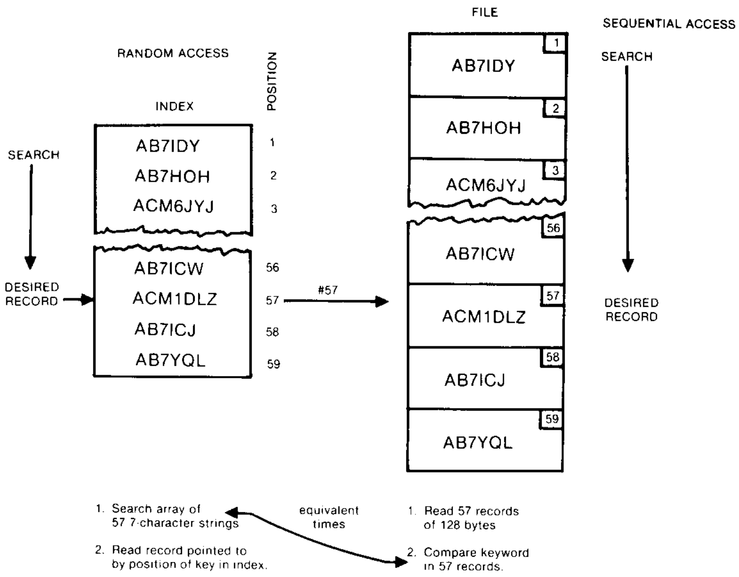


Figure 1. Comparison of random and sequential access methods.

DELETE—Remove a member from the database as specified by the operator. Locate member specified by the operator and replace key in both index and record with a zero, thus marking the space as available for new additions.

LIST—Print record for a specified member. If a call sign of "all" is specified, the entire database will be listed in the same order in which it exists on the disc.

Support subroutines are provided to perform most of the actual mechanics of database maintenance. Routines are provided to read and write a member record to a particular disc record; to fetch member data from the operator; to locate a member in the index and return the record number; and to print a member's entire record at the terminal.

Presumably, the functions provided by this database maintenance program could be extended indefinitely by adding sorting modules, mailing label printers, etc. However, it is usually more appropriate to keep the maintenance functions in a program separate from the data retrieval programs to minimize the danger of inadvertently changing the contents of the file. Consequently, such functions which only need to read the data will be kept separate and optimized for their particular functions.

It is seriously doubted that this program is of direct use to any reader in its present form, especially in light of the peculiarities of BASIC-E. Hopefully, though, it will serve as an illustration of the utility of random file access for record keeping. Such methods have uses in almost all fields of data processing, from engineering to bookkeeping systems to home recipe files to stamp collecting.

PROGRAM LISTING

```
REMARK U.S. ARMY-ALASKA MARS MEMBERSHIP FILE SYSTEM
REMARK
REMARK THIS IS A COMPLETE MEMBERSHIP FILE MAINTENANCE SYSTEM
REMARK ROUTINES ARE AVAILABLE TO ADD, CHANGE, DELETE OR
REMARK LIST ALL OR PARTS OF THE MEMBERSHIP FILE
REMARK
REMARK FILE IS OF RANDOM READ/WRITE DESIGN. EACH MEMBER IS
REMARK ASSIGNED A SEPERATE RECORD AS FOLLOWS:
REM
REM
REM      FIRST NAME        FIRST$
REM      MIDDLE INIT       MI $
REM      LAST NAME         LAST$
REM      CALL SIGN         SIGN$
REM      ADDRESS           ADR ,
REM      CITY              CITY $
REM      STATE             STATE$
REM      ZIP CODE          ZIP
REM      HOME PHONE        HFONE$
REM      WORK PHONE        WFONE$
REM      LICENSE CLASS     CLAS$
REM      EXPIRATION        EXPIRE
REM
REM
REMARK
REMARK
```

```
REM - - STATEMENT FUNCTIONS USED IN PRINT ROUTINES

        REM  CONVERT ZIPCODE TO STRING OF THE FORM "XXXXX"
        DEF FN.ZIP$(X)=MID$(STR$(X),1,5)

        REM  CONVERT PHONE NUMBER TO STRING OF FORM "XXX-XXXX"
        DEF FN.FONE$(X$)=LEFT$(X$,3)+"-"+RIGHT$(X$,4)

        REM  CONVERT DATE TO STRING OF FORM "XX/XX/XX"
        DEF FN.DATE$(X)=MID$(STR$(X),1,2)+"/"+     \
                        MID$(STR$(X),3,2)+"/"+     \
                        MID$(STR$(X),5,2)
REM     ANNOUNCE PROGRAM TO OPPERATOR
        PRINT TAB(10);"U.S. ARMY-ALASKA MARS MEMBERSHIP FILE SYSTEM"
        PRINT : PRINT : PRINT : PRINT : PRINT


REM - - ARRAY DEFINITION

        MAX.MEMBERS=100
        DIM CALL$(MAX.MEMBERS+1),CODE$(5)

REM - - CLEAR INDEX ARRAY

        FOR I=1 TO MAX.MEMBERS
           CALL$(I)="?"
        NEXT I

REM - - OPEN MEMBER FILE
        MEMBER$=" B:MEMBERS.LIB"
        FILE MEMBER$(128)


REM - - CREATE INDEX IN CORE
        IF END #1 THEN 11
        FOR I=1 TO MAX.MEMBERS
           READ #1,I;DUM$,DUM$,DUM$,CALL$(I),DUM$,DUM$,DUM$,DUM,DUM$
        NEXT I

11      SIZE=I-1
        PRINT " SIZE OF MEMBER FILE IS CURRENTLY ";SIZE


REM - - DETERMINE FUNCTION TO BE PERFORMED
        CODE$(1)=" ADD"
        CODE$(2)=" CHA"
        CODE$(3)="DEL"
        CODE$(4)=" LIS"
        CODE$(5)=" STOP"

10      FOR J=1 TO 1 STEP 0
           PRINT
           PRINT "FUNCTION (ADD,CHA,DEL,LIS,STOP)";
           INPUT FUNC$
           FOR I=1 TO 5
              IF FUNC$=CODE$(I) THEN    \
                 ON I GOTO 20,30,40,50,60
           NEXT I
        NEXT J
50      STOP

REM - - ADDITIONS TO DATABASE


20      FOR I=1 TO 1 STEP 0
                PRINT
                PRINT "CALLSIGN OF NEW MEMBER (0 TO QUIT)"
                INPUT SIGN$
                IF SIGN$="0" THEN    \
                   PRINT    :\
                PRINT "CURRENT MEMBER FILE SIZE IS ";SIZE    :\
                   PRINT    :\
                   GOTO 10
        REM   CHECK FOR DUPLICATE ENTRY
                GOSUB 30
                IF J<>0 THEN    \
                PRINT SIGN$;" ALREADY ON FILE"    :\
                GOTO 202

        REM   GO GET DATA
                GOSUB 70

        REM   FIND EMPTY SPACE IN FILE
                FOR J=1 TO MAX.MEMBERS
                   IF CALL$(J)="0" THEN 201
                NEXT J
                PRINT "NO MORE ROOM IN DIRECTORY"
                GOTO 202

        REM   WRITE NEW ENTRY TO FILE
        201     CALL$(J)=SIGN$
                GOSUB 75

        REM   IF FILE SIZE EXPANDS, WRITE NEW EOF
                IF J>SIZE THEN :\
                   SIZE = J    :\
                   PRINT #1,(SIZE+1);CHR$(26)
202     NEXT I
REM


REM - - CHANGES TO DATABASE


30      FOR I=1 TO 1 STEP 0
                PRINT
                PRINT " CALL SIGN OF MEMBER(0 TO QUIT)"
                INPUT SIGN$
                IF SIGN$="0" THEN     :\
                   PRINT   :\
                   GOTO 10

        REM     LOCATE CALL IN DIRECTORY, IF NOT FOUND J=0
                GOSUB 80
                IF J=0 THEN    \
                PRINT SIGN$;" NOT ON FILE."    :\
                GOTO 303

        REM     GET CURRENT RECORD FOR MEMBER
                GOSUB 35

        REM     GET NEW ENTRIES FROM OPERATOR
                PRINT "FOR EACH CORRECTION RESPOND TO PROMPT"
                PRINT "WITH - ITEM NAME,NEW ENTRY - (0,0 TO QUIT)"
        301     INPUT ITEM$,ENTRY$
                IF ITEM$="0" THEN 302
                IF ITEM$="NAME1" THEN FIRST$=ENTRY$ : GOTO 301
                IF ITEM$="NAME2" THEN MI$ =ENTRY$ : GOTO 301
                IF ITEM$="NAME3" THEN LAST$ =ENTRY$ : GOTO 301
                IF ITEM$=" CALL" THEN SIGN$ =ENTRY$ : GOTO 301
                IF ITEM$="ADR" THEN ADR$ =ENTRY$ : GOTO 301
                IF ITEM$="CITY" THEN CITY$ =ENTRY$ : GOTO 301
                IF ITEM$="STATE" THEN STATE$ =ENTRY$ : GOTO 301
                IF ITEM$="ZIP" THEN ZIP =VAL(ENTRY$) : GOTO 301
                IF ITEM$="HFONE" THEN HFONE$=ENTRY$  :  GOTO 301
                IF ITEM$="WFONE" THEN WFONE$=ENTRY$  :  GOTO 301
                IF ITEM$=" CLASS" THEN CLASS$ =ENTRY$ : GOTO 301
                IF ITEM$="DATE" THEN EXPIRE =VAL(ENTRY$) : GOTO 301
                PRINT "INVALID ITEM NAME"
                GOTO 301

        REM     WRITE NEW DATA TO FILE
        302     GOSUB 75

        REM     CONFIRM NEW DISK CONTENTS
                GOSUB 35
                GOSUB 37
        REM       DONE, GO DO ANOTHER
303     NEXT I

REM

REM - - DELETE A RECORD

40      FOR I=1  TO 1  STEP Ø
        PRINT
                 PRINT "CALL SIGN OF MEMBER  TO DELETE(Ø  TO QUIT)"
                 INPUT SIGN$
                 IF SIGN$="Ø"  THEN    \
                    PRINT    :\
                    GOTO 1Ø

        REM      FIND CALL IN DIRECTORY , IF NOT FOUND J=Ø
                 GOSUB 3Ø
                 IF J=Ø THEN    \
                 PRINT SIGN$;" NOT ON FILE."    :\
                 GOTO 4Ø1

        REM      CLEAR CALL SIGN FROM THE RECORD
                 TEMP$= SIGN$    :    SIGN$="Ø"
                 GOSUB 75

        REM      CLEAR CALL FROM DIRECTORY
                 CALL$(J)="Ø"

        REM      NOTIFY OPERATOR OF SUCCESFUL REMOVAL
                 PRINT TEMP$;" REMOVED FROM RECORD #";J

4Ø1     NEXT I
REM

REM - - LIST DATABASE CONTENTS
REM     EITHER ONE MEMBER OR ALL MEMBERS CAN BE LISTED

5Ø      FOR I=1  TO 1  STEP Ø
                 PRINT
                 PRINT "CALLSIGN OF MEMBER  TO PRINT (Ø  TO STOP)"
                 INPUT SIGN$
                 IF SIGN$="Ø"  THEN    \
                    PRINT    :\
                    GOTO 1Ø

        REM      'ALL' WILL LIST ENTIRE FILE
                 IF SIGN$=" ALL"  THEN    \
                    PRINT CHR$(12);TAB(15);" "    :\
                    FOR J=1  TO SIZE    :\
                         GOSUB 85    :\
                       GOSUB 9Ø    :\
                    NEXT J    :\
                    PRINT CHR$(12);TAB(15);" "    :\
                    GOTO 1Ø

        REM      FIND MEMBER IN DIRECTORY
                 GOSUB 3Ø
                 IF J=Ø THEN    \
                 PRINT SIGN$;" NOT ON FILE."    :\
                 GOTO 5Ø2

        REM      PRINT CONTENTS OF MEMBER RECORD
                 GOSUB 85
                 GOSUB 9Ø

5Ø2     NEXT I

REM - - SUBROUTINES

REM - - COLLECT DATA FROM TERMINAL

7Ø      PRINT " LAST NAME,FIRST,MIDDLE INIT";
        INPUT LAST$,FIRST$,MI$

        PRINT " STREET ADDRESS OR PO BOX";
        INPUT ADR$
        PRINT " CITY,STATE,ZIPCODE";
        INPUT CITY$,STATE$,ZIP

        PRINT "HOME FONE$,WORK FONE$";
        INPUT HFONE$,WFONE$

        PRINT "LICENSE CLASS,EXPIRATYON DATE";
        INPUT CLAS$,EXPIRE

        RETURN

REM - - WRITE DATA TO MEMBER FILE

75      PRINT #1,J;FIRST$,MI$,LAST$,SIGN$,ADR$,CITY$,STATE$,ZIP,
                  HFONE$,WFONE$,CLAS$,EXPIRE

        RETURN


REM - - LOCATE MEMBER IN DIRECTORY

80      FOR J=1 TO SIZE
           IF SIGN$=CALL$(J) THEN 81
@       NEXT J

        J=0 REM 'NOT FOUND' FLAG

81      RETURN

REM - - GO READ MEMBERS RECORD

85      READ #1,J;FIRST$,MI$,LAST$,SIGN$,ADR$,CITY$,STATE$,
                  ZIP,HFONE$,WFONE$,CLAS$,EXPIRE
        RETURN

REM - - PRINT MEMBER DATA

90      PRINT
        PRINT LAST$;",";FIRST$;",";MI$;" - - - ";SIGN$
        PRINT "CLASS - ";CLAS$,"EXPIRES - ";FN.DATE$(EXPIRE)
        PRINT "PHONE NMBRS :  HOME  ";FN.FONE$(HFONE$)
        PRINT TAB(15);"WORK  ";FN.FONE$(WFONE$)
        PRINT " ADDRESS : ";ADR$
        PRINT TAB(11);CITY$;",";STATE$;",";FN.ZIP$(ZIP)
        PRINT



A>RUN MARSBASE
BASIC-E INTERPRETER - VER 2.2


          U.S. ARMY-ALASKA MARS MEMBERSHIP FILE SYSTEM




SIZE OF MEMBER FILE IS CURRENTLY 33

FUNCTION (ADD,CHA,DEL,LIS,STOP)? LIS

CALLSIGN OF MEMBER TO PRINT (0 TO STOP)
? ACM1DLZ

LAPLANTE,F,E - - - ACM1DLZ
CLASS - ADV    EXPIRES - 80/12/02
PHONE NMBRS :  HOME  243-2957
               WORK  274-7253
ADDRESS : 7151 TALL SPRUCE DR
          ANCHORAGE,AK,99502


CALLSIGN OF MEMBER TO PRINT (0 TO STOP)
? 0
```

```
FUNCTION (ADD,CHA,DEL,LIS,STOP)? DEL
CALL SIGN OF MEMBER TO DELETE(Ø TO QUIT)
? KL7XYZ
KL7XYZ REMOVED FROM RECORD #38

CALL SIGN OF MEMBER TO DELETE(Ø TO QUIT)
? Ø
FUNCTION (ADD,CHA,DEL,LIS,STOP)? ADD

CALLSIGN OF NEW MEMBER (Ø TO QUIT)
? AB7XYZ
LAST NAME,FIRST,MIDDLE INIT? JONES,JOHN,D
STREET ADDRESS OR PO BOX? PO BOSSX 1234-A
CITY, STATE,ZIPCODE? ANCHORAGE,AK,99501
HOME FONE$,WORK FONE$? 1234567,7654321
LICENSE CLASS,EXPIRATION DATE? UNK,811231

CALLSIGN OF NEW MEMBER (Ø TO QUIT)
? Ø

CURRENT MEMBER FILE SIZE IS 38

FUNCTION (ADD,CHA,DEL,LIS,STOP)? CHA

CALL SIGN OF MEMBER(Ø TO QUIT)
? Ø


FUNCTION (ADD,CHA,DEL,LIS,STOP)? LIS

CALLSIGN OF MEMBER TO PRINT (Ø TO STOP)
? AB7XYZ

JONES,JOHN,D - - - AB7XYZ
CLASS - UNK    EXPIRES - 81/12/31
PHONE NMBRS :  HOME  123-4567
               WORK  765-4321
ADDRESS : PO BOX 1234-A
          ANCHORAGE,AK,99501

CALLSIGN OF MEMBER TO PRINT (Ø TO STOP)
? Ø
FUNCTION (ADD,CHA,DEL,LIS,STOP)? CHA

CALL SIGN OF MEMBER(Ø TO QUIT)
? AB7XYZ
FOR EACH CORRECTION RESPOND TO PROMPT
WITH - ITEM NAME,NEW ENTRY - (Ø,Ø TO QUIT)
? HFONE,3217654
? Ø,Ø

JONES, JOHN,D - - - AB7XYZ
CLASS - UNK    EXPIRES - 81/12/31
PHONE NMBRS :  HOME  321-7654
               WORK  765-4321
ADDRESS9: PO BOX 1234-A
          ANCXORAGE,AK,99501

CALL SIGN OF MEMBER(Ø TO QUIT)
? Ø

FUNCTION (ADD,CHA,DEL,LIS,STOP)? DEL

CALL SIGN OF MEMBER TO DELETE(Ø TO QUIT)
? KL7XYZ
KL7XYZ NOT ON FILE.

CALL SIGN OF MEMBER TO DELETE(Ø TO QUIT)
? AB7XYZ
AB7XYZ REMOVED FROM RECORD #38

CALL SIGN OF MEMBER TO DELETE(Ø TO QUIT)
? Ø


FUNCTION (ADD,CHA,DEL,LIS,STOP)? STOP
```

Chapter 8

It's Not a Big Miracle

by Mathew Tekulsky

When Ryan Faber was 13 months old, he was admitted to the hospital for what appeared to be a normal viral disease. His condition worsened, however, and by the time his ailment was identified he was close to going into a coma. Then lab reports came back revealing an extremely high level of sugar and ketones in his urine and blood.

"This was a red flag for diabetes," recalls Dr. Clifford Rubin. The child was saved, but he was in the hospital for eight days. During that time, while sitting at his son's bedside, Steven Faber developed a remarkable and original computer program that monitors Ryan's blood sugar level and thus helps maintain better control over his disease.

How remarkable and original is it?

"Using the computer in this way is a new aspect in the medical care of diabetes," states Dr. Rubin, "and other doctors have shown a great deal of interest in following this up in the future. But it doesn't have any meaning unless it benefits many cases and many people. It could eventually be a routine type of thing in the next ten years."

There's nothing routine, however, about diabetes, an inherited disease that prevents the body from using sugar by blocking the production of insulin in the pancreas. Insulin is what enables blood sugar (glucose) to enter the cells. When glucose can't enter the cells, it builds up in the bloodstream.

But since the cells are still not receiving sugar, the body turns to stored sugar (glycogen) in the liver and muscle. When the level of glycogen is low, energy is then derived from the breakdown of body fat. Some of this fat is turned into toxins called ketones, which also build up in the blood. If too much sugar and ketones are in the blood, the individual can become extremely ill and may require hospitalization. This is what happened to Ryan.

Now 23 months old, Ryan takes daily injections of insulin to keep his blood sugar level under control. But contrary to popular belief, insulin is not a cure for diabetes. In fact, its use is dangerous in and of itself.

Insulin must be administered extremely carefully in relation to the diabetic's food intake and exercise. If too much food is eaten, the insulin is quickly used up and the blood sugar and ketone levels rise. While this abnormal blood chemistry presents an immediate danger, even moderate excesses of sugar in the blood, over a long period of time, can shorten life spans, cause blindness, circulatory diseases, arteriosclerosis, kidney failure and other disorders.

On the other hand, if not enough food is eaten or if too much exercise is had, the insulin will deplete the available sugar and unless more sugar is administered, an insulin reaction can occur, which causes anything from dizziness to a coma and convulsions.

So while he's too young to realize it now, Ryan Faber is in a very precarious position. This is why it's so important to accurately monitor the level of blood sugar and the amount of insulin he has in his system at all times.

And that's exactly what the computer does. It not only charts Ryan's urine sugar by the hour on a daily basis and produces an average for the week, but it charts the two types of insulin (regular, or short-term and longer-acting NPH) he uses on the same graph so that the amount of insulin in the blood can be compared to the amount of urine sugar. When the insulin level is high, the urine sugar should be low.

Why test urine sugar? Basically because urine sugar represents the blood condition of an hour earlier. Normal blood sugar is 80-120 mg. per 100 milliliters. But if the level goes over 140 mg%, sugar spills out in the urine. By calculating the concentration of sugar in the urine, the

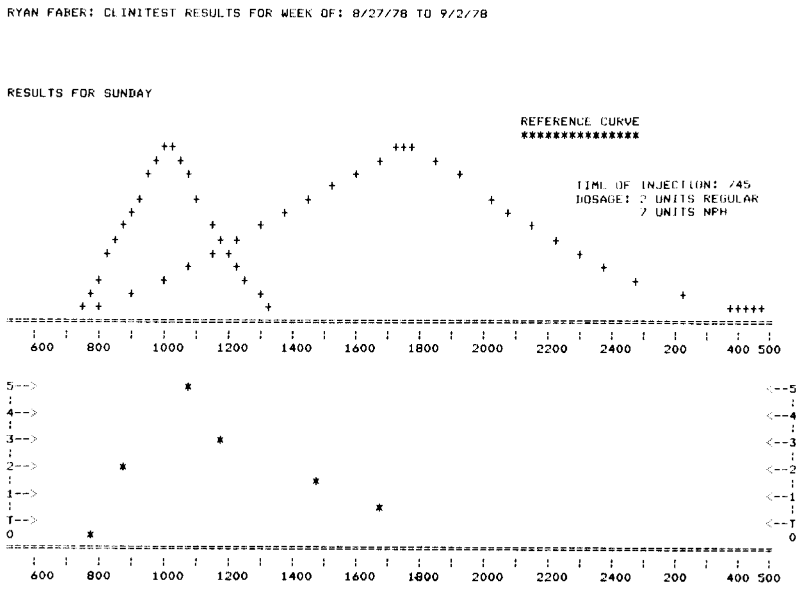

```
RYAN FABER: CLINITEST RESULTS FOR WEEK OF: 8/27/78 TO 9/2/78

RESULTS FOR SUNDAY

REFERENCE CURVE
***************

TIME OF INJECTION: 745
DOSAGE: 2 UNITS REGULAR
        7 UNITS NPH
```

Figure 1

SAMPLE RUN

```
READY
RUN


STARTING DATE FOR RUN? 8/27/78

CLOSING DATE FOR RUN? 9/2/78


DATA FOR SUNDAY

TIME OF INJECTION? 745

DOSAGE (REGULAR,NPH)? 2,7
 600 ?
 700 ?
 800 ? 0
 900 ? 2
 1000 ?
 1100 ? 5
 1200 ? 3
 1300 ?
 1400 ?
 1500 ? 1.5
 1600 ?
 1700 ? .5
 1800 ?
 1900 ?
 2000 ?
 2100 ?
 2200 ?
 2300 ?
 2400 ?
 100 ?
 200 ?
 300 ?
 400 ?
 500 ?
```

degree of excess sugar in the blood can be determined. This is measured on the Clinitest scale, which goes from zero (negative, or no sugar) to five.

The significance of doing this for a diabetic can be explained very simply. It's like watching a dam and trying to determine how much water is on the other side. If there's no water coming over the dam, the reservoir could be almost full or almost empty. Similarly, when there's no sugar in a diabetic's urine, he could be almost normal or have so little sugar in his blood that he's setting himself up for an insulin reaction. Conversely, while a constant stream of water over the top of the dam indicates that the water level is too high, a constant stream of sugar in a diabetic's urine indicates that his blood sugar level is too high.

However, if there were a tiny trickle of water coming over the top of the dam, it would indicate that the water level was just up to the top. Likewise, if just a tiny bit of sugar spills out into the urine, it can be inferred that the blood sugar is just above normal, which is where it should be.

"You want to walk that thin line," explains Faber, "of spilling the absolute minimum excess sugars in the urine to protect against insulin shock and not be so high as to contribute to the deterioration of the body. What the computer does is make it easy for us to perceive what kind of control he's in so we can adjust for changes in insulin needs earlier and more accurately. For example, if he's spilling negative when he should be spilling positive, we just give him a little more food."

The use of the computer is a joint effort between Faber and his wife Debby. During the day, Debby takes about eight urine samples from Ryan and marks down the percentage of sugar in the urine for each hour that a sample was taken. At the end of the week, a full sheet of numbers is brought into the computer room. Steven then loads the program and the computer interrogates for each day and every hour during that day.

"If no tests were taken," he says, "the carriage return is hit without any number being entered. If a test was made, the results of the test are entered as a number at that hour. If the test is negative, zero is entered. If there is a trace of sugar, it's attributed to be a quarter of one percent. One percent is the number one, up to five percent.

"At the end of seven days, the computer prints out individual charts automatically for each day, averages the results for each hour during the week and prints an average chart which gives us a trend projection of what the average day looks like. This is taken to our physician every two months, and a major part of the checkup is going through the book of charts."

The charts themselves are arranged with two insulin curves on top and the urine samples (represented by asterisks), with their respective times when taken, on the bottom. The numbers 0-5 in the vertical column (the letter "T" means a trace of sugar) represent the Clinitest results. High urine sugar levels occur around 1000 and 1800 hours because a meal has just been eaten. But if they were to occur at an unexpected time, it would be cause for alarm. Conversely, too many asterisks in the zero column would mean more sugar should be taken to protect against an insulin reaction. In order to walk that "thin line," therefore, there should be some negative results, some that show a slight spill of sugar (traces, 1's or 2's) and few, if any, 3's, 4's and 5's.

What is the value of charting these statistics, as opposed to simply relying on numerical data?

"Look in the newspaper at the stock market page," says Faber, "and tell me whether the market went up or down and by how much. It's impossible to tell by looking at the individual stocks. But if you look at a chart of the Dow Jones industrials, you an tell at a glance. It's the same with this program."

The basic value is that in a glance it forms a picture of the diabetic's day in terms of his metabolic processes from hour to hour. So instead of having to look at a page full of numbers, it makes it graphically obvious when there's too much or too little sugar in the urine. The second, and more important value is that by averaging the results each week, trends that don't show up in day to day testing do show up before they become obvious.

For example, it's extremely dangerous for Ryan to get sick. Since running a high fever is like exercising, his sugar is used up and an insulin reaction could occur. If he vomits, his supply of sugar is drastically reduced while the flow of insulin continues, so the same thing could happen.

However, if Ryan has an infection, it will attack the insulin before any symptoms of sickness occur. Therefore, sugar and ketones will appear in his urine. When this happens, the Fabers increase the dosage of insulin and prepare for an impending illness.

It is because of benefits like this that the charts give Debby Faber more confidence in her ability to deal with Ryan's condition.

"It makes me feel that when I go to the doctor, I'm giving him the most information about my child that I could possibly get," she says. "With a child this young, you want the doctor to be as knowledgeable about his condition as possible and to be in touch with every detail. So when I walk into my doctor's office, I'm confident that he can see really quickly what's going on.

"I don't have to say, 'Well, this week he was a little bit high on sugar one day and little bit low the next and I'm not really sure how the whole week averaged out.' This way I'm sure that he can look at these charts and say, 'I know exactly what to do with your son. I know exactly how much control he has.' It's making my relationship with my doctor really comfortable and I'm much more secure knowing he can track down a problem in a second."

Dr. Rubin says that "Ryan has been doing so well that I've had very little contact with the Fabers other than their normal visits. This is my first experience with the computer in this respect and it's been a learning experience for me. Although juvenile diabetics are notorious for being in and out of control, the computer helps me to evaluate how much in control Ryan has been." Rubin adds that by knowing what's going on with the patient, medical costs can be reduced by avoiding unnecessary tests.

Although Steven Faber is currently the only person using this computer program for diabetes, that could change soon. It may have been created for a juvenile diabetic, but it can also be used for adults, especially those who are new diabetics or are having difficulty keeping in control. In addition, hospitals that have diabetic floors could make use of it.

However, it's difficult to predict when the use of this technique will become, as Dr. Rubin says, "routine."

"Nobody is going to run out and buy a computer to keep track of their diabetes," says Faber, "but we're now moving towards that period of time so long predicted when the computer will become a commonplace appliance in everybody's home.

"Now how far along that road we are I don't know. I do not believe that we are at the point yet when someone who hears about this, knows nothing about computers and has had no predisposition towards buying a computer will go out and buy a system to do this. Those whose natural bent in this direction has put them on a fence and are thinking about getting a computer might find the impetus to go out and do it. But once you have a computer, it's a simple process."

Faber sees the computer as "one of the few generalized tools that have been available in the history of man," and contends that as uses of the computer become more widespread, diabetics who own computers for a variety of reasons will incorporate this application into their systems.

"Fire is a generalized tool," he says. "It was no more designed to be a cooker of meat than a power of rockets. Because of the nature of the computer and the fact that software can be created to produce desired results, in my way of thinking it's a generalized tool.

"In fact, if the computer weren't there [he's had his for three years], I wouldn't have thought of going out and getting a computer to make a diabetes chart. People should consider the computer as a tool in their lives, a generalized tool that they can mold to their specific needs. And this program just shows that when the tool is there, it's instantly of great use."

Faber's program is a simple one. It's written to lead the user along and can be modified to an individual's needs with a minimum of effort or expertise. The hardware necessary is a microcomputer with 4.5 bytes of memory on top of BASIC and a 132 character printer for the charts.

But despite its value, Faber is quick to point out that "the computer at this point does not keep diabetics in control. Diabetics and parents of diabetics keep diabetics in control. The computer is only an aid to make that easier. And since the urine test is an hour behind, the computer is better at predicting trends than handling emergencies, in which a minute can make a big difference."

But even the computer's predictive potential has some enormous implications.

"Because Ryan's so young," says Faber, "and is going to have to lie with diabetes for so long, we're trying to avoid that long-term damage to his blood vessels that would show up later in life. That's the value of the program for us."

How much value will it have for other diabetics? Only time will tell. But for now, Faber, who donated this program to *Interface Age*, is pleased with what he's done and with the opportunity to make his discovery available to others who need it.

"Obviously I can't do anything about curing diabetes," he says, "so I've just made my contribution this way and I'm doing everything in my power to make it available to the greatest number of people."

PROGRAM LISTING

```
1 REM PROGRAM TO PLOT CHARTS OF URINE SUGAR WITH INSULIN CURVES
2 REM FOR REFERENCE - - PROGRAM WRITTEN IN MICROPOLIS BASIC VERS. 3.0
3 REM PROGRAM DEDICATED TO CLIFFORD RUBIN, M.D., SIR FREDERICK BANTING,
4 REM J. J. R. MACLEOD, C. H. BEST, AND COWS AND PIGS EVERYWHERE
10DIMD(7,24),I(7,2),W$(7,40)
15DIMD$(2,40)
20DATASUNDAY,MONDAY,TUESDAY,WEDNESDAY,THURSDAY,FRIDAY,SATURDAY
25DIMU(7)
30FORI=1TO7:READW$(I):NEXTI
40FORI=0TO7:FORJ=1TO24:D(I,J)=-1:NEXTJ:NEXTI
50C$="0T¦1¦2¦3¦4¦5"
100PRINTCHAR$(12):PRINT:INPUT"STARTING DATE FOR RUN";D$(1)
110PRINT:INPUT"CLOSING DATE FOR RUN";D$(2)
120FORI=1TO7:PRINTCHAR$(12):PRINT:PRINT"DATA FOR ";W$(I):PRINT
130INPUT"TIME OF INJECTION";I(I,0):PRINT
140INPUT"DOSAGE (REGULAR,NPH)";I(I,1),I(I,2)
150FORJ=1TO24
160IFJ>19THEN190
170PRINT(J*100)+500;:INPUTD(I,J):GOTO200
190PRINT(J*100)-1900;:INPUTD(I,J)
200NEXTJ:NEXTI
210GOSUB40000:PRINT:PRINT"RYAN FABER: CLINITEST RESULTS FOR WEEK OF: ";
220PRINTD$(1);" TO ";D$(2):FORI=1TO5:PRINT:NEXTI
230FORI=1TO7:PRINT"RESULTS FOR ";W$(I):PRINT
235U=600:U1=-1
260IFU=I(I,0)THEN285
270U=U+15:U1=U1+1:IF(U1+1)/4=INT((U1+1)/4)THENU=U+40
280GOTO260
285U(I)=U1
290PRINTTAB(65)"REFERENCE CURVE":PRINTTAB(65)"***************"
300PRINTTAB(14+U1)"++";TAB(43+U1)"+++";:IFI=0THENPRINT:GOTO310
305PRINTTAB(100)"TIME OF INJECTION:";I(I,0)
310PRINTTAB(13+U1)"+  +";TAB(41+U1)"+       +";:IFI=0THENPRINT:GOTO330
320PRINTTAB(100)"DOSAGE:";I(I,1);"UNITS REGULAR"
330PRINTTAB(12+U1)"+     +";TAB(38+U1)"+";TAB(51+U1)"+";:IFI=0THENPRINT
335IFI=0THEN350
340PRINTTAB(107)I(I,2);"UNITS NPH"
350PRINTTAB(35+U1)"+"
360PRINTTAB(11+U1)"+        +";TAB(32+U1)"+";TAB(55+U1)"+"
370PRINTTAB(10+U1)"+";TAB(29+U1)"+";TAB(57+U1)"+"
380PRINTTAB(9+U1)"+";TAB(20+U1)"+";TAB(26+U1)"+";TAB(60+U1)"+"
390PRINTTAB(8+U1)"+";TAB(21+U1)"+ +";TAB(63+U1)"+"
400PRINTTAB(7+U1)"+";TAB(20+U1)"+ +";TAB(66+U1)"+"
410PRINTTAB(17+U1)"+";TAB(23+U1)"+";TAB(69+U1)"+"
420PRINTTAB(6+U1)"+";TAB(14+U1)"+";TAB(24+U1)"+";TAB(73+U1)"+"
430PRINTTAB(5+U1)"+     +";TAB(26+U1)"+";TAB(79+U1)"+"
440PRINTTAB(4+U1)"+ +";TAB(27+U1)"+";TAB(85+U1);
450FORIO=85+U1TO95:PRINT"+";:NEXTIO:PRINT
455GOSUB460:GOTO550
460FORIO=1TO96:PRINT"=";:NEXTIO:PRINT
480FORIO=1TO24:PRINT"   ¦";:NEXTIO:PRINT
490PRINT"   600";
500FORIO=800TO2400STEP200:PRINTTAB((((IO-600)/100)*4)+2)IO;:NEXTIO
510FORIO=200TO400STEP200:PRINTTAB(74+((IO/100)*4))IO;:NEXTIO
520PRINTTAB(95)"500"
530RETURN
550PRINT:PRINT
560FORIO=12TO4STEP-1:PRINTMID$(C$,IO,1);
562IFIO/2=INT(IO/2)THENPRINT"-->";
570FORJ=1TO24
575PRINTTAB((J*4)-1);
580IFD(I,J)<=(IO-2)/2ANDD(I,J)>(IO-3)/2THENPRINT"*";:GOTO600
590PRINT" ";
600NEXTJ:IFIO/2=INT(IO/2)THENPRINT"<--";:GOTO620
610PRINT"   ";
620PRINTMID$(C$,IO,1)
```

```
630NEXTIO
640FORIO=3TO1STEP-1:PRINTMID$(C$,IO,1);:IFIO/2=INT(IO/2)THENPRINT"-->";
650FORJ=1TO24:PRINTTAB((J*4)-1);
660IFD(I,J)<=(IO-1)/4ANDD(I,J)>(IO-2)/4THENPRINT"*";:GOTO680
670PRINT" ";
680NEXTJ:IFIO/2=INT(IO/2)THENPRINT"<--";:GOTO700
690PRINT"    ";
700PRINTMID$(C$,IO,1)
710NEXTIO
720GOSUB460:PRINT:PRINT:PRINT:PRINT:PRINT
722IFI=1THENPRINT:PRINT
725PRINT:PRINT:PRINT:PRINT:IFI=1THEN730
726PRINT:PRINT:PRINT:PRINT:PRINT
730FORIO=1TO132/3:PRINT"<*>";:NEXTIO:PRINT
735IFI=0THEN800
736PRINT:PRINT:PRINT:PRINT:IFI=1THEN740
737PRINT:PRINT:PRINT:PRINT:PRINT
740PRINT:PRINT:PRINT:PRINT:PRINT:NEXTI
750FORJ=1TO24:K=0:A=0:FORI=1TO7:IFD(I,J)<>-1THENA=A+D(I,J):K=K+1
760NEXTI:IFK=0THEN765
762D(0,J)=A/K
765NEXTJ:I=0
770U1=0:FORQ9=1TO7:U1=U1+U(Q9):NEXTQ9:U1=INT((U1/7)+.5)
780PRINT"AVERAGED RESULTS FOR WEEK OF ";D$(1);" TO ";D$(2)
790PRINT:GOTO290
800GOSUB30000
810END
30000ASSIGN(2,2)
30010RETURN
40000ASSIGN(2,1):RETURN
```

Due to the importance of this article, we recommend that it be copied and used as often as needed. However, proper credit must be given to: INTERFACE AGE, Mat Tekulsky and Steve Faber.

Those of you interested in getting in contact with Steve Faber can write to him at P.O. Box 69200, Los Angeles, CA 90069.

—editor

Chapter 9

Heart Attack: How You Can Predict It and Some Things You Can Do About It

by Leo P. Biese, M.D., F.C.A.P.

Heart disease is the number one killer in the United States, and one form, the acute myocardial infarct, or "coronary," is the most frequent cause of sudden unexpected death. The facts are that by the age of 70 at least one out of every five readers of this book will be dead of a heart attack. Such loss of life is not "preordained" but is a consequence of the way we live.

In recent years, we have been accustomed to reading that eggs, butter, smoking, and all sorts of other things are bad for our hearts. Sometimes we are even given some sort of vague figures about how bad these things are for us. This program will calculate your risk of a heart attack, but far more importantly, it will show the improvement that can be achieved by reducing the factors over which you have some control.

BACKGROUND

For twenty-eight years the Heart and Lung Institute of the National Institutes of Health, U.S. Public Health Service, has been closely studying the population of Framingham, Massachusetts, in an effort to determine the incidence and factors that influence heart and vascular diseases of all types. From a massive program of multivariate regression analysis, seven factors have been isolated as clearly influencing the probability of heart disease. Numerous studies by other research groups in various parts of the country have substantiated the validity of this data applied to the general population of the U.S. as a whole.

An eighteen-year follow-up of the Framingham population was published by the U.S. Government Printing Office in 1974. Dr. Kammel of the Framingham Project and D. McGee of the Biometrics Research Branch kindly provided copies of the statistical data. Neither these researchers nor the National Institutes of Health is in any way responsible for the use to which the author has put this data.

The statistical base is only valid for the ages of 35-65 (45-65 for women) and the accuracy of the program outside these ages has not been determined. It will be noted that the author has "fudged" a bit on the lower age limits. The data, furthermore, applies only to those free of known heart disease at the time the program is run. A previous heart attack, for example, would make the program completely invalid. The data of HDL is further qualified below.

THE PROGRAM

The program is written in MITS 4.0 Disk BASIC and is largely self-explanatory. Formatting and console-switching program lines for the production of a professional report have largely been eliminated in the interest of brevity. Double precision arithmetic is declared in Line 380 and is redundant since the "#" appendage and the D-exponent also signify double precision arithmetic in the MITS format; they are retained only for clarity. The program takes only about two seconds to compute probability in this format and does not constitute a significant time delay. For clarity, the actual published NIH regression coefficients are shown in a Remark statement ('. . . . .) as well as the alternates for all heart and vascular diseases beginning in Line 760.

The only other problems that may occur in translating this program into other versions of BASIC are as follows: In Line 1010 EXP(X) returns e to the power X. The formula for probability is:

$$P = \frac{1}{1 + e^{-sum}}$$

where sum is the total of all the coefficients times their multipliers plus the intercept.

Lines 1380 and 1400 have a PRINT USING statement to avoid printing out the fourteen-place double precision number. If your BASIC does not have this, an appropriate rounding procedure must be used. Finally, multi-letter variables have been used for clarity and you may need to change these to single letters. The inputs have been assigned single letters so they can be recalled for the printout in their original form.

THE RISK FACTORS

Age is the single most important factor, since it is a high multiplier. Unfortunately, it—along with sex—is one that is not under our control. A2 is used as a correction factor for the non-linearity of age with risk, which is actually a quadratic term in the original equations.

Cholesterol is the value of your blood cholesterol in mg/100 ml, and can easily be obtained from your doctor if you don't already know it. The factor is correct for most modern automated and semi-automated laboratory methods, but your result may have to be lowered five to ten percent if an older, manual method was used. Ask your doctor to find out the comparison of his laboratory's value with the "ABELL-KENDAL" reference procedure if there is some doubt. Like age, cholesterol is a non-linear function and CXA is the correction factor for a cross-product term in the original equations.

BP is the resting systolic blood pressure. This is the higher of the two numbers you are usually given as "something over something," and is the peak pressure during the heart's contraction. (The other number is the diastolic pressure when the heart is relaxed.)

ECG refers primarily to evidence of left heart (the side that does most of the work) enlargement as shown on your electrocardiogram. Any other abnormalities in the ECG would also qualify, as would any other evidence of enlargement, such as an x-ray. Heart enlargement is an important indication that your heart may have to work too hard pushing the blood around your body (a bigger pump for a bigger job) and as such is an important indicator of risk. It is, however, often a reversible change.

CIG refers to the history of smoking regularly within the past year.It is a one-time additive factor (0 or 1) to the risk rather than a multiplier. Note that the data is *not* available for increased risk based on how long or how much you smoke, just whether you do or don't. Strangely, the data on smoking in women gave a negative correlation, suggesting that they are better off (though only slightly) for it. This is believed to be an artifact of the population (it is not negative, for example, in the overall heart-vascular risk), but has been retained as given.

GLU refers to glucose intolerance as manifested by a "high blood sugar," sugar in the urine, or a known diagnosis of diabetes. Like smoking and ECG, it is a one-time additive factor rather than a multiplier. It is not known if correction of diabetes reduces the risk.

GLU and ECG are probably the factors which the reader may have the most difficulty in determining, though this information should be readily available from your doctor. If you do not know these, and set them to zero, the probability will be low (if you really did have them) by somewhere between three and ten percent for each factor, and you may want to run the program with various possibilities.

IN is in the X-intercept for the statistical data, the point at which all risk factors in the probability equation would be zero.

HDL, or High Density Lipoprotein, has been very much in the news recently as "the fat that's good for you." This protein and its associated cholesterol seem to protect the heart and indeed, very high levels (over 85) are associated with longevity, often occurring in families of people who customarily live to 90 or 100 without evidence of heart disease.

HDL was not part of the original Framingham criteria, but was from later studies on an older population and was based on an eight-year projection rather than a six-year one. For these reasons, it is not part of the "official" U.S. Government criteria. The figures appear, however, quite reliable. Unlike the other factors, it is a multiplier of the previously determined risk above. A value of 45 (55 in women) means that your overall risk is unchanged. Below this, the overall risk is multiplied up to approximately three times, depending upon the level. Values above this decrease your risk progressively. It is quite a new test, and if you have not had yours measured yet, a zero will bypass this part of the program. An exciting prospect is that it may be controllable in the near future. As of yet, however, only alcohol (and fish, to a very slight degree) have been shown to have any significant effect in raising your HDL. Alcohol does indeed raise HDL, but since cirrhosis is also a prominent cause of death, the reader is cautioned to wait until further research has clarified the prospects of controlling HDL before going off on a toot.

COM is your risk compared to the same age group (without regard to HDL), in which BP = 105, CHO = 185, and CIG, GLU, and ECG are all zero. This is the part of the program that will pointedly show you the decrease in risks that can be obtained by a little clean living. It was programmed as a simple table based on overall heart and vascular disease to avoid introducing seven more variables and repeating the calculations.

CONCLUSIONS

Put out the cigarette, switch to margarine, exercise a little, lose a little weight, and OFF THE KLINGONS for a good many years to come.

PROGRAM LISTING

```
10  'HEART:    PREDICTS THE PROBABILITY OF CORONARY HEART DISEASE WITHIN
20  '6 YEARS   COMPARED TO A CONTROL POPULATION AND MAY BE RERUN TO SHOW
30  'A DECREASED RISK OBTAINABLE BY REDUCING THE 'CORRECTABLE' VARIABLES
40  'SMOKING,BLOOD PRESSURE AND CHOLESTEROL. COMPARISON IS TO A SAME AGE
50  'GROUP WITHOUT  OTHER RISK FACTORS AND IS BASED ON THE DATA FROM THE
60  '28+ YEAR ONGOING STUDY OF THE POPULATION OF FRAMINGHAM,MASS. BY THE
70  'NATIONAL INSTITUTES OF HEALTH  AND  PUBLISHED BY THE U.S.GOVERNMENT
80  'PRINTING OFFICE IN 1974.  ADDITIONAL DATA ON THE HDL FACTOR IS FROM
90  'FROM LATER PUBLICATIONS AND CORRESPONDENCE (1977)
100 '
110 'PROGRAMED IN MITS 4.0 DISC BASIC BY:
120 ' LEO P. BIESE, MD, FCAP
130 ' NEW ENGLAND CLINICAL LABORATORIES
140 ' 183 MAIN STREET
150 ' TILTON, N.H. (03276)
160 '
170 'LIST OF VARIABLES FOR RISK CORELATION FACTORS:
180 '
190 'AGE    IN YEARS, DIRECTLY CORELATED FOR 35-76 ONLY
200 'A2     A CORRECTION FACTOR FOR THE NON-LINEARITY OF AGE
210 'CHO    FASTING BLOOD CHOLESTEROL IN MG./100ML.
220 'BP     MEAN SYSTOLIC RESTING BLOOD PRESSURE IN MM.HG.
230 'AVE    TABLE OF MINIMAL RISK PROB FOR AGE GROUP
240 'COM    COMPARISON WITH THE MINIMAL RISK GROUP
250 'SUM    SUM OF THE INDIVIDUAL RISK COEFFICIENTS
260 'CIG    SMOKING HISTORY (NON-SMOKER IS ONE ABSTAINING >1 YEAR); 0 OR 1
270 'ECG    EVIDENCE OF LEFT HEART ENLARGEMENT IN THE ECG (0 OR 1)
280 'GLU    GLUCOSE INTOLERANCE=1, NONE=0
```

```
290 'CXA   CHOLESTEROL X AGE, A CORRECTION FACTOR FOR NON-LINEARITY
300 'IN    THE REGRESSION ANALYSIS INTERCEPT
310 'HDL   HIGH DENSITY LIPOPROTEIN LEVEL IN MG/100 ML
320 'SEX$  "M" OR "F"
330 'PROB  THE PROBABILITY OF CORONARY DISEASE
340 'COM   COMPARISON WITH THE MINIMAL RISK GROUP
350 'SUM   SUM OF THE RISK COEFFICIENTS

360 '                        DATA INPUT MODULE

370 PRINT:PRINT:PRINT:PRINT
380 DEFDBL A-Z
390 INPUT"NAME                                 ";N$
400 INPUT"SEX (M OR F)                         ";SEX$
410 INPUT"AGE (IN YEARS)                       ";A
420 LINEINPUT"DATE                               ";DAY$
430 INPUT"DOCTOR/CLINIC                        ";DR$
440 INPUT"COLESTEROL                           ";C
450 INPUT"BLOOD PRESSURE                       ";B
460 INPUT"ABNORMAL ECG (YES=1,NO=0)            ";E
470 INPUT"SMOKER (YES=1,NO=0)                  ";S
480 INPUT"GLUCOSE INTOLERANCE (YES=1,NO=0) ";G
490 INPUT"HDL                                  ";HDL
500 '         CALCULATION MODULE (AS PRINTED BY MITS BASIC
                   AND AS GIVEN BY NIH)

510 IF SEX$ = "F" THEN 630
520 AGE = A*.3754941#'                       0.3754941   MALES
530 A2  = A*A* -2.2165D-03 '              -  0.0022165
540 CHO = C* .0271697#'                      0.0271697
550 BP  = B* .0118041#'                      0.0118041
560 CIG = S* .4389169#'                      0.4389169
570 ECG = E* .5219694#'                      0.5219694
580 GLU = G* .2312953#'                      0.2312953
590 CXA = C*A* -3.718D-04'                -  0.0003718
600 IN  = -19.4532586#'                   - 19.4532586
610 GOTO 870
620 :
630 AGE = A* .3769988#'                      0.3769988   FEMALES
640 A2  = A*A* -2.325D-03'                -  0.0023250
650 CHO = C*.0185534#'                       0.0185534
660 BP  = B*.0132024#'                       0.0132024
670 CIG = S*-.1779578#'                   -  0.1779578
680 ECG = E*.7187659#'                       0.7187659
690 GLU = G*.5602516#'                       0.5602516
700 CXA = C*A* -2.288D-04'                -  0.0002288
710 IN  = -19.9537134#'                   - 19.9537134

720 '    ------------------------------------------------------------------
730 '    ! THE ABOVE COEFFICIENTS ARE SPECIFIC FOR CORONARY HEART DISEASE. !
740 '    ! FOR OVERALL RISK OF HEART DISEASE OF ANY KIND,INCLUDING STROKE, !
750 '    ! SUBSTITUTE THE FOLLOWING COEFFICIENTS:                          !
760 '    !                                                                 !
770 '    ! AGE   0.3743307  MALES          0.2665693  FEMALES              !
780 '    ! A2  - 0.0021165              -  0.0012655                       !
790 '    ! CHO   0.0258102                 0.0160593                       !
800 '    ! BP    0.0156953                 0.0144265                       !
810 '    ! CIG   0.5583013                 0.0395348                       !
820 '    ! ECG   1.0529656                 0.8745090                       !
830 '    ! GLU   0.6020336                 0.6821258                       !
840 '    ! CXA - 0.0003619              -  0.0002157                       !
850 '    ! IN  -19.7709560              - 16.4598427                       !
860 '    ------------------------------------------------------------------

870 SUM = CXA+A2+AGE+CHO+BP+CIG+ECG+GLU+IN
880 PROB= (1/(1+EXP(-SUM))*100)
890 IF HDL=0 THEN 920
900 IF SEX$="M" THEN PROB=PROB*( 6*EXP(-.04 *HDL)):GOTO 920
910 IF SEX$="F" THEN PROB=PROB*(11*EXP(-.043*HDL))

920             COMPARISON WITH MINIMAL RISK GROUP

930 IF SEX$="F" THEN 1030
940 IF A>33 AND A<=37 THEN P=.6
950 IF A>38 AND A<=42 THEN P= 1.1
960 IF A>42 AND A<=47 THEN P= 2
```

```
970 IF A>47 AND A<=52 THEN P= 3.3
980 IF A>52 AND A<=57 THEN P= 4.6
990 IF A>57 AND A<=62 THEN P= 5.9
1000 IF A>62 THEN P= 6.8
1010 GOTO 1090
1020 :
1030 IF A>42 AND A<=47 THEN P= .8
1040 IF A>48 AND A<=52 THEN P= 1.5
1050 IF A>53 AND A<=57 THEN P= 2.3
1060 IF A>57 AND A<=62 THEN P= 3.2
1070 IF A>62 THEN P= 3.9
1090 COM=PROB/P

1180 '                    PRINTOUT MODULE

1190 PRINT:PRINT
1200 PRINT"PATIENT:      ";N$;TAB(50)"DOCTOR/CLINIC:";DR$
1210 PRINT:PRINT"DATE :";DAY$
1220 PRINT:PRINT
1230 PRINT"AGE";A;: IF SEX$="M" THEN PRINT"  MALE";: ELSE PRINT"  FEMALE";
1240 PRINT TAB(40);IF S=1 THEN PRINT"** SMOKER **" ELSE PRINT"NONSMOKER"
1250 PRINT"CHOLESTEROL =";C;"MG/100 ML";TAB(40);
1260 IF G=1 THEN PRINT"GLUOSE INTOLERANT"ELSE PRINT"NO GLUCOSE INTOLERANCE"
1270 PRINT"SYSTOLIC BP =";B;"MM. HG.";TAB(40);
1280 IF E=1 THEN PRINT"ABNORMAL"; ELSE PRINT"NORMAL";
1290 PRINT" ECG"
1300 IF HDL=0 THEN PRINT"HDL NOT EVALUATED":GOTO 1320
1310 PRINT"HDL            = ";HDL;"MG/100 ML"
1320 PRINT:PRINT:PRINT
1330 PRINT"    BASED ON THE ABOVE DATA, THE PROBABILITY OF CORONARY HEART"
1340 PRINT"    DISEASE WITHIN 6 YEARS IS ";
1350 PRINT USING "##.#";PROB;
1360 PRINT"% OR ";
1370 PRINT USING "##.#";COM;
1380 PRINT"  TIMES THAT OF THE"
1390 PRINT"    SAME AGE GROUP WITHOUT OTHER RISK FACTORS.":PRINT
OK
```

SAMPLE RUN

```
NAME                                 ? JOHN DOE
SEX (M OR F)                         ? M
AGE (IN YEARS)                       ? 35
DATE                                  4/28/78
DOCTOR/CLINIC                        ? ANY CLINIC USA
COLESTEROL                           ? 185
BLOOD PRESSURE                       ? 105
ABNORMAL ECG (YES=1,NO=0)            ? 0
SMOKER (YES=1,NO=0)                  ? 0
GLUCOSE INTOLERANCE (YES=1,NO=0)     ? 0
HDL                                  ? 0


PATIENT:      JOHN DOE                           DOCTOR/CLINIC:ANY CLINIC USA

DATE :4/28/78


AGE 35   MALE                           NONSMOKER
CHOLESTEROL = 185 MG/100 ML             NO GLUCOSE INTOLERANCE
SYSTOLIC BP = 105 MM. HG.               NORMAL ECG
HDL NOT EVALUATED


    BASED ON THE ABOVE DATA, THE PROBABILITY OF CORONARY HEART
    DISEASE WITHIN 6 YEARS IS  0.6% OR  0.9  TIMES THAT OF THE
    SAME AGE GROUP WITHOUT OTHER RISK FACTORS.

OK
RUN
```

```
NAME                                  ? FATHER DOE
SEX (M OR F)                          ? M
AGE (IN YEARS)                        ? 65
DATE                                   4/28/78
DOCTOR/CLINIC                         ? THE SAME
COLESTEROL                            ? 335
BLOOD PRESSURE                        ? 195
ABNORMAL ECG (YES=1,NO=0)             ? 1
SMOKER (YES=1,NO=0)                   ? 1
GLUCOSE INTOLERANCE (YES=1,NO=0)      ? 1
HDL                                   ? 45


PATIENT:     FATHER DOE                                    DOCTOR/CLINIC:THE SAME

DATE :4/28/78


AGE 65    MALE                                  ** SMOKER **
CHOLESTEROL = 335 MG/100 ML                     GLUOSE INTOLERANT
SYSTOLIC BP = 195 MM. HG.                       ABNORMAL ECG
HDL         =  45 MG/100 ML


    BASED ON THE ABOVE DATA, THE PROBABILITY OF CORONARY HEART
    DISEASE WITHIN 6 YEARS IS 51.8% OR  7.6  TIMES THAT OF THE
    SAME AGE GROUP WITHOUT OTHER RISK FACTORS.

OK
```

Chapter 10

Shooting Stars

By H. DeMonstoy

INTRODUCTION

This game was written to run on my SWTP 6800 with the CT-1024 terminal. The BASIC program is Tom Pittman's "TINY BASIC 6800" from Itty Bitty Computers. Tom Pittman's Tiny BASIC takes about 2K of memory.

The idea for this game came from BYTE's May 76 issue, but I have seen it in several forms from high level language to a game for the calculator. I wrote this program to fit my needs with the Tiny BASIC and TV terminal.

RULES OF THE GAME

The game is played in a "universe" of stars (★) and black holes (–) that is arranged in a 3×3 matrix. Each position has its own number from one to nine. Position 1, 2 and 3 are across the top, with 4, 5 and 6 through the center and 7, 8 and 9 across the bottom. The first print out has a star in the center (position 5) with black holes all around it. The idea is to shoot stars only, never a black hole, and change the "universe" into eight stars surrounding one black hole.

Each star has its own "galaxy," and when a star is hit, every position in that galaxy changes: all stars become black holes, and all black holes become stars. The first shot must be position 5, the only star in the universe. When this is done, position 5 becomes a black hole and position 2 (above), position 4 (left), position 6 (right), and position 8 (below) all become stars. So it goes on and on. The best score is eleven shots, but watch out for the all black hole "universe" because it is a loser; no stars left to shoot.

The instruction subroutine has a "galaxy" map to follow. Each one is different, which makes for an interesting challenge.

FRAME NUMBERS DISPLAYED FOR ADDED INTEREST

For added interest the frame number is printed in the first line of each frame to keep a running count of the tries. The record here is 56 tries before a win. By the way, I think the quickest loss is in 5 tries, but I may be proven wrong.

CRT TERMINAL CONTROL

As I mentioned before this program was written for the CT-1024 TV terminal and so has some special statements to control the cursor. In my system the "Home Up" is a CONTROL P (DEL), and "Erase to EOL" is a CONTROL U (NAK), but may be different in your system. These do not show in the written program, but are used 4 times. The first is line 149 where, after 3 frames, a new start is made at the top of the screen. The control signals are located in the quotation marks after the PRINT statement. Lines 900, 919 and 939 are similar, and start the three instruction frames.

TTY TERMINAL CONTROL

If you are running this on the TTY terminal, then there will be no use for the control statements used as cursor control. There are three sets of statements at the end of each instruction frame designed to hold that frame until the next one is wanted. These statements and the INPUT Z statements that follow will have no use with a TTY terminal.

RUNNING SHOOTING STARS WITH 4K MEMORY

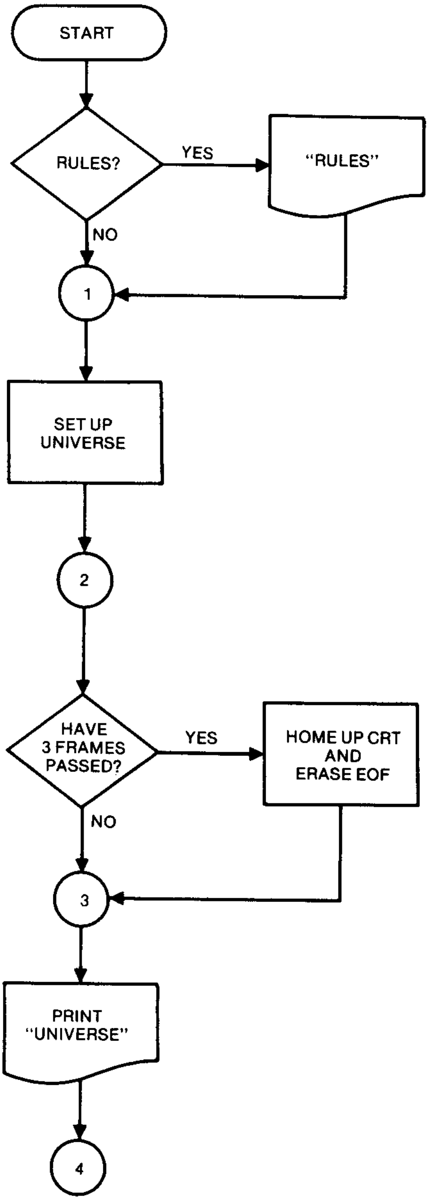
By removing the instructions, this could be run on a machine with only 4K of memory.

I hope you enjoy Shooting Stars.

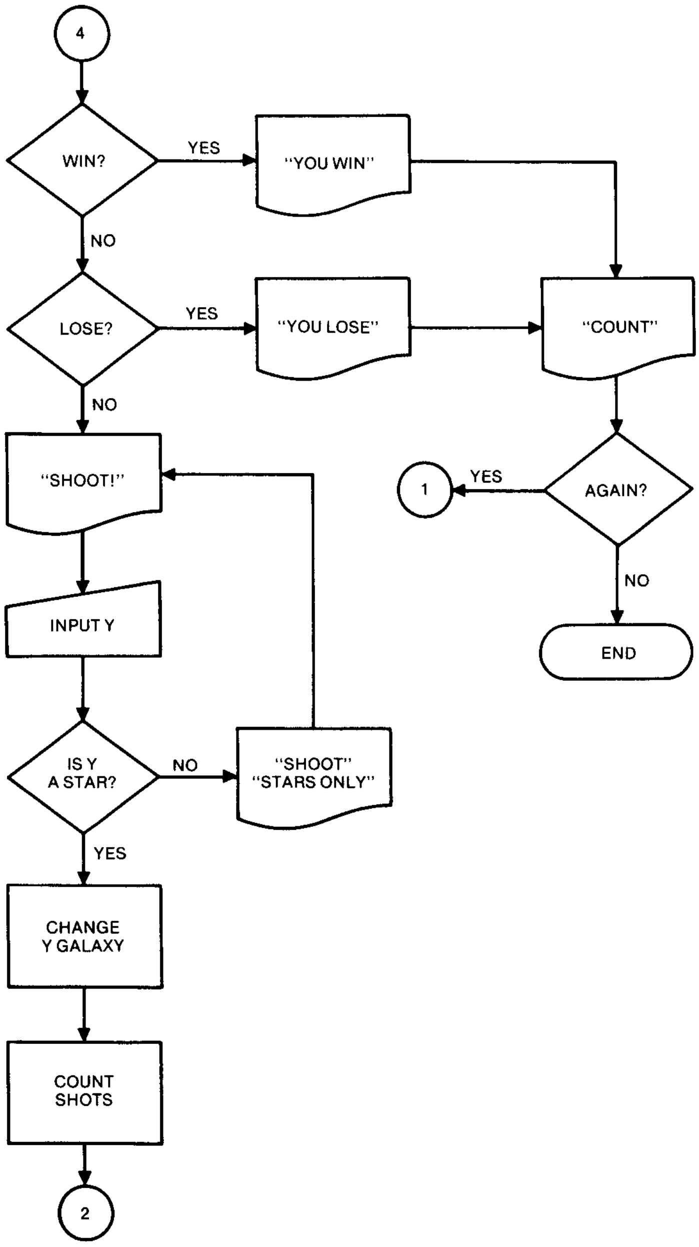
PROGRAM LISTING

```
ØØØ REM SHOOTING STARS IN TBX
ØØ1 REM BY HERMAN DEMONSTOY
ØØ2 REM DATE: 12-18-76
ØØ3 REM MICROCOMPUTER: SWTP'S 68ØØ
ØØ4 REM SUPPORT SOFTWARE: TOM PITTMAN'S TBX
ØØ5 REM MEMORY REQUIRED: 2K FOR TBX & GAME
ØØ6 REM TERMINAL: CT-1Ø24 KEYBOARD-CRT OR TTY
ØØ7 REM
ØØ8 REM
Ø1Ø PRINT "INSTRUCTIONS (1=YES, Ø=NO)";
Ø2Ø INPUT Z
Ø3Ø IF Z=1 GOSUB 9ØØ
1ØØ A=-1
1Ø1 B=-1
1Ø2 C=-1
1Ø3 D=-1
1Ø4 E=1
1Ø5 F=-1
1Ø6 G=-1
1Ø7 H=-1
1Ø8 I=-1
1Ø9 J=Ø
148 PRINT
149 IF J/3*3=J PRINT "";
15Ø IF A=1 PRINT "* ";
151 IF A=-1 PRINT "- ";
```

PROGRAM FLOW DIAGRAM



Flow Chart—Part 1. Shooting Stars



Flow Chart—Part 2. Shooting Stars

```
155 IF B=1 PRINT "* ";
156 IF B=-1 PRINT "- ";
160 IF C=1 PRINT "*  ";J
161 IF C=-1 PRINT "-  ";J
165 IF D=1 PRINT "* ";
166 IF D=-1 PRINT "- ";
170 IF E=1 PRINT "* ";
171 IF E=-1 PRINT "- ";
175 IF F=1 PRINT "*"
176 IF F=-1 PRINT "-"
180 IF G=1 PRINT "* ";
181 IF G=-1 PRINT "- ";
185 IF H=1 PRINT "* ";
186 IF H=-1 PRINT "- ";
190 IF I=1 PRINT "*      ";
191 IF I=-1 PRINT "-      ";
250 IF E=1 GOTO 390
260 IF A+B+C+D+E+G+H+I=8 GOTO 809
270 IF A+B+C+D+E+G+H+I=-8 GOTO 820
390 PRINT "SHOOT";
391 INPUT Y
395 GOSUB 499+Y*10
397 J=J+1
400 GOTO 148
499 PRINT "YOU GAVE UP ON ";J;" TRYS !!!"
500 GOTO 830
509 IF A=-1 GOTO 800
510 A=-A
511 B=-B
512 D=-D
513 E=-E
516 RETURN
519 IF B=-1 GOTO 800
520 A=-A
521 B=-B
522 C=-C
526 RETURN
529 IF C=-1 GOTO 800
530 B=-B
531 C=-C
532 E=-E
533 F=-F
536 RETURN
539 IF D=-1 GOTO 800
540 A=-A
541 D=-D
542 G=-G
546 RETURN
549 IF E=-1 GOTO 800
550 B=-B
551 D=-D
552 E=-E
553 F=-F
554 H=-H
556 RETURN
559 IF F=-1 GOTO 800
560 C=-C
561 F=-F
562 I=-I
566 RETURN
569 IF G=-1 GOTO 800
570 D=-D
571 E=-E
572 G=-G
573 H=-H
576 RETURN
579 IF H=-1 GOTO 800
580 G=-G
581 H=-H
582 I=-I
586 RETURN
589 IF I=-1 GOTO 800
590 E=-E
591 F=-F
592 H=-H
593 I=-I
```

```
596 RETURN
599 RETURN
800 PRINT "HEY !! YOU CAN ONLY SHOOT"
801 PRINT "STARS, NOT BLACK HOLES."
802 GOTO 390
809 PRINT
810 PRINT "YOU WON WITH ";J;" SHOTS"
815 GOTO 830
820 PRINT "YOU LOST WITH ";J;" TRYS."
830 PRINT
831 PRINT "TRY AGAIN (1=YES, 0=NO)";
832 INPUT X
833 IF X=1 GOTO 100
834 IF X=0 GOTO 890
835 PRINT "FOLLOW INSTRUCTIONS "
836 GOTO 831
890 PRINT "HOPE YOU HAD FUN"
895 END
899 REM INSTRUCTION SUBROUTINE
900 PRINT "";
901 PRINT "* * *  THERE ARE STARS"
902 PRINT "- - -  AND BLACK HOLES"
903 PRINT "* * *  IN THE UNIVERSE"
904 PRINT
905 PRINT "1 2 3  YOU SHOOT A STAR *"
906 PRINT "4 5 6  NOT A BLACK HOLE -"
907 PRINT "7 8 9  BY TYPING ITS NUMBER"
908 PRINT
910 PRINT "EACH STAR IS IN A GALAXY."
911 PRINT "WHEN YOU SHOOT A STAR, EVERY-"
912 PRINT "THING IN ITS GALAXY CHANGES."
913 PRINT "ALL STARS BECOME BLACK HOLES,"
914 PRINT "ALL BLACK HOLES BECOME STARS."
915 PRINT "TYPE '2' TO GO ON.";
916 INPUT Z
920 PRINT "GALAXY MAPS:"
921 PRINT
923 PRINT "1 * -  * 2 *  - * 3"
924 PRINT "* * -  - - -  - * *"
925 PRINT "- - -  - - -  - - -"
926 PRINT
927 PRINT "* - -  - * -  - - *"
928 PRINT "4 - -  * 5 *  - - 6"
929 PRINT "* - -  - * -  - - *"
930 PRINT
931 PRINT "- - -  - - -  - - -"
932 PRINT "* * -  - - -  - * *"
933 PRINT "7 * -  * 8 *  - * 9"
934 PRINT
935 PRINT "TYPE '3' TO GO ON";
936 INPUT Z
940 PRINT "PATTERNS TO LOOK FOR:"
941 PRINT
942 PRINT "START     WIN     LOSE"
943 PRINT
945 PRINT "- - -     * * *     - - -"
946 PRINT "- * -     * - *     - - -"
947 PRINT "- - -     * * *     - - -"
948 PRINT
949 PRINT "TYPE '0' TO END GAME"
950 PRINT
951 PRINT "TYPE '4' TO GO ON";
952 INPUT Z
999 RETURN
1000 END
```

```
SHOOTING STARS BY H. DEMONSTOY



 RUN
INSTRUCTIONS (1=YES, 0=NO)? 0

- - -  0
- * -
- - -     SHOOT? 5

- * -  1
* - *
- * -     SHOOT? 2

* - *  2
* - *
- * -     SHOOT? 8

* - *  3
* - *
* - *     SHOOT? 1

- * *  4
- * *
* - *     SHOOT? 9

- * *  5
- - -
* * -     SHOOT? 3

- - -  6
- * *
* * -     SHOOT? 5

- * -  7
* - -
* - -     SHOOT? 7

- * -  8
- * -
- * -     SHOOT? 2

* - *  9
- * -
- * -     SHOOT? 8

* - *  10
- * -
* - *     SHOOT? 5

* * *  11
* - *
* * *
YOU WON WITH 11 SHOTS

TRY AGAIN (1=YES, 0=NO)? 0
HOPE YOU HAD FUN
```

Chapter 11

European Roulette in Color

By W. C. Hoffer

Fortunes have been made and lost at the roulette tables of the world, but after an initial investment for the hardware, you can play roulette without fear of losing your fortune. The game, European Roulette in Color, runs on the Compucolor microcomputer and requires a minimum of 16K bytes of user random access memory.

The original program was written in Dartmouth BASIC. The conversion to a non-pictorial version for Compucolor BASIC was not difficult. The time-consuming effort was animating the game and adding color. Currently, the "wheel" consists of a ball which rolls counterclockwise on one circle, then clockwise on a smaller circle simulating how the ball falls into the winning number.

I have dispersed REMARK statements throughout the program to help the reader determine what happens in each section.

Operating instructions and playing rules are available at the beginning of each game. New players are urged to read them in order to avoid confusion.

However, the program is completely self-instructing and prompts the player for each input as required. Player inputs are checked for validity. Invalid plays are politely refused and the player is asked to play again.

After the playing surface appears on the screen, you will be prompted to PLACE YOUR BETS. The cursor will be positioned where a question is being asked, and the player must respond with either a YES or a NO each time the cursor points to a word. After YES answers are given to the ODD or EVEN, RED or BLACK display, a "$" will appear asking for the bet in dollars. A YES to the COLUMN question will result in a "1-2-3?" display, asking for the column of your choice and your bet. A YES to the NUMBER question will prompt "0-36?", asking for the number and your bet.

When all the bets have been placed, the ball will begin to roll just above the playing surface. When the ball stops, the winning number will appear on the left of the screen, and the BETTING RESULTS sign on the right. The actual results for each of your bets will follow that. Losses are displayed in RED and winnings in GREEN. The cumulative total for the games is kept for you and is constantly displayed on the right of the screen.

The HOUSE wishes you the best of luck and reminds you that you may pick up your winnings at the same location you deposited your losses.

PROGRAM LISTING

```
1ØØ REM EUROPEAN ROULETTE
11Ø REM CONVERTED FROM DARTMOUTH BASIC BY:
115 REM W.C.HOFFER-2721 N. WANDA-SIMI VALLEY,CA-93Ø65
116 REM REQUIRES 16K OF USER RAM
118 REM DISPLAY THE INTRODUCTION
12Ø PLOT6:PLOT33:PLOT12:PLOT27:PLOT11:PLOT14
13Ø PLOT3:PLOT21:PLOT6
14Ø PLOT6:PLOT25
15Ø PRINT"COMPUCOLOR PRESENTS EUROPEAN ROULETTE"
16Ø PLOT3:PLOT8Ø:PLOTØ
17Ø GOSUB 1ØØØØ
19Ø PLOT6:PLOT7:PLOT12
21Ø K=Ø
22Ø PRINT "WELCOME TO THE COMPUCOLOR CASINO AND OUR EUROPEAN ROULETTE TABLE."
23Ø PRINT "WE WISH YOU THE BEST OF LUCK!."
24Ø PRINT:PRINT:PRINT
26Ø PRINT"DO YOU WANT INSTRUCTIONS";
265 PRINT
27Ø INPUT Z$
28Ø IF Z$="NO" THEN 46Ø
29Ø IF Z$="YES" THEN 32Ø
3ØØ GOSUB 196Ø
31Ø GOTO 24Ø
32Ø PLOT6:PLOT7:PLOT12
325 PRINT"   THIS IS A GAME OF ROULETTE.  YOU ARE ALLOWED TO BET ON"
33Ø PRINT"ANY (OR ALL) OF THE FOLLOWING: WHETHER A NUMBER IS ODD OR EVEN,"
34Ø PRINT"COLOR (RED OR BLACK) OF THE NUMBER, A COLUMN OF NUMBERS,"
35Ø PRINT"A NUMBER ITSELF.  NUMBERS RANGE FROM Ø TO 36.  IF A Ø APPEARS,"
36Ø PRINT"THE BANK COLLECTS ALL BETS EXCEPT THOSE BET ON THE NUMBER Ø."
362 PRINT:PRINT
365 PRINT"THE PAYOFFS ARE AS FOLLOWS:"
37Ø PRINT"ODD OR EVEN    1 TO 1"
38Ø PRINT"RED OR BLACK   1 TO 1"
39Ø PRINT"A COLUMN       2 TO 1"
4ØØ PRINT"A NUMBER      35 TO 1"
4Ø5 PRINT
41Ø PRINT"   YOU ARE ALLOWED TO BET FROM $1 TO $1Ø,ØØØ, BUT THE"
42Ø PRINT"TABLE WILL ONLY ACCEPT BETS OF WHOLE DOLLARS."
425 PRINT
43Ø PRINT"(IF YOU WANT TO BET CHANGE--GO USE THE SLOT MACHINES)"
46Ø PRINT "HIT THE SPACE BAR WHEN YOU ARE READY"
47Ø P1=RND(1):IF INP(1)<>96 THEN 47Ø
49Ø GOSUB 5ØØØ :REM DRAW THE TABLE
5ØØ REM TAKE THE BETS
5Ø1 PLOT3:PLOTØ:PLOT1Ø:PLOT6:PLOT7
5Ø2 PRINT"                                        "
5Ø3 GOSUB 63ØØ
5Ø4 GOSUB 652Ø
5Ø5 PLOT14
51Ø PLOT6:PLOT79
52Ø PLOT3:PLOTØ:PLOT2Ø
53Ø PRINT"PLACE YOUR BETS"
54Ø FOR J1=1 TO 5ØØ:NEXT J1
55Ø PLOT6:PLOT2
56Ø PLOT3:PLOTØ:PLOT2Ø
```

```
570 PRINT"PLACE YOUR BETS"
580 REM ODD?
590 PLOT6:PLOT1
600 PLOT3:PLOT4:PLOT24
610 INPUT"-";A$
620 IF A$="NO" THEN 760
630 IF A$="YES" THEN 660
640 GOSUB 1960
642 PLOT3:PLOT4:PLOT24
644 PRINT"          ";
650 GOTO 600
660 REM GET AMOUNT
665 B$="ODD"
670 PLOT3:PLOT9:PLOT24
680 INPUT"$";H
690 IF H<=10000 THEN 720
700 GOSUB 1980
702 PLOT3:PLOT9:PLOT24
704 PRINT"          ";
710 GOTO 670
720 IF H<0 THEN 740
730 IF H=INT(H) THEN 950
740 GOSUB 2010
750 GOTO 702
760 REM EVEN?
770 PLOT3:PLOT5:PLOT28
780 INPUT"-";B$
790 IF B$="NO" THEN 930
800 IF B$="YES" THEN 830
810 GOSUB 1960
812 PLOT3:PLOT5:PLOT28
814 PRINT"          ";
820 GOTO 770
830 REM GET AMOUNT
835 B$="EVEN"
840 PLOT3:PLOT10:PLOT28
850 INPUT"$";H
860 IF H<=10000 THEN 890
870 GOSUB 1980
872 PLOT3:PLOT10:PLOT28
874 PRINT"          ";
880 GOTO 840
890 IF H<0 THEN 910
900 IF H=INT(H) THEN 950
910 GOSUB 2010
920 GOTO 872
930 REM NO ODD/EVEN BET
940 H=0
950 REM RED?
960 PLOT3:PLOT4:PLOT32
970 INPUT"-";C$
980 IF C$="NO" THEN 1120
990 IF C$="YES" THEN 1020
1000 GOSUB 1960
1002 PLOT3:PLOT4:PLOT32
1004 PRINT"          ";
1010 GOTO 960
1020 REM GET AMOUNT
1025 D$="RED"
1030 PLOT3:PLOT9:PLOT32
1040 INPUT"$";I
1050 IF I<=10000 THEN 1080
1060 GOSUB 1980
1062 PLOT3:PLOT9:PLOT32
1064 PRINT"          ";
1070 GOTO 1030
1080 IF I<0 THEN 1100
1090 IF I=INT(I) THEN 1240
1100 GOSUB 2010
1110 GOTO 1062
1120 REM BLACK?
1122 PLOT3:PLOT6:PLOT36
1124 INPUT"-";C$
1126 IF C$="NO" THEN 1220
1128 IF C$="YES" THEN 1136
```

```
1130 GOSUB 1960
1131 PLOT3:PLOT6:PLOT36
1132 PRINT"  ";
1134 GOTO 1122
1136 REM GET AMOUNT
1138 D$="BLACK"
1140 PLOT3:PLOT11:PLOT36
1142 INPUT"$";I
1150 IF I<=10000 THEN 1180
1160 GOSUB 1980
1162 PLOT3:PLOT11:PLOT36
1164 PRINT"
1170 GOTO 1140
1180 IF I<0 THEN 1200
1190 IF I=INT(I) THEN 1240
1200 GOSUB 2010
1210 GOTO 1162
1220 REM NO RED/BLACK BET
1230 I=0
1240 REM COLUMN?
1250 PLOT3:PLOT7:PLOT40
1260 INPUT"-";B1$
1270 IF B1$="NO" THEN 1490
1280 IF B1$="YES" THEN 1310
1290 GOSUB 1960
1292 PLOT3:PLOT7:PLOT40
1294 PRINT"    ";
1300 GOTO 1250
1310 PLOT3:PLOT7:PLOT40
1320 INPUT"1-2 OR 3?";B2
1330 IF B2>0 THEN IF B2<4 THEN 1400
1340 PLOT3:PLOT7:PLOT40
1345 PLOT6:PLOT79
1350 PRINT"1-2 OR 3?"
1360 FOR J1=1 TO 500:NEXT J1
1370 PLOT3:PLOT7:PLOT40
1380 PLOT6:PLOT1
1390 PRINT"          "
1395 GOTO 1310
1400 REM GET AMOUNT
1410 PLOT3:PLOT17:PLOT40
1420 INPUT" $";B8
1430 IF B8<=10000 THEN 1460
1440 GOSUB 1980
1442 PLOT3:PLOT19:PLOT40
1444 PRINT"    ";
1450 GOTO 1410
1460 IF B8<0 THEN 1480
1470 IF B8=INT(B8) THEN 1510
1480 GOSUB 2010
1482 GOTO 1442
1490 REM NO COLUMN BET
1500 B8=0
1510 REM NUMBER BET?
1520 PLOT3:PLOT7:PLOT44
1530 INPUT"-";E$
1540 IF E$="NO" THEN 1790
1550 IF E$="YES" THEN 1580
1560 GOSUB 1960
1562 PLOT3:PLOT7:PLOT44
1564 PRINT"    ";
1570 GOTO 1520
1580 PLOT3:PLOT12:PLOT44
1590 INPUT"0-36?";F
1600 IF F<0 THEN 1630
1610 IF F>36 THEN 1630
1620 IF F=INT(F) THEN 1670
1630 PLOT3:PLOT12:PLOT44
1635 PLOT6:PLOT79
1640 PRINT"0-36?"
1650 FOR J1=1 TO 500:NEXT J1
1652 PLOT3:PLOT12:PLOT44
1654 PLOT6:PLOT2
1656 PRINT"          "
1660 GOTO 1580
1670 REM GET AMOUNT
```

```
1700 PLOT3:PLOT20:PLOT44
1710 INPUT"$";G
1720 IF G<=10000 THEN 1750
1730 GOSUB 1980
1732 PLOT3:PLOT20:PLOT44
1734 PRINT"         ";
1740 GOTO 1700
1750 IF G<0 THEN 1770
1760 IF G=INT(G) THEN 2040
1770 GOSUB 2010
1780 GOTO 1732
1790 REM NO NUMBER BET
1800 G=0
1810 GOTO 2050
1960 PLOT3:PLOT0:PLOT4
1962 PLOT6:PLOT79
1964 PRINT"PLEASE!! YES OR NO";
1968 FOR J1=1 TO 500:NEXT J1
1970 PLOT3:PLOT0:PLOT4
1972 PLOT6:PLOT1
1974 PRINT"                              ";
1976 RETURN
1980 PLOT3:PLOT0:PLOT4
1982 PLOT6:PLOT79
1984 PRINT"HOUSE LIMIT IS $10,000!!";
1985 FOR J1=1 TO 500:NEXT J1
1988 PLOT3:PLOT0:PLOT4
1990 PLOT6:PLOT1
1992 PRINT"                              ";
2000 RETURN
2010 PLOT3:PLOT0:PLOT4
2012 PLOT6:PLOT79
2014 PRINT"FULL DOLLAR BETS ONLY PLEASE"
2016 FOR J1=1 TO 500:NEXT J1
2017 PLOT3:PLOT0:PLOT4
2018 PLOT6:PLOT1
2020 PRINT"
2030 RETURN
2040 REM
2050 REM
2070 T=INT(37*RND(F+H+I+B8+B2+G))
2080 REM THE NUMBER IS
2090 T1=INT(T/10)+1
2091 I1=I
2092 GOSUB 9000
2093 I=I1
2094 PLOT3:PLOT0:PLOT10:PLOT6:PLOT15
2100 ON T1 GOTO 2110,2120,2130,2140
2110 ON T+1 GOTO 2390,2210,2290,2250,2270,2230,2310,2210,2290,2250
2120 ON T-9 GOTO 2270,2350,2190,2330,2170,2370,2150,2350,2190,2210
2130 ON T-19 GOTO 2290,2250,2270,2230,2310,2210,2170,2370,2270,2350STOP
2140 ON T-29 GOTO 2190,2230,2170,2370,2150,2350,2190
2150 PRINTT;" RED,EVEN,COLUMN 1"
2160 GOTO 2400
2170 PRINTT;" RED,EVEN,COLUMN 2"
2180 GOTO 2400
2190 PRINTT;" RED,EVEN,COLUMN 3"
2200 GOTO 2400
2210 PRINTT;" RED,ODD,COLUMN 3"
2220 GOTO 2400
2230 PRINTT;" RED,ODD,COLUMN 2"
2240 GOTO 2400
2250 PRINTT;" RED,ODD,COLUMN 3"
2260 GOTO 2400
2270 PRINTT;" BLACK,EVEN,COLUMN 1"
2280 GOTO 2400
2290 PRINTT;" BLACK,EVEN,COLUMN 2"
2300 GOTO 2400
2310 PRINTT;" BLACK,EVEN,COLUMN 3"
2320 GOTO 2400
2330 PRINTT;" BLACK,ODD,COLUMN 1"
2340 GOTO 2400
2350 PRINTT;" BLACK,ODD,COLUMN 2"
2360 GOTO 2400
2370 PRINTT;" BLACK,ODD,COLUMN 3"
2380 GOTO 2400
```

```
2390 PRINT"THE NUMBER IS  0"
2400 PLOT3:PLOT60:PLOT20
2401 PLOT6:PLOT79
2402 PRINT"BETTING RESULTS"
2403 FOR G9=1 TO 500:NEXT G9
2404 PLOT6:PLOT2:PLOT3:PLOT60:PLOT20
2405 PRINT"BETTING RESULTS"
2409 IF G=0 THEN 2470
2410 IF T=F THEN 2450
2430 G=-G
2440 GOTO 2470
2450 REM
2460 G=35*G
2470 IF H=0 THEN 2660
2480 IF T=0 THEN 2570
2490 IF B$="EVEN" THEN 2540
2500 FOR X=1 TO 35 STEP 2
2510 IF T=X THEN 2610
2520 NEXT X
2530 GOTO 2570
2540 FOR X1=2 TO 36 STEP 2
2550 IF T=X1 THEN 2610
2560 NEXT X1
2570 REM
2580 H=-H
2610 REM
2620 IF B$="EVEN" THEN 2640
2630 PLOT3:PLOT63:PLOT24
2635 GOTO 2642
2640 PLOT3:PLOT64:PLOT28
2642 IF H<0 THEN 2646
2644 PLOT6:PLOT2
2645 GOTO 2650
2646 PLOT6:PLOT1
2650 PRINT" $";H
2660 IF I=0 THEN 2940
2670 IF T=0 THEN 2940
2680 FOR A1=1 TO 9 STEP 2
2690 IF T=A1 THEN 2930
2700 NEXT A1
2710 FOR A2=12 TO 18 STEP 2
2720 IF T=A2 THEN 2830
2730 NEXT A2
2740 FOR A3=19 TO 25 STEP 2
2750 IF T=A3 THEN 2830
2760 NEXT A3
2770 FOR A4=30 TO 36 STEP 2
2780 IF T=A4 THEN 2830
2790 NEXT A4
2800 IF T=26 THEN 2930
2810 IF D$="BLACK" THEN 2880
2820 GOTO 2840
2830 IF D$="RED" THEN 2880
2840 REM
2860 I=-I
2880 REM
2890 IF D$="BLACK" THEN 2920
2900 PLOT3:PLOT63:PLOT32
2910 GOTO 2930
2920 PLOT3:PLOT65:PLOT36
2930 IF I<0 THEN 2936
2932 PLOT6:PLOT2
2934 GOTO 2938
2936 PLOT6:PLOT1
2938 PRINT" $";I
2940 IF B8=0 THEN 3212
2950 IF T=0 THEN 3160
2960 FOR B3=1 TO 34 STEP 3
2970 IF T=B3 THEN 3050
2980 NEXT B3
2990 FOR B4=2 TO 35 STEP 3
3000 IF T=B4 THEN 3070
3010 NEXT B4
3020 FOR B5=3 TO 36 STEP 3
3030 IF T=B5 THEN 3090
3040 NEXT B5
```

```
3050 IF B2=1 THEN 3110
3060 IF B2<>1 THEN 3160
3070 IF B2=2 THEN 3110
3080 IF B2<>2 THEN 3160
3090 IF B2=3 THEN 3110
3100 IF B2<>3 THEN 3160
3110 REM
3130 REM
3140 B8=2*B8
3150 GOTO 3200
3160 REM
3190 B8=-B8
3200 IF B8<0 THEN 3206
3202 PLOT6:PLOT2
3204 GOTO 3208
3206 PLOT6:PLOT1
3208 PLOT3:PLOT66:PLOT40
3210 PRINT" $";B8
3212 IF G=0 THEN 3220
3213 IF G<0 THEN 3216
3214 PLOT6:PLOT2
3215 GOTO 3218
3216 PLOT6:PLOT1
3218 PLOT3:PLOT66:PLOT44
3219 PRINT" $";G
3220 PLOT3:PLOT69:PLOT12
3230 PRINT"                    "
3235 K9=K9+G+H+I+B8
3240 IF K9<0 THEN 3270
3250 PLOT6:PLOT2
3260 GOTO 3280
3270 PLOT6:PLOT1
3280 PLOT3:PLOT68:PLOT10
3290 PRINT" $";K9
4990 FOR I1=1 TO 2000:NEXT I1
4995 GOTO 500
4999 END
5000 REM DRAW THE BOARD
5010 PLOT6:PLOT7:PLOT15:PLOT12
5040 PLOT6:PLOT56
5050 PLOT3:PLOT35:PLOT47
5060 FOR I=1 TO 11:PLOT32:NEXT I
5080 FOR I=44 TO 20 STEP -2
5090 PLOT3:PLOT35:PLOT I
5100 FOR J=1 TO 11:PLOT32:NEXT J
5120 NEXT I
5140 PLOT3:PLOT35:PLOT17
5160 FOR I=1 TO 11:PLOT32:NEXT I
5180 FOR I=20 TO 44 STEP 12
5200 PLOT3:PLOT29:PLOT I
5220 FOR J=1 TO 23:PLOT32:NEXT J
5230 NEXT I
5235 FOR I=28 TO 52 STEP 24 :REM START VERTICAL
5240 FOR J=44 TO 20 STEP-1
5260 PLOT3:PLOT I:PLOT J
5280 PLOT32:NEXT J
5300 NEXT I
5320 FOR I=34 TO 46 STEP 12
5340 FOR J=17 TO 47
5360 PLOT3:PLOT I:PLOT J
5380 PLOT32:NEXT J:NEXT I
5400 FOR I=38 TO 42 STEP 4
5420 FOR J=20 TO 47
5440 PLOT3:PLOT I:PLOT J
5460 PLOT32:NEXT J:NEXT I
5500 REM LABEL THE BOARD
5520 PLOT6:PLOT20
5540 FOR I=29 TO 47 STEP 18
5560 FOR J=21 TO 31
5580 PLOT3:PLOTI:PLOTJ
5590 FOR K=1 TO 5:PLOT32:NEXT K
5600 NEXT J:NEXT I
5610 FOR I=18 TO 19
5615 PLOT3:PLOT35:PLOTI
5620 FOR J=1 TO 11
5640 PLOT32:NEXT J:NEXT I
```

```
5650 FOR I=45 TO 46
5660 FOR J=35 TO 43 STEP 4
5670 PLOT3:PLOTJ:PLOTI
5680 FOR K=1 TO 3:PLOT32:NEXT K
5690 NEXT J:NEXT I
5700 PLOT3:PLOT40:PLOT19:PRINT"0"
5710 PLOT3:PLOT30:PLOT26:PRINT"EVEN"
5720 PLOT3:PLOT48:PLOT26:PRINT"ODD"
5730 FOR I=35 TO 43 STEP 4
5740 PLOT3:PLOTI:PLOT45
5750 PRINT"COL":NEXT I
5760 PLOT3:PLOT36:PLOT46
5765 K=0
5770 FOR I=35 TO 43 STEP 4
5780 K=K+1
5790 PLOT3:PLOTI:PLOT46
5800 PRINTK
5810 NEXT I
5820 PLOT6:PLOT7
5830 PLOT3:PLOT29:PLOT38
5840 PRINT"BLACK"
5850 PLOT6:PLOT15
5860 FOR I=33 TO 43
5870 PLOT3:PLOT47:PLOTI
5880 FOR J=1 TO 5:PLOT32:NEXTJ
5890 NEXT I
5900 PLOT3:PLOT48:PLOT38:PRINT"RED"
5910 PLOT3:PLOT35:PLOT21:PRINT" 1 "
5912 PLOT3:PLOT43:PLOT21:PRINT" 3 "
5914 PLOT3:PLOT39:PLOT23:PRINT" 5 "
5916 PLOT3:PLOT35:PLOT25:PRINT" 7 "
5918 PLOT3:PLOT43:PLOT25:PRINT" 9 "
5920 PLOT3:PLOT43:PLOT27:PRINT"12 "
5922 PLOT3:PLOT39:PLOT29:PRINT"14 "
5924 PLOT3:PLOT35:PLOT31:PRINT"16 "
5926 PLOT3:PLOT43:PLOT31:PRINT"18 "
5928 PLOT3:PLOT35:PLOT33:PRINT"19 "
5930 PLOT3:PLOT43:PLOT33:PRINT"21 "
5932 PLOT3:PLOT39:PLOT35:PRINT"23 "
5934 PLOT3:PLOT35:PLOT37:PRINT"25 "
5936 PLOT3:PLOT39:PLOT37:PRINT"26 "
5938 PLOT3:PLOT43:PLOT39:PRINT"30 "
5940 PLOT3:PLOT39:PLOT41:PRINT"32 "
5942 PLOT3:PLOT35:PLOT43:PRINT"34 "
5944 PLOT3:PLOT43:PLOT43:PRINT"36 "
5946 PLOT6:PLOT7
5950 PLOT3:PLOT39:PLOT21:PRINT" 2"
5952 PLOT3:PLOT35:PLOT23:PRINT" 4"
5954 PLOT3:PLOT43:PLOT23:PRINT" 6"
5956 PLOT3:PLOT39:PLOT25:PRINT" 8"
5958 PLOT3:PLOT35:PLOT27:PRINT"10"
5960 PLOT3:PLOT39:PLOT27:PRINT"11"
5898 PLOT3:PLOT35:PLOT29:PRINT"13"
5964 PLOT3:PLOT43:PLOT29:PRINT"15"
5966 PLOT3:PLOT39:PLOT31:PRINT"17"
5968 PLOT3:PLOT39:PLOT33:PRINT"20"
5970 PLOT3:PLOT35:PLOT35:PRINT"22"
5972 PLOT3:PLOT43:PLOT35:PRINT"24"
5974 PLOT3:PLOT43:PLOT37:PRINT"27"
5975 PLOT3:PLOT35:PLOT39:PRINT"28"
5976 PLOT3:PLOT39:PLOT39:PRINT"29"
5978 PLOT3:PLOT35:PLOT41:PRINT"31"
5980 PLOT3:PLOT43:PLOT41:PRINT"33"
5982 PLOT3:PLOT39:PLOT43:PRINT"35"
5990 REM END OF BOARD
6000 REM PLACE TEXT
6010 PLOT6:PLOT2
6015 PLOT146
37399 3:PLOT0:PLOT20
6040 PRINT"PLACE YOUR BETS":PRINT
6050 PRINT"ODD":PRINT
6060 PRINT"EVEN":PRINT
6070 PRINT"RED":PRINT
6080 PRINT"BLACK":PRINT
6090 PRINT"COLUMN":PRINT
```

```
6100 PRINT"NUMBER"
6110 PLOT3:PLOT60:PLOT20
6120 PRINT"BETTING RESULTS"
6130 PLOT3:PLOT60:PLOT24
6140 PRINT"ODD"
6150 PLOT3:PLOT60:PLOT28
6160 PRINT"EVEN"
6170 PLOT3:PLOT60:PLOT32
6180 PRINT"RED"
6190 PLOT3:PLOT60:PLOT36
6200 PRINT"BLACK"
6210 PLOT3:PLOT60:PLOT40
6220 PRINT"COLUMN"
6222 PLOT3:PLOT60:PLOT44
6224 PRINT"NUMBER"
6230 PLOT3:PLOT0:PLOT10
6250 PLOT3:PLOT60:PLOT10
6260 PRINT"BALANCE $0"
6270 PLOT15:PLOT6:PLOT7
6280 RETURN
6300 REM CLEAR THE BET AREA
6310 PLOT6:PLOT7
6320 PLOT3:PLOT3:PLOT24
6330 PRINT"                       "
6340 PLOT3:PLOT4:PLOT28
6350 PRINT"                      "
6360 PLOT3:PLOT3:PLOT32
6370 PRINT"                       "
6380 PLOT3:PLOT5:PLOT36
6390 PRINT"                     "
6400 PLOT3:PLOT6:PLOT40
6410 PRINT"                    "
6420 PLOT3:PLOT6:PLOT44
6430 PRINT"                      "
6440 RETURN
6500 REM CLEAR THE RESULTS AREA
6510 PLOT6:PLOT7
6520 PLOT3:PLOT63:PLOT24
6530 PRINT"                 "
6540 PLOT3:PLOT64:PLOT28
6550 PRINT"                "
6560 PLOT3:PLOT63:PLOT32
6570 PRINT"                 "
6580 PLOT3:PLOT65:PLOT36
6590 PRINT"               "
6600 PLOT3:PLOT66:PLOT40
6610 PRINT"              "
6620 PLOT3:PLOT66:PLOT44
6630 PRINT"              "
6640 RETURN
9000 REM SPIN THE BALL COUNTER CLOCKWISE
9005 PLOT3:PLOT80:PLOT0
9010 PLOT2:PLOT253:PLOTX:PLOTY
9020 FOR I=1 TO 2
9030 FOR K=1 TO K4
9035 REM PLOT THE WHITE BALL
9040 PLOT255:PLOT6:PLOT7:PLOT2:PLOT253
9050 PLOT X1(K):PLOT Y1(K)
9060 REM PLOT THE BLACK BALL
9070 PLOT255:PLOT6:PLOT0:PLOT2:PLOT253
9080 PLOT X1(K):PLOT Y1(K)
9090 NEXT K:NEXT I
9095 REM END OF CCW SPIN
9100 REM SPIN ONCE CW
9110 PLOT255:PLOT3:PLOT80:PLOT0
9112 PLOT6:PLOT7
9115 PLOT2:PLOT253:PLOTX:PLOTY
9120 FOR I=K4 TO 1 STEP-1
9140 PLOT255:PLOT6:PLOT7:PLOT2:PLOT253
9160 PLOTX2(I)
9170 PLOTY2(I)
9180 PLOT255:PLOT6:PLOT0:PLOT2:PLOT253
9190 PLOTX2(I)
9200 PLOTY2(I)
9210 NEXT I
```

```
9220 PLOT255:PLOT6:PLOT7
9230 REM END OF SPIN
9240 RETURN
10000 REM CALCULATE THE PATH OF THE BALL
10010 DIM X1(64),Y1(64)
10015 DIM X2(64),Y2(64)
10020 S1=10:K1=0:K2=158:K3=S1:K4=0
10030 XX=20:YY=30
10035 X3=17:Y3=27
10040 REM CENTER OF WHEEL
10050 X=80:Y=160
10060 FOR KK=1 TO 4
10070 IF KK<>2 THEN 10090
10080 XX=-XX:K1=158:K2=0:K3=-S1
10085 X3=-X3
10090 IF KK<>3 THEN 10110
10100 YY=-YY:K1=0:K2=158:K3=S1
10105 Y3=-Y3
10110 IF KK<>4 THEN 10130
10120 XX=-XX:K1=158:K2=0:K3=-S1
10125 X3=-X3
10130 FOR I=K1 TO K2 STEP K3
10140 A=I*.01
10150 K4=K4+1
10160 X1(K4)=X+XX*COS(A)
10170 Y1(K4)=Y+YY*SIN(A)
10172 X2(K4)=X+X3*COS(A)
10174 Y2(K4)=Y+Y3*SIN(A)
10180 NEXT I
10190 NEXT KK
10192 RETURN
10195 REM END OF CALCULATION
```

Chapter 12

Child's Play Number Game for Beginning Micro-Bugs

By Karen S. Wolfe

Anyone with the remotest knowledge of computers realizes their great educational possibilities. But you seldom see an elementary program for just that purpose—to educate the young. So here's a quick program that serves two purposes: to provide a mathematical game for youngsters just learning about numbers, and to provide a short, easy program with which beginning programmers can experiment.

WHAT DOES IT DO?

The program initially asks the child to enter a number between 1 and 10. Of course, most children won't be able to read the question right away. You must guide their way through the game the first few times. You'll probably be surprised how quickly the child catches on to the questions and the feedback. So another purpose is served: teaching the child the necessity of learning to read.

Now numbers other than those between 1 and 10 can be entered, but it is best to start with small numbers. Suppose the child enters a 4. The program will then display four stars (★) on the monitor, followed by the number "4". Then, on the first pass through the program, one star followed by a "1" will appear below the four stars. This provides the child with a visual display of a set of "4" objects and a set of "1" object.

Next, the program asks the child to enter the answer for

$4 + 1 = ?$ in algebraic form and also

$$\begin{array}{r} 4 \\ \underline{1} \\ ? \end{array}$$ in column addition form.

The visual set of stars is still on the screen, so the child will initially count all the stars to arrive at an answer. As the child begins to associate numerals with the concept of so many objects, you can rewrite the program so that the stars are no longer printed (see the section on experimentation following).

When the child enters an answer, the program checks to see it is the right answer. If it is right a series of stars is printed and the message "YOU ARE CORRECT—YOU WIN!!!". If the answer is wrong, the feedback is "SORRY, THE ANSWER IS NOT CORRECT, TRY AGAIN!" Even if the child cannot yet read, he or she soon learns these responses and their meanings with just a little help from you.

When the child's answer is wrong, the problem is repeated. When the answer is correct, the program automatically forms a new problem if the child wishes to play another game. The program continues to use the 4 which was originally input, but on the second pass through it asks the child to add a "2" to the 4. On the third pass it adds a "3" and so on through six passes. At that point, the program will ask the child if he or she wants a different input number and if he wants to play another game.

EXPERIMENTATION

The accompanying program is short and uses a number of "FOR-TO" loops, "IF-THEN'S" and "GOTO's" to accomplish its objectives. The beginning programmer should be able to follow the steps with just a little study and then be ready for some experimentation of their own.

First of all, this program was written for North Star BASIC in which multiple line statements are separated by "\". In this program, they are used only to separate PRINT statements which are used for spacing the screen displays.

The C variable is the number entered by the child. The K variable is the internal number which is added to C. When the program is first started, K is set at zero (line 20). In line 60, K is increased by 1. In each successive pass through, the program is cycled back to line 60. When K finally equals 6, the program jumps out of that loop at line 360 and goes to line 400. Now the child is asked if he wants to enter another number for a new game. If he answers yes (Y), the program goes back to line 20 where K is again set at zero.

The beginning programmer can start trying his own ideas for changing this fundamental program. For instance, suppose you want to eliminate the stars from the display. You could simply delete lines 70 through 150. Perhaps you wish to change lines 190 and 200 which form the column addition format in the North Star BASIC. Maybe your BASIC has a different format procedure such as a PRINTUSING statement.

Another possibility is to change the mathematical operation from addition to subtraction, multiplication or division. You must make several changes throughout the program. You'll have to make the appropriate operational sign changes in lines 170 and 220. You'll probably also wish to eliminate the stars, lines 70 through 150.

If you really want to get playful, you can devise a scoring system with a new variable, call it S, and set it initially at 100. Then each time the child answers a problem correctly, 10 is added to S and for each wrong answer, 10 is deducted. This might be done by adding the following statements:

15 LET S = 100; 253 LET S = S – 10;318 LET S = s + 10;
352PRINT;353PRINT"YOUR CURRENT SCORE IS NOW",S.

You may have a better way of setting up a scoring routine. Go ahead, experiment. It's the best way to learn programming. But do let your youngster have a crack at playing this number game once in a while. Remember, micro-bugs come in all sizes.

SAMPLE RUN

```
THIS IS A NUMBER GAME

ENTER A NUMBER FROM 1 TO 10, 4


* * * *     4

*       1

ENTER THE ANSWER FOR:   4 +  1 =  ?

 4.00
 1.00
?5
                   *** *** *** *** *** *** *** ***

                   *** *** *** *** *** *** *** ***


                   YOU ARE CORRECT---YOU WIN ! ! !

                   *** *** *** *** *** *** *** ***

                   *** *** *** *** *** *** *** ***


 DO YOU WANT ANOTHER GAME? (Y/N), Y
* * * *     4

* *     2

ENTER THE ANSWER FOR:   4 +  2 =  ?

 4.00
 2.00
?7

SORRY, THAT ANSWER IS NOT CORRECT, TRY AGAIN!


* * * *     4

* *     2

ENTER THE ANSWER FOR:   4 +  2 =  ?

 4.00
 2.00
?6
```

```
                       *** *** *** *** *** *** *** ***

                       *** *** *** *** *** *** *** ***


                       YOU ARE CORRECT---YOU WIN ! ! !

                       *** *** *** *** *** *** *** ***

                       *** *** *** *** *** *** *** ***


DO YOU WANT ANOTHER GAME? (Y/N) N

DO YOU WANT TO ENTER ANOTHER NUMBER AND PLAY AGAIN?

(Y/N) N
READY
```

PROGRAM LISTING

```
LIST 1

10 DIM Z$(2)
20 LET K=0
30 PRINT "THIS IS A NUMBER GAME"
35 PRINT\ PRINT\ PRINT\ PRINT
40 INPUT "ENTER A NUMBER FROM 1 TO 10 ",C
50 PRINT\ PRINT
60 LET K=K+1
70 FOR I = 1 TO C
80 PRINT "* ",
90 NEXT I
100 PRINT "    ",C
110 PRINT
120 FOR I = 1 TO K
130 PRINT "* ",
140 NEXT I
150 PRINT "     ",K
160 PRINT
170 PRINT "ENTER THE ANSWER FOR: ",C;" + ",K," = "," ?"
180 PRINT
190 PRINT %5F2,C
200 PRINT %5F2,K
210 INPUT A
220 LET A1 = C + K
230 IF A = A1 THEN 270
240 PRINT
250 PRINT "SORRY, THAT ANSWER IS NOT CORRECT, TRY AGAIN!"
255 PRINT\ PRINT
260 GOTO 70
270 FOR I = 1 TO 2
280 PRINT "                  *** *** *** *** *** *** *** *** "
285 PRINT
290 NEXT I
300 PRINT
310 PRINT "                  YOU ARE CORRECT---YOU WIN ! ! ! "
320 PRINT
330 FOR I = 1 TO 2
340 PRINT "                  *** *** *** *** *** *** *** *** "
345 PRINT
350 NEXT I
360 IF K > 5 THEN 400
370 PRINT
380 INPUT " DO YOU WANT ANOTHER GAME? (Y/N) ",Z$
390 IF Z$ = "Y" THEN 60
400 PRINT\ PRINT\ PRINT\ PRINT
410 PRINT "DO YOU WANT TO ENTER ANOTHER NUMBER AND PLAY AGAIN?"
420 PRINT
430 INPUT "(Y/N) ",Z$
440 IF Z$ = "Y" THEN 20
450 END
READY
```

Chapter 13

On A Bi-Lingual Math Tutoring Program

By Marvin Mallon

All forms of programming are challenging and unquestionably, if success follows, create a strong feeling of accomplishment. There is a special sense of gratification, however, that can be identified with writing educational programs. The thought of making the computer serve as a tutoring appliance for the school child is quite inspiring. We are certainly on the threshhold of having the microcomputer serve the needs of the business community as it never has before. We will also soon see the influx of personal computers in the home in numbers undreamed of five years ago. Surely then, one of the more worthy causes to be served will be the teaching of the young.

Of course, a lot has happened in this area already, most notably in school districts in Northern California, due to the prodding and patience of Albrecht, Verplank, et al. The proximity of Hewlett-Packard, Digital Equipment Corporation and IBM has not been wasted in the San Jose-Palo Alto school districts, and Los Angeles teachers, if not in fact the rest of the country, look in envy to their achievements. Hopefully, it is only a matter of time before educators everywhere find the means to enhance classroom activities with the aid of the small computer.

A specific area worthy of extra attention is in dealing with minority students. The Los Angeles Unified School District is responsible for the formative education of thousands of such children, and it encounters special problems where language is a barrier. The computer can help. This article demonstrates that a start has been made and urges greater participation elsewhere to produce constructive programs aimed at overcoming language differences as an obstacle.

Specifically, I wrote a Math Tutoring program which helped my daughter and her friends practice basic arithmetic problems. Some time later I sold and installed a number of microcomputers into Los

Angeles high schools. At one of these schools, San Fernando High, I met and worked with Alan Samow who is on the staff of the mathematics department. He was delighted to have a copy of my tutoring program and in short order, translated it for use by the Spanish-speaking students in his classes. This simply required rephrasing the prompt messages, but he went on to enhance the original program with some fine touches of his own. Following this article is a listing of both the English and Spanish versions with a typical run of the Spanish program included as well.

The program opens with a personalized introduction that sets the mood for the intercourse to follow. It then offers ten problems in either addition, subtraction, multiplication or division. The student also limits the largest numbers he or she will work with based on his or her educational level. Multiplication problems have the further option of being created with the same repetitive multiplier should the student wish to practice his "times table". An option of the division choice is the selection of problems with or without odd remainders.

Answers are greeted with either criticism or praise dependent, of course, on the correctness of the response. These are randomized so as to enhance the "personal touch" of the lesson. The student is given three chances at the right answer before the program moves on to the next question. At the end of the exercise a summary is printed of the score and all missed or troublesome problems are recapped. This tends to enforce the lesson and hopefully encourages the user to repeat difficult areas or go on to larger numbers.

The reader is not only encouraged to use these programs as they appear here, but Mr. Samow and I would be especially gratified if others followed our example and translated new or existing programs so as to make them suitably usable by a broader group of students. Surely nothing can be of greater consequence than the promotion of the computer as a learning tool for children of all ages and background.

SAMPLE RUN

```
RUN
HOLA, CUAL ES TU NOMBRE ?? GERARDO
GUSTO EN CONOCERTE GERADO.  VAMOS A PRACTICAR
MATEMATICAS JUNTOS.
PODEMOS HACER SUMAS, RESTAS
MULTIPLICACION Y DIVISIONES.
ESCRIBE UN SIMBOLO Y HAREMOS 10 PROBLEMAS
           +   -   X  OR /
CUAL EJERCICIO ?? +
VAMOS A TRABAJOR CON 2 NUMEROS DIFFERENTES
CUAL SERA LA CANTIDAD DEL PRIMER NUMERO ?? 987
CUAL SERA LA CANTIDAD DEL SEGUNDO NUMERO? 97
AQUI HAY UN PROBLEMA DE SUMAR
  # 1                 498
               +        4
               -----------
? 502
MUY BIEN!! CONTINUA ASI.
  # 2                 562
               +       63
               -----------
```

```
? 625
ESTA CORRECTO GERADO. TRATA OTRO----
  # 3                      592
                 +          65
                 -----------
? 34
ESTA INCORRECTO GERADO.
  # 3                      592
                 +          65
                 -----------
? 725
ESTA INCORRECTO GERADO.
  # 3                      592
                 +          65
                 -----------
? 345
TUS 3 PRERAS TERMINARON--LO SIENTO!!
  # 4                      271
                 +          61
                 -----------
? 232
ESTA INCORRECTO GERADO.
  # 4                      271
                 +          61
                 -----------
? 332
MUY BIEN!! CONTINUA ASI.
  # 5                      3Ø5
                 +          71
                 -----------
? 376
ERES MUY INTELEGNETE GERADO. AQUI TENEMOS OTRO.
  # 6                      858
                 +          78
                 -----------
? 936
MUY BIEN!! CONTINUA ASI.
  # 7                      543
                 +          47
                 -----------
? 59Ø
ERES MUY INTELEGNETE GERADO. AQUI TENEMOS OTRO.


  # 8                      874
                 +          64
                 -----------
? 938
MUY BIEN!! CONTINUA ASI.
  # 9                      4Ø8
                 +          38
                 -----------
? 746
ERRASTE GERADO.  TRATA OTRA VEZ.
  # 9                      4Ø8
                 +          38
                 -----------
? 546
ESTA INCORRECTO GERADO.
  # 9                      4Ø8
                 +          38
                 -----------
? 446
ESTA CORRECTO GERADO. TRATA OTRO----
  # 1Ø                     1Ø5
                 +          67
                 -----------
? 172
EXCELENTE-EJERCICIO COMPLETO.

         SUMAR

GERADO TIENES 9 CORRECTO Y 1 INCORRECTOS
TEINES PROBLEMAS CON 3 PROBLEMAS
TU GRADO ES  8Ø
```

```
TUVISTE DIFICULTA CON LOS SIGUIENTES PROBLEMAS
        271 + 61 = 332
        4Ø8 + 38 = 446

HIZISTE MAL ESTOS PROBLEMAS
        592 + 65 = 657

TRATAMOS MAS PROBLEMAS ? SI
ESCRIBE UN SIMBOLO Y HAREMOS 1Ø PROBLEMAS
           +    -    X   OR /
CUAL EJERCICIO ?? /
VAMOS A TRABAJOR CON 2 NUMEROS DIFERENTES
CUAL SERA LA CANTIDAD DEL PRIMER NUMERO ?? 89Ø
CUAL SERA LA CANTIDAD DEL SEGUNDO NUMERO? 9
QUIERES PROBLEMAS CON RESIDUO (SOBRANTES)
? NO
AQUI HAY UN PROBLEMA DE DIVICION
# 1 CUANTO ES  342 DIVIDIDO POR 6
? 57
MUY BIEN!! CONTINUA ASI.
# 2 CUANTO ES  119 DIVIDIDO POR 7
? 17
ERES MUY INTELEGNETE GERADO. AQUI TENEMOS OTRO.
# 3 CUANTO ES  74Ø DIVIDIDO POR 5
? 148
MUY BIEN!! CONTINUA ASI.
# 4 CUANTO ES  723 DIVIDIDO POR 3
? 241
MUY BIEN!! CONTINUA ASI.
# 5 CUANTO ES  16Ø DIVIDIDO POR 8
? 2Ø
ERES MUY INTELEGNETE GERADO. AQUI TENEMOS OTRO.
# 6 CUANTO ES  816 DIVIDIDO POR 6
? 136
MUY BIEN!! CONTINUA ASI.
# 7 CUANTO ES  387 DIVIDIDO POR 3
? 129
ESTA CORRECTO GERADO. TRATA OTRO----
# 8 CUANTO ES  24Ø DIVIDIDO POR 4
? 6Ø
MUY BIEN!! CONTINUA ASI.
# 9 CUANTO ES  747 DIVIDIDO POR 3
? 249
ESTA CORRECTO GERADO. TRATA OTRO----
# 1Ø CUANTO ES  57Ø DIVIDIDO POR 5
? 114
EXCELENTE-EJERCICIO COMPLETO.

       DIVICION

GERADO TIENES 1Ø CORRECTO Y Ø INCORRECTOS
TEINES PROBLEMAS CON Ø PROBLEMAS
TU GRADO ES  1ØØ

NINGUN ERROR...TE FELICITO!!

TRATAMOS MAS PROBLEMAS ? NO
HASTA LUEGO POR AHORA.  TE VEO PRONTO!!

OK
```

PROGRAM 1

```
10 REM - MATH TUTOR PROGRAM
20 REM - WRITTEN BY M. MALLON
30 REM - AUGUST, 1976
100 REM - OPENING DIALOG
110 INPUT "HELLO, WHAT'S YOUR NAME";A$
120 PRINT"GLAD TO MEET YOU ";A$;".  LET'S PRACTICE"
130 PRINT"SOME MATHEMATICS TOGETHER."
200 REM - WHICH DRILL?
210 PRINT"WE CAN DO ADDITION, SUBTRACTION"
```

```
220 PRINT"MULTIPLICATION, OR DIVISION."
230 PRINT"TYPE A SYMBOL AND WE WILL DO 10 PROBLEMS:"
232 FOR L=1TO10:P(L)=0:E(L)=0:Q(L)=0:NEXT
234 L=0
235 C=0:N=0:R=0:T=0
237 FOR K=1 TO 10:F(K)=0:G(K)=0:H(K)=0:NEXT
238 K=0
240 PRINT TAB(10);"+  -  X  OR  %"
250 INPUT "WHICH WILL IT BE";B$
270 IF B$<>"+"AND B$<>"-"AND B$<>"X"AND B$<>"%" THEN 240
300 REM - PICKING THE NUMBERS
310 PRINT"WE'LL WORK WITH TWO DIFFERENT NUMBERS"
320 INPUT"HOW BIG CAN THE FIRST ONE BE";N1
332 IF N1=<0 OR N1 >1000 THEN 380
340 INPUT"HOW BIG CAN THE OTHER NUMBER BE";N2
352 IF N2=<0 OR N2 >1000 THEN 380
360 GOTO 400
380 PRINT"THE NUMBERS HAVE TO BE BETWEEN 1 AND 1000"
385 PRINT"LET'S TRY AGAIN----"
390 GOTO 320
400 REM - GOTO DRILL ROUTINE
410 IF B$="+" THEN 600
420 IF B$="-" THEN 700
430 IF B$="X" THEN 800
440 GOTO 900
600 REM - ADDITION
615 S$="PLUS":R$="ADDITION"
620 PRINT"HERE'S AN ADDITION PROBLEM FOR YOU---"
625 GOSUB 1500
630 GOSUB 1000
635 W=A+B
640 IF X=W THEN GOSUB 3000
650 IF X=W THEN 625
660 GOSUB 4000
665 IF X<>A+B AND T=3 THEN 625
670 GOTO 630
700 REM - SUBTRACTION
710 PRINT"HERE'S A PROBLEM IN SUBTRACTION."
715 S$="MINUS"
718 R$="SUBTRACTION"
720 GOSUB 1500
740 IF B>A THEN Y=A:A=B:B=Y
750 GOSUB 1000
760 W=A-B
770 IF X=W THEN GOSUB 3000
780 IF X=W THEN 720
790 GOSUB 4000
795 IF X<>W AND T=3 THEN 720
797 GOTO 750
800 REM - MULTIPLICATION
815 S$="TIMES":R$="MULTIPLICATION"
816 PRINT"DO YOU WANT TO PRACTICE WITH A SPECIAL NUMBER";
817 INPUT C$
818 IF C$="YES" THEN INPUT "WHAT IS THE NUMBER";A
819 IF C$="YES" THEN GOSUB 1510
820 IF C$="YES" THEN GOTO 850
822 PRINT"HERE'S A MULTIPLICATION PROBLEM FOR YOU."
830 GOSUB 1500
850 GOSUB 1000
855 W=A*B
860 IF X=W THEN GOSUB 3000
870 IF X=W AND C$="YES" THEN GOTO 819
875 IF X=W AND C$<>"YES" THEN GOTO 830
880 GOSUB 4000
885 IF X<>W AND T=3 AND C$="YES" THEN GOTO 819
887 IF X<>W AND T=3 AND C$<>"YES" THEN GOTO 830
890 GOTO 850
900 REM - DIVISION
910 S$="DIVIDED BY"
912 R$="DIVISION"
913 U=0
914 PRINT "DO YOU WANT PROBLEMS WITH REMAINDERS";
915 INPUT V$
916 IF V$="YES" THEN U=1
920 PRINT"HERE'S A PROBLEM IN DIVISION."
```

```
930 GOSUB 1500
950 IF B>A THEN Y=A:A=B:B=Y
955 IF U=1 THEN 970
960 Z=INT(A/B):A=B*Z
965 IF A=B THEN 930
970 GOSUB 1000
980 W=INT((A/B)*100)/100
982 IF X>(W-.01)AND X<(W+.01)THEN GOSUB 3000
984 IF X>(W-.01)AND X<(W+.01)THEN 930
986 GOSUB 4000
988 IF X<>W AND T=3 THEN 930
990 GOTO 970
1000 REM - ASK THE QUESTION
1010 PRINT"  #";N+1;" HOW MUCH IS";A;S$;B;:INPUT X
1020 RETURN
1490 REM - CREATE RANDOM NUMBERS
1500 A=INT((N1-2)*RND(1)+2)
1510 B=INT((N2-2)*RND(1)+2)
1520 T=0:RETURN
1990 REM - EXERCISE CONCLUDED
1991 POKE 2020,223:POKE 1234,1
1993 PRINT
1995 PRINTTAB(6);R$:PRINT
2000 PRINTA$;" YOU GOT";C;"RIGHT AND";N-C;"WRONG."
2005 PRINT"YOU HAD TROUBLE WITH";L+K;"PROBLEMS."
2010 PRINT"YOUR SCORE IS";INT(C/N*100)-(5*L)
2020 PRINT:S=L
2022 IFH(1)=0ANDL=0THENPRINT"NO MISTAKES--CONGRATULATIONS!":GOTO2065
2026 PRINT"YOU HAD DIFFICULTY WITH THESE PROBLEMS:"
2027 FOR L=1 TO S
2028 PRINTTAB(6);P(L);S$;E(L);"=";Q(L)
2029 NEXT
2030 L=0:PRINT
2032 IF H(1)=0 THEN GOTO 2062
2038 PRINT"YOU MISSED THESE PROBLEMS:"
2040 FOR K=1 TO N-C
2050 PRINTTAB(6);F(K);S$;G(K);"=";H(K)
2060 NEXT K
2062 K=0
2064 PRINT
2065 POKE 1234,0:POKE 2020,0
2070 INPUT "SHALL WE TRY SOME MORE";D$
2080 IF D$="YES" THEN 230
2090 IF D$ ="NO" THEN 9000
2095 PRINT"YOU MUST ANSWER YES OR NO PLEASE.":GOTO 2070
3000 REM - CORRECT ANSWER!
3010 T=0:C=C+1:N=N+1
3012 M=0
3015 IF N=10 THEN PRINT"EXCELLENT--EXERCISE IS COMPLETE!":GOTO1990
3020 R=INT((4-1)*RND(1)+1)
3040 ON R GOTO 3050,3070,3090
3050 PRINT"THAT'S RIGHT ";A$;". TRY ANOTHER---"
3060 RETURN
3070 PRINT"THAT'S VERY GOOD! KEEP IT UP."
3080 RETURN
3090 PRINT"YOU'RE SURE SMART ";A$;". HERE'S ANOTHER--"
3110 RETURN
4000 REM - WRONG ANSWER
4010 T=T+1
4015 IF T=2 THEN 4030
4020 IF T=3 THEN 5000
4022 M=1
4025 L=L+1:P(L)=A:E(L)=B:Q(L)=W
4030 R=INT((4-1)*RND(1)+1)
4040 ON R GOTO 4050,4070,4090
4050 PRINT"YOU GOOFED ";A$;". TRY AGAIN."
4060 RETURN
4070 PRINT"THAT'S WRONG ";A$;"."
4080 RETURN
4090 PRINT"THINK AGAIN---"
4095 RETURN
5000 REM - 3 MISSES
5010 PRINT"YOUR 3 TRIES ARE UP----TOO BAD."
5020 N=N+1:K=K+1
5030 IF M=1 THEN L=L-1
```

```
5040 F(K)=A:G(K)=B
5042 H(K)=W
5045 IF N<10 THEN RETURN
5050 GOTO 1990
9000 PRINT"GOODBYE FOR NOW. SEE YOU SOON, I HOPE!"
                                                    OK
```

PROGRAM 2

```
1Ø REM - MATH TUTOR PROGRAM
2Ø REM ORIGINAL PROGRAM BY MARVIN MALLON
25 REM RE-WRITTEN AND PREPARED BY ALAN SAMOW SFHS
27 REM TRANSLATION BY LACS
3Ø REM - AUGUST, 1976
1ØØ REM - OPENING DIALOG
11Ø INPUT "HOLA, CUAL ES TU NOMBRE ?";A$
12Ø PRINT "GUSTO EN CONOCERTE ";A$;".  VAMOS A PRACTICAR"
13Ø PRINT"MATEMATICAS JUNTOS."
2ØØ REM - WHICH DRILL?
21Ø PRINT"PODEMOS HACER SUMAS, RESTAS"
22Ø PRINT "MULTIPLICACION Y DIVISIONES."
23Ø PRINT "ESCRIBE UN SIMBOLO Y HAREMOS 1Ø PROBLEMAS"
232 FOR L=1TO1Ø:P(L)=Ø:F(L)=Ø:Q(L)=Ø:NEXT
234 L=Ø
235 C=Ø:N=Ø:R=Ø:T=Ø
237 FOR K=1 TO 1Ø:F(K)=Ø:G(K)=Ø:H(K)=Ø:NEXT
238 K=Ø
24Ø PRINT TAB(1Ø);" +    -    X   OR / "
25Ø INPUT "CUAL EJERCICIO ?";B$
27Ø IF B$<> "+"AND B$<>"-"AND B$<>"X"AND B$<>"/" THEN 24Ø
3ØØ REM - PICKING THE NUMBERS
31Ø PRINT"VAMOS A TRABAJOR CON 2 NUMEROS DIFERENTES"
32Ø INPUT "CUAL SERA LA CANTIDAD DEL PRIMER NUMERO ?";N1
332 IF N1=<Ø OR N1 >1ØØØ THEN 38Ø
34Ø INPUT"CUAL SERA LA CANTIDAD DEL SEGUNDO NUMERO";N2
352 IF N2=<Ø OR N2 >1ØØØ THEN 38Ø
36Ø GOTO 4ØØ
38Ø PRINT"LOS NUMEROS TIENES QUE ESTAR ENTRE DEL 1 A 1ØØØ"
385 PRINT "VAMOS A TRATAR"
39Ø GOTO 32Ø
4ØØ REM - GOTO DRILL ROUTINE
41Ø IF B$="+" THEN 6ØØ
42Ø IF B$="-" THEN 7ØØ
43Ø IF B$="X" THEN 8ØØ
44Ø GOTO 9ØØ
6ØØ REM - ADDITION
615 S$="+":R$="SUMAR"
62Ø PRINT "AQUI HAY UN PROBLEMA DE SUMAR"
625 GOSUB 15ØØ
63Ø GOSUB 1ØØØ
635 W=A+B
64Ø IF X=W THEN GOSUB 3ØØØ
65Ø IF X=W THEN 625
66Ø GOSUB 4ØØØ
665 IF X<>A+B AND T=3 THEN 625
67Ø GOTO 63Ø
7ØØ REM - SUBTRACTION
71Ø PRINT "AQUI HAY UN PROBLEMA DE RESTAR"
715 S$="-"
718 R$="RESTAR"
72Ø GOSUB 15ØØ
74Ø IF B>A THEN Y=A:A=B:B=Y
75Ø GOSUB 1ØØØ
76Ø W=A-B
77Ø IF X=W THEN GOSUB 3ØØØ
78Ø IF X=W THEN 72Ø
79Ø GOSUB 4ØØØ
795 IF X<>W AND T=3 THEN 72Ø
797 GOTO 75Ø

8ØØ REM - MULTIPLICATION
815 S$="X":R$="MULTIPLICACION"
816 PRINT "QUIERES PRACTICAR CON NUMEROS ESPECIALES"
817 INPUT C$
```

```
818 IF C$="SI" THEN INPUT "CUAL ES EL NUMERO";A
819 IF C$="SI" THEN GOSUB 1510
820 IF C$="SI" THEN GOTO 850
822 PRINT "AQUI HAY UN PROBLEMA DE MULTIPLICACION"
830 GOSUB 1500
850 GOSUB 1000
855 W=A*B
860 IF X=W THEN GOSUB 3000
870 IF X=W AND C$="SI" THEN GOTO 819
875 IF X=W AND C$<>"SI" THEN GOTO 830
880 GOSUB 4000
885 IF X<>W AND T=3 AND C$="SI" THEN GOTO 819
887 IF X<>W AND T=3 AND C$<>"SI" THEN GOTO 830
890 GOTO 850
900 REM - DIVISION
910 S$="DIVIDIDO POR"
912 R$="DIVICION"
913 U=0
914 PRINT "QUIERES PROBLEMAS CON RESIDUO (SOBRANTES)"
915 INPUT V$
916 IF V$="SI" THEN U=1
920 PRINT "AQUI HAY UN PROBLEMA DE DIVICION"
930 GOSUB 1500
950 IF B>A THEN Y=A:A=B:B=Y
955 IF U=1 THEN 970
960 Z=INT(A/B):A=B*Z
965 IF A=B THEN 930
970 GOSUB 1100
980 W=INT((A/B)*100)/100
982 IF X>(W-.01)AND X<(W+.01)THEN GOSUB 3000
984 IF X>(W-.01)AND X<(W+.01)THEN 930
986 GOSUB 4000
988 IF X<>W AND T=3 THEN 930
990 GOTO 970
1000 REM ASK THE QUESTION
1001 IF A<10 THEN Z=22:GOTO 1005
1002 IF A<100 THEN Z=21:GOTO 1005
1003 Z=20
1005 IF B<10 THEN O=22:GOTO 1010
1006 IF B<100 THEN O=21:GOTO 1010
1007 O=20
1010 PRINT "   #";N+1;
1012 PRINT TAB(Z);A
1014 PRINT TAB(15);S$;SPC(O-16);B
1015 PRINT TAB(15);"-----------"
1016 INPUT X
1020 RETURN
1100 REM ASK DIVISION QUESTION
1110 PRINT "#";N+1;"CUANTO ES "A;S$;B:INPUT X
1115 RETURN
1490 REM - CREATE RANDOM NUMBERS
1500 A=INT((N1-2)*RND(1)+2)
1510 B=INT((N2-2)*RND(1)+2)
1520 T=0:RETURN
1990 REM - EXERCISE CONCLUDED
1993 PRINT
1995 PRINTTAB(6);R$:PRINT
2000 PRINT A$;" TIENES";C;"CORRECTO Y";N-C;"INCORRECTOS"
2005 PRINT "TEINES PROBLEMAS CON";L+K;"PROBLEMAS"
2010 PRINT "TU GRADO ES ";INT(C/N*100)-(5*L)
2020 PRINT:S=L
2022 IFH(1)=0ANDL=0THEN PRINT "NINGUN ERROR...TE FELICITO!!"
2023 IFH(1)=0AND L=0 THEN 2062
2026 PRINT"TUVISTE DIFICULTA CON LOS SIGUIENTES PROBLEMAS"
2027 FOR L=1 TO S
2028 PRINTTAB(6);P(L);S$;E(L);"=";Q(L)
2029 NEXT
2030 L=0:PRINT
2032 IF H(1)=0 THEN GOTO 2062
2038 PRINT "HIZISTE MAL ESTOS PROBLEMAS"
2040 FOR K=1 TO N-C
2050 PRINTTAB(6);F(K);S$;G(K);"=";H(K)
2060 NEXT K
2062 K=0
2064 PRINT
```

```
2070 INPUT "TRATAMOS MAS PROBLEMAS ";D$
2080 IF D$="SI" THEN 230
2090 IF D$="NO" THEN 9000
2095 PRINT"YOU MUST ANSWER YES OR NO PLEASE.":GOTO 2070
3000 REM - CORRECT ANSWER!
3010 T=0:C=C+1:N=N+1
3012 M=0
3015 IFN=10 THEN PRINT"EXCELENTE-EJERCICIO COMPLETO.":GOTO1990
3020 R=INT((4-1)*RND(1)+1)
3040 ON R GOTO 3050,3070,3090
3050 PRINT "ESTA CORRECTO ";A$;". TRATA OTRO----"
3060 RETURN
3070 PRINT "MUY BIEN!! CONTINUA ASI."
3080 RETURN
3090 PRINT "ERES MUY INTELEGNETE ";A$;". AQUI TENEMOS OTRO."
3110 RETURN
4000 REM - WRONG ANSWER
4010 T=T+1
4015 IF T=2 THEN 4030
4020 IF T=3 THEN 5000
4022 M=1
4025 L=L+1:P(L)=A:E(L)=B:Q(L)=W
4030 R=INT((4-1)*RND(1)+1)
4040 ON R GOTO 4050,4070,4090
4050 PRINT "ERRASTE ";A$;".  TRATA OTRA VEZ."
4060 RETURN
4070 PRINT "ESTA INCORRECTO ";A$;"."
4080 RETURN
4090 PRINT "PIENSA OTRA VEZ"
4095 RETURN
5000 REM - 3 MISSES
5010 PRINT "TUS 3 PRERAS TERMINARON--LO SIENTO!!"
5020 N=N+1:K=K+1
5030 IF M=1 THEN L=L-1
5040 F(K)=A:G(K)=B
5042 H(K)=W
5045 IF N<10 THEN RETURN
5050 GOTO 1990
9000 PRINT "HASTA LUEGO POR AHORA.  TE VEO PRONTO!!"
                                                    OK
```

Chapter 14

The Personal Management Program

by Carl Townsend

Now that you have got your computer going you have probably found yourself with dozens of projects that need to be done. The computer has multiplied your effectiveness, but how can it manage your time and projects?

Why not use the computer itself to manage the projects? The computer can monitor an inventory of all your existing projects, the relative priority and any deadline dates. This little managing program performs a sort each time the projects are listed, sorting the list in priority and date order.

EASY AS A-B-C

Control begins with planning. What are your long term goals? How do you plan to accomplish these? Can you define some short term goals that would be steps to the larger goals?

1. What resources do you need? (people and materials)
2. What education will you need?

You should try to translate the larger blue sky goals to smaller, realizable and specific subgoals. List these subgoals as projects on a sheet of paper without assigning any priorities. List any relevant deadline dates (income taxes, for example, may have to be mailed before the fifteenth of April). Then go over this list and mark an "A" by those that will give you the most value or need most immediate attention. Those next in order should get a "B," and the next a "C." These values are relative based on your goals and the rewards you envision. For more help on this, read Alan Lakein's *How to Get Control of Your Time and Your Life.*[1] This list will be used as the input to the computer program and should be updated weekly. A sample list is shown in Figure 1.

```
PROJECT LIST —
MAILOUT PROGRAM —         Build Module — A
                          Sort Module — B
                          List Module — B
                          Extraction Module — B
                          Update Module — B
                          Documentation — A
Build system for delivery: New Book — A
                          Letters — A
                          Next Newsletter — A
                          Church Proposal — A
                          Business Proposal — A
Read:           Winter's Book — A
                Magazines — A
Software:       Nutrition Program — C
```

Figure 1. Initial Project List

USING THE PROGRAM

The program as listed runs in the new commercial BASIC with 24K of memory. It can easily be modified for BASIC-E, Microsoft BASIC, or North Star BASIC. The sort is performed on random files on the disk, so only enough memory is needed for two strings at a time. The sort using a PerSci disk and CP/M takes only a few seconds and the disk head will not drop from the disk during the entire sort. The flow diagram is shown in Figure 2. The program is in Program 1.

GOING TO LARGER PROJECTS

Once the program is mastered, visit your local library and locate books on critical path charting, Ghant charts, and PERT charts. Study up on these and find the methods that seem best for your projects. Use the project codes in this program listing to flag phases of larger projects and you will find this program can monitor progress on your larger projects. Always list the larger project name as well as the name of the particular phase, as:

```
A 12/20/77 MAILOUT—Build Module
B 12/31/77 MAILOUT—Sort Module
C 12/31/77 MAILOUT—List Module
D 01/15/77 MAILOUT—Extraction Module
E 01/15/77 MAILOUT—Search Module
```

Start the program and the program will request on operation mode:

l —sort and list the current file
p—sort and print the current file
u—update the current file
e—exit

The "l" and "p" mode use the same routine, with the only difference being the output device. Both output the project list (see Figure 3). The

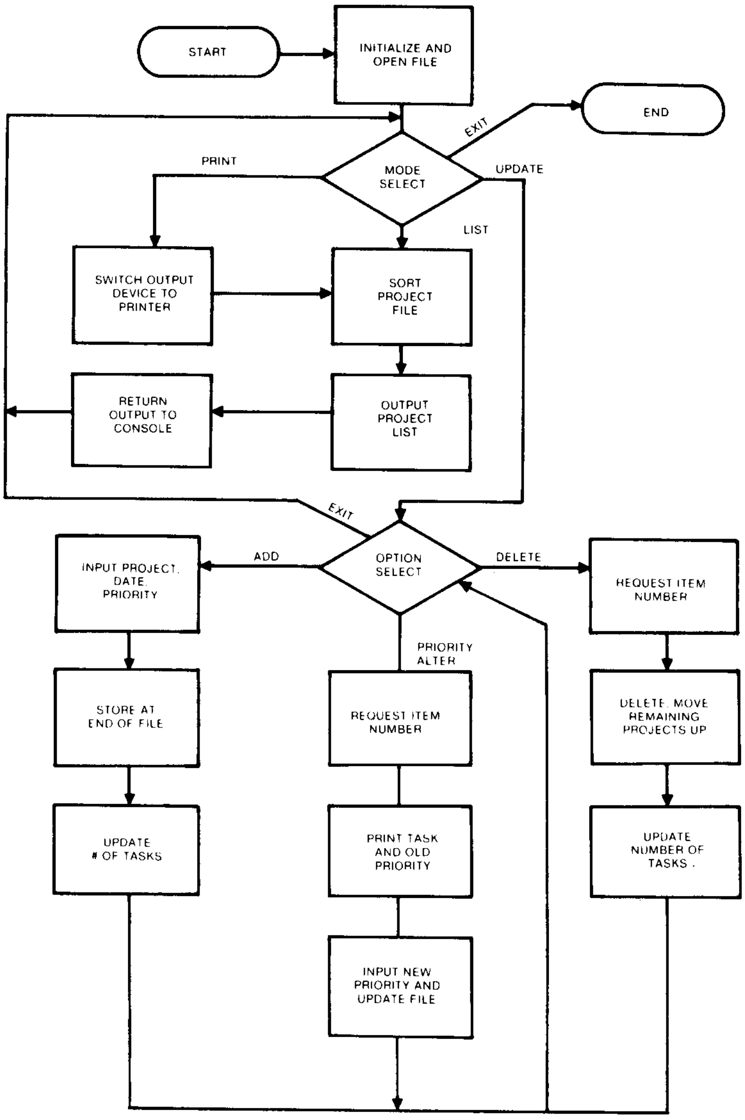


Figure 2. Personal Management Program Flow Diagram

```
crun task
CRUN VER 1.01
Personal Management Program
Create a New File? n
Option (update,list,print or exit): p
Date: 01/16/78

Personal Task Schedule
Date: 01/16/78
 1  A  01/14/78  Write letters (Center)
 2  A  01/14/78  Next Patterns
 3  A  01/18/78  Continue writing book
 4  A  01/18/78  Document Personal Management Program
 5  A  01/18/78  Mailout — Sort Module
 6  A  01/25/78  Do coordinate maps
 7  A  01/18/78  Repair tape recorder for church
 8  A  01/21/78  Read Winter's Book
 9  A  02/15/78  Church Information System — Proposal
10  B  01/25/78  Mailout — Update Module
11  B  01/25/78  Mailout — Extraction Module
12  B  01/31/78  Read — Corporation Books
13  B  01/31/78  Read — book on volunteer organizations
14  B  01/25/78  Business Proposal
15  B  02/15/78  Assemble 2SIO and Floppy Disk Interface
16  C  02/28/78  Income Tax — Calculate
17  C  02/28/78  Nutrition Program
18  C  02/15/78  Checkout 2SIO and interface
Option (update,list,print or exit): e
```

Figure 3. Sample Project Listing

exit mode returns the user to the operating system. The update mode, when requested, asks the user for the type of update option desired:

p—alter priority of specified item
a—add an item
d—delete an item
e—exit update mode

The "a" option in the update mode permits the user to add any project to the current list. The project is appended to the end of the current file (see Figure 4). The first record on the file always is incremented by one as each project is added. The file is not sorted until the next list or print mode. As each project is added a project "number" is assigned to the project automatically based on its order in the file. This number is used to later priority on the project or for deletions.

The "d" option (delete) permits the user to delete any project from the file. The "item number" is requested, and the user inputs the cur-

```
crun task
CRUN VER 1.01
Personal Management Program
Create a New File? n
Option (update, list, print or exit): u
Priority alter,delete,add or exit: a
Job Description: Accounting Program
Priority: B
Date: 01/31/78
19     B  01/31/78  Accounting Program
Priority alter,delete,add or exit: d
Item # : 12
Priority alter,delete,add or exit: p
Item # : 5
Job: A     01/18/78     Mailout — Sort Module
New Priority: B
Priority alter,delete,add or exit: e
Option (update,list,print or exit): l
Date: 01/17/78

Personal Task Schedule
Date: 01/17/78
 1  A  01/14/78  Write letters (Center)
 2  A  01/14/78  Next Patterns
 3  A  01/18/78  Continue writing book
 4  A  01/18/78  Document Personal Management Program
 5  A  01/25/78  Do coordinate maps
 6  A  01/18/78  Repair tape recorder for church
 7  A  01/21/78  Read Winter's Book
 8  A  02/15/78  Church Information System — Proposal
 9  B  01/18/78  Mailout — Sort Module
10  B  01/25/78  Mailout — Update Module
11  B  01/25/78  Mailout — Extraction Module
12  B  01/31/78  Read — book on volunteer organizations
13  B  01/25/78  Business Proposal
14  B  02/15/78  Assemble 2SIO and Floppy Disk Interface
15  B  01/31/78  Accounting Program
16  C  02/28/78  Income Tax — Calculate
17  C  02/28/78  Nutrition Program
18  C  02/15/78  Checkout 2SIO and interface
Option (update,list,print or exit): e
```

Figure 4. Sample of Update

rent project number for the project to be deleted. The project is deleted from the file, and all subsequent projects "moved up" to recover the lost space. The first record on the file that indicates the total number of projects is decremented by one. This alters the project numbers for all subsequent projects in the file, and in multiple delete operations the user should start from the bottom of the listing and work up.

The "p" option alters the priority of any project in the file. The current project number is entered and the project priority date, and name printed. The user enters the new priority. The file is then updated.

The "e" or exit option returns the user from the update mode. No sorts are made until the next list or print mode select.

PROGRAM APPLICATIONS

Notice that the program, as written, does not request the name of the input file with the projects. This is because each person can have their own disk, personal management program, and project file. The management program is stored as TASK, with the project file containing the name of the person who uses the disk. Everybody has their own list of projects, and even the project priorities will vary among family members.

The bubble sort of this program will help you to keep the progress of the project in order. The sort will also keep all phases of a particular program together if they have the same priority and deadline date.

PROGRAM LISTING

```
CBASIC COMPILER VER 1.00
Rem Personal Management Program
rem by Carl Townsend
rem last edit date: 1/15/78
       carl.asc$="carl.asc"
       print "Personal Management Program":print
       true = -1
       Input "Create a New File? ";i$
       if left$(i$,1)="y" then goto 80
       open carl.asc$ recl 80 as 1
10     n=1
       if end # 1 then 90
       read # 1,1;q
       input "Option (update,list,print or exit): ";i$
       if left$(i$,1)="l" then goto 21
       if left$(i$,1)="p" then goto 20
       if left$(i$,1)="u" then goto 30
       if left$(i$,1)="e" then goto 90
       goto 10
20 rem print mode
       lprinter
21 rem list mode
       input "Date: ";d$
       print:print "Personal Task Schedule":print
       print "Date: ";d$
       print
       flag = true
       if end # 1 then 25
       while flag = true
              n=2
              flag = false
              read # 1,n;i$
              while q-n
```

```
                     read # 1,n+1;j$
                     if left$(i$,1)>left$(j$,1) then
                            k$=i$:i$=j$:j$=k$:flag = true
                     print # 1,n;i$
                     i$=j$
                     n=n+1
              wend
              print # 1,n  ;j$
       wend
25     n=2
       if end # 1 then 10
27     read # 1,n;i$
       i$="  "+i$
       print using "##";n-1;:print i$
       n=n+1
       if (n-1) <> q then goto 27
       console
       goto 10
30 rem update mode
       read # 1,1;q
       input "Priority alter,delete,add or exit: ";i$
       if left$(i$,1)="p" then goto 40
       if left$(i$,1)="d" then goto 50
       if left$(i$,1)="a" then goto 60
       if left$(i$,1)="e" then goto 10
       goto 30
40 rem priority alter option
       input "Item # :";n
       if n>(q-1) then goto 30
       read # 1,n+1;i$
       print "Job: ";i$
       input "New Priority: ";p$
       i$=left$(p$,1)+mid$(i$,2,len(i$)-1)
       print # 1,n+1;i$
       goto 30
50 rem delete option
       input "Item # :";n
       if n>(q-1) then goto 30
       if n=q-1 then print # 1,1;q-1:goto 30
       for s=n+1 to q-1
       read # 1,n+2;i$
       print # 1,n+1;i$
       n=n+1
       next s
       read # 1,1;s
       print # 1,1;s-1
       goto 30
60 rem add option
       input "Job Description: ";j$
       input "Priority: ";p$
       input "Date: ";d$
       i$=left$(p$,1)+"  "+left$(d$,8)+"  "+j$
       q=q+1
       if len(i$)>78 then i$=left$(i$,78)
       print q-1;"  ";i$
       print # 1,q;i$
       print # 1,1;q
       goto 30
80 rem create new file
       create carl.asc$ recl 80 as 1
       n=1:print # 1,1;n
       goto 10
90 rem close files
       close 1
       stop
       end
NO ERRORS DETECTED
```

Appendix A

8080 Instruction Set

Some of the instructions include references to specific registers. For instance, the MOVr_1, r_2 instruction takes the value stored in register r_2 (called the **source** register) and stores it in register r_1 (called the **destination** register). The three bit value used to identify the source is shown as SSS in the op code; the three bit value used to identify the destination is shown as DDD. The correspondences between registers and three bit values are

	register	**SSS or DDD**
(accumulator)	A	111
	B	000
	C	001
	D	010
	E	011
	H	100
	L	101

Thus, the op code for MOV A, B is

$$01\overbrace{111}^{\text{DDD}}\underbrace{000}_{\text{SSS}}$$

The 8080 (and the 8085) has five **status flags** (also called **condition flags** or **condition codes**).

status flag	**abbreviation**	**meaning for instructions which affect the flag**
zero	Z	if the result of an instruction is zero (all bits 0), *zero* = 1, otherwise *zero* = 0.
sign	S	if the leftmost bit of the result is 1, *sign* = 1, else 0.
carry	CY	if an arithmetic operation resulted in a carry or a borrow out of the leftmost bit, *carry* = 1.
parity	P	if there is an even number of 1's in the result, *parity* = 1.
auxiliary carry	AC	carry out of bit 3. Used when dealing with binary coded decimal values (see DAA instruction).

Other abbreviations

The second and third bytes of multibyte instructions are identified as $byte_2$ and $byte_3$.

pc	—	the **program counter** (a 16-bit register)
r	—	a register, one of A, B, C, D, E, H, L
sp	—	the **stack pointer** (a 16-bit register)

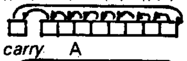
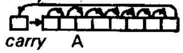
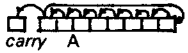
*Thanks to Intel Corp. for permission to include this material.

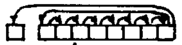
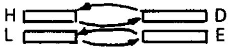
			flags affected							
mnemonic	**description**	**op code**	**Z**	**S**	**P**	**CY**	**AC**	**clock cycles**	**length**	**meaning**
ACI	Add immediate to A with carry	11001110	•	•	•	•	•	7	2	(A)←(A) + (*byte*$_2$) + (*carry*)
ADC M	Add memory to A with carry	10001110	•	•	•	•	•	7	1	(A)←(A) + ((H)(L)) + (*carry*)
ADC *r*	Add register to A with carry	10001SSS	•	•	•	•	•	4	1	(A)←(A) + (*r*) + (*carry*)
ADD M	Add memory to A	10000101	•	•	•	•	•	7	1	(A)←(A) + ((H) (L))
ADD *r*	Add to register to A	10000SSS	•	•	•	•	•	4	1	(A)←(A) + (*r*)
ADI	Add immediate to A	11000110	•	•	•	•	•	7	2	(A)←(A) + (*byt*$_2^2$)
ANA M	**And** memory with A	10100110	•	•	•	•	•	7	1	(A)←(A)∧((H)(L))
ANA *r*	**And** register with A	10100SSS	•	•	•	•	•	4	1	(A)←(A)∧(*r*)
ANI	**And** immediate with A	11100110	•	•	•	•	•	7	2	(A)←(A)∧(*byte*$_2$)
CALL	Call unconditional	11001101						17	3	((*sp*) - 1)←(high order byte of *pc*) ((*sp*) - 2)←(low order byte of *pc*) (*sp*)←(*sp*) - 2 (*pc*)←(*byte*$_3$) (*byte*$_2$) i.e. (*pc*) is pushed on the stack, control is transferred to (*byte*$_3$ (*byte*$_2$)
CC	Call on carry	11011100						11/17	3	same as CALL if (*carry*) = 1 otherwise continue in sequence (i.e. (*pc*)←(*pc*) + 3)
CM	Call on minus	11111100						11/17	3	same as CALL if (*sign*) = 1 otherwise go on
CMA	Complement A	00101111						4	1	(A)←one's complement of (A) i.e. all 0's become 1's and vice versa
CMC	Complement carry	00111111				•		4	1	(*carry*)←($\overline{carry}$)
CMP M	Compare memory with A	10111110	•	•	•	•	•	7	1	set status flags based on the value of (A) - ((H) (L)). (H), (L), and (A) remain unchanged
CMP *r*	Compare register with A	10111SSS	•	•	•	•	•	4	1	set status flags based on value of (A) - (*r*). (A) and (*r*) remain unchanged

			flags affected							
mnemonic	description	op code	Z	S	P	CY	AC	clock cycles	length	meaning
CNC	Call on no carry	11010100						11/17	3	same as CALL if (*carry*) = 0, otherwise go on
CNZ	Call on not zero	11000100						11/17	3	same as CALL if (*zero*) = 0, otherwise go on
CP	Call on positive	11110100						11/17	3	same as CALL if (*sign*) = 0, otherwise go on
CPE	Call on parity even	11101100						11/17	3	same as CALL if (*parity*) = 1, otherwise go on
CPI	Compare immediate with A	11111110	•	•	•	•	•	7	2	set status flags based on value of (A) - (*byte*$_2$). (A) remains unchanged
CPO	Call on parity odd	11100100						11/17	3	same as CALL if (*parity*) = 0, otherwise go on
CZ	Call on zero	11001100						11/17	3	same as CALL if (*zero*) = 1, otherwise go on
DAA	Decimal adjust A	00100111	•	•	•	•	•	4	1	convert the 8-bit value in A into 2 BCD digits (in A), used after additions on BCD values
DAD B	Add B & C to H & L	00001001				•		10	1	(H)(L)←(H)(L) + (B)(C)
DAD D	Add D & E to H & L	00011001				•		10	1	(H)(L)←(H)(L) + (D)(E)
DAD H	Add H & L to H & L	00101001				•		10	1	(H)(L)←(H)(L) + (H)(L)
DAD SP	Add stack pointer to H & L	00111001				•		10	1	(H)(L)←(H)(L) + (*sp*)
DCR M	Decrement memory	00110101	•	•	•		•	10	1	((H)(L))←((H)(L)) - 1
DCR *r*	Decrement register	00DDD101	•	•	•		•	5	1	(*r*) ← (*r*) - 1
DCX B	Decrement B & C	00001011						5	1	(B)(C)←(B)(C) - 1
DCX D	Decrement D & E	00011011						5	1	(D)(E)←(D)(E) - 1
DCX H	Decrement H & L	00101011						5	1	(H)(L)←(H)(L) - 1
DCX SP	Decrement stack pointer	00111011						5	1	(*sp*) ← (*sp*) - 1
DI	Disable interrupts	11110011						4	1	ignore interrupt requests from now on
EI	Enable interrupts	11111011						4	1	respond to interrupt requests from now on
HLT	Halt	01110110						7	1	stop, i.e. don't carry out any further instructions.

			flags affected					clock		
mnemonic	**description**	**op code**	**Z**	**S**	**P**	**CY**	**AC**	**cycles**	**length**	**meaning**
IN	Input	11011011						10	2	place a value from the input port specified by (*$byte_2$*) in A
INR M	Increment memory	00110100	•	•	•		•	10	1	((H)(L))←((H)(L)) + 1
INR *r*	Increment register	00DDD100	•	•	•		•	5	1	(*r*) ←(*r*) + 1
INX B	Increment B & C registers	00000011						5	1	(B)(C)←(B)(C) + 1
INX D	Increment D & E registers	00010011						5	1	(D)(E)· (D)(E) + 1
INX H	Increment H & L registers	00100011						5	1	(H)(L)←(H)(L) + 1
INX SP	Increment stack pointer	00110011						5	1	(*sp*) ←(*sp*) + 1
JC	Jump on carry	11011010						10	3	same as JMP if (*carry*) + 1, otherwise go on in sequence
JM	Jump on minus	11111010						10	3	same as JMP if (*sign*) = 1, otherwise go on
JMP	Jump unconditional	11000011						10	3	(*pc*)←(*$byte_3$*) (*$byte_2$*)
JNC	Jump on no carry	11010010						10	3	same as JMP if (*carry*) = 0, otherwise go on
JNZ	Jump on not zero	11000010						10	3	same as JMP if (*zero*) = 0, otherwise go on
JP	Jump on positive	11110010						10	3	same as JMP if (*sign*) = 0, otherwise go on
JPE	Jump on parity even	11101010						10	3	same as JMP if (*parity*) = 1, otherwise go on
JPO	Jump on parity odd	11100010						10	3	same as JMP if (*parity*) = 0, otherwise go on
JZ	Jump on zero	11001010						10	3	same as JMP if (*zero*) = 1, otherwise go on
LDA	Load A direct	00111010						13	3	(A)←((*$byte_3$*) (*$byte_2$*))
LDAX B	Load A indirect	00001010						7	1	(A)←((B)(C))
LDAX D	Load A indirect	00011010						7	1	(A)←((D)(E))
LHLD	Load H & L direct	00101010						16	3	(L)←((*$byte_3$*) (*$byte_2$*))
										(H)←((*$byte_3$*) (*$byte_2$*) + 1

mnemonic	description	op code	flags affected Z	S	P	CY	AC	clock cycles	length	meaning
LXI B	Load immediate register Pair B & C	00000001						10	3	(B) ← (*byte*$_3$) (C) ← (*byte*$_2$)
LXI D	Load immediate register Pair D & E	00010001						10	3	(D) ← (*byte*$_3$) (E) ← (*byte*$_2$)
LXI H	Load immediate register Pair H & L	00100001						10	3	(H) ← (*byte*$_3$) (L) ← (*byte*$_2$)
LXI SP	Load immediate stack pointer	00110001						10	3	(*sp*) ← (*byte*$_3$) (*byte*$_2$)
MVI M	Move immediate memory	00110110						10	2	((H)(L)) ← (*byte*$_2$)
MVI *r*	Move immediate register	00DDD110						7	2	(*r*) ← (*byte*$_2$)
MOV M,*r*	Move register to memory	01110SSS						7	1	((H)(L)) ← (*r*)
MOV *r*,M	Move memory to register	01DDD110						7	1	(*r*) ← ((H)(L))
MOV r_1,r_2	Move register to register	01DDDSSS						5	1	(r_1) ← (r_2), r_1 is the **destination** r_2 is the **source**
NOP	No-operation	00000000						4	1	don't do anything except increment (*pc*) to get the next instruction
ORA M	**Or** memory with A	10110110	•	•	•	0	0	7	1	(A) ← (A) ∨ ((H)(L))
ORA *r*	**Or** register with A	10110SSS	•	•	•	0	0	4	1	(A) ← (A) ∨ (*r*)
ORI	**Or** immediate with A	11110110	•	•	•	0	0	7	2	(A) ← (A) ∨ (*byte*$_2$)
OUT	Output	11010011						10	2	send (A) to the port specified by (*byte*$_2$)
PCHL	H & L to program counter	11101001						5	1	(*pc*) ← (H) (L), i.e. jump to (H) (L)
POP B	Pop register pair B & C off stack	11000001						10	1	(C) ← ((*sp*)), (B) ← ((*sp*) + 1), (*sp*) ← (*sp*) + 2
POP D	Pop register pair D & E off stack	11010001						10	1	(E) ← (*sp*)), (D) ← ((*sp*) + 1), (*sp*) ← (*sp*) + 2
POP H	Pop register pair H & L off stack	11100001						10	1	(L) ← ((*sp*)), (H) · ((*sp*) + 1), (*sp*) ← (*sp*) + 2

			flags affected							
mnemonic	description	op code	Z	S	P	CY	AC	clock cycles	length	meaning
POP PSW	Pop A and Flags off stack	11110001	•	•	•	•	•	10	1	(*status flags*) ← ((*sp*)), (A) ← ((*sp*) + 1), (*sp*) ← (*sp*) + 2
PUSH B	Push register Pair B & C on stack	11000101						11	1	((*sp*) - 1) ← (B), ((*sp*) - 2) ← (C), (*sp*) ← (*sp*) - 2
PUSH D	Push register Pair D & E on stack	11010101						11	1	((*sp*) - 1) ← (D), ((*sp*) - 2) ← (E), (*sp*) ← (*sp*) - 2
PUSH H	Push register Pair H & L on stack	11100101						11	1	((*sp*) - 1) ← (H), ((*sp*) - 2) ← (L), (*sp*) ← (*sp*) - 2
PUSH PSW	Push A and Flags on stack	11110101						11	1	((*sp*) - 1) ← (A), ((*sp*) - 2) ← (*status flags*)
RAL	Rotate A left through carry	00010111				•		4	1	
RAR	Rotate A right through carry	00011111				•		4	1	
RC	Return on carry	11011000						5/11	1	same as RET if (*carry*) = 1, otherwise go on in sequence
RET	Return	11001001						10	1	(*pc*) ← ((*sp*) + 1) ((*sp*)), (*sp*) ← (*sp*) + 2, i.e. jump to address stored on the top of the stack
RLC	Rotate A left	00000111				•		4	1	
RM	Return on minus	11111000						5/11	1	same as RET if (*sign*) = 1, otherwise go on
RNC	Return on no carry	11010000						5/11	1	same as RET if (*carry*) = 0, otherwise go on
RNZ	Return on not zero	11000000						5/11	1	same as RET if (*zero*) = 0, otherwise go on
RP	Return on positive	11110000						5/11	1	same as RET if (*sign*) = 0, otherwise go on
RPE	Return on parity even	11101000						5/11	1	same as RET if (*parity*) = 1, otherwise go on
RPO	Return on parity odd	11100000						5/11	1	same as RET if (*parity*) = 0, otherwise go on

mnemonic	description	op code	flags affected Z	S	P	CY	AC	clock cycles	length	
RRC	Rotate A right	00001111						4	1	 *carry* A
RST	Restart	11AAA111						11	1	push (*pc*) on the stack, then (*pc*) ← 0000000000AAA000 (for responding to interrupts)
RZ	Return on zero	11001000						5/11	1	same as RET if (*zero*) = 1, otherwise go on
SBB M	Subtract memory from A with borrow	10011110	•	•	•	•	•	7	1	(A)←(A) - ((H) (L)) - (*carry*)
SBB *r*	Subtract register from A with borrow	10011SSS	•	•	•	•	•	4	1	(A)←(A) - (*r*) - (*carry*)
SBI	Subtract immediate from A with borrow	11011110	•	•	•	•	•	7	2	(A)←(A) - ($byte_2$) - (*carry*)
SHLD	Store H & L direct	00100010						16	3	(($byte_3$) ($byte_2$)) ← (L) (($byte_3$) ($byte_2$) + 1) ← (H)
SPHL	H & L to stack pointer	11111001						5	1	(*sp*) ← (H) (L)
STA	Store A direct	00110010						13	3	(($byte_3$) ($byte_2$)) ← (A)
STAX B	Store A indirect	00000010						7	1	((B) (C))←(A)
STAX D	Store A indirect	00010010						7	1	((D) (E)) ← (A)
STC	Set carry	00110111				•		4	1	(*carry*) ← 1
SUB M	Subtract memory from A	10010110	•	•	•	•	•	7	1	(A)←(A) - ((H) (L))
SUB *r*	Subtract register from A	10010SSS	•	•	•	•	•	4	1	(A)←(A) - (*r*)
SUI	Subtract immediate from A	11010110	•	•	•	•	•	7	2	(A)←(A) - ($byte_2$)
XCHG	Exchange D & E, H & L Registers	11101011						4	1	

			flags affected					ciock		
mnemonic	description	op code	Z	D	P	CY	AC	cycles	length	meaning
XRA M	**Exclusive or** memory with A	10101110	•	•	•	0	0	7	1	(A) ← (A) ⊕ ((H) (L))
XRA *r*	**Exclusive or** register with A	10101SSS	•	•	•	0	0	4	1	(A) ← (A) ⊕ (*r*)
XRI	**Exclusive or** immediate with A	11101110	•	•	•	0	0	7	2	(A ← (A) ⊕ (*byte*$_1$)
XTHL	Exchange top of stack, H & L	11100011						18	1	(L) ← ((*sp*)) , (H) ← ((*sp*) + 1)
	Note: the newer 8085 microprocessor has all the above instructions plus two more:									
RIM	read interrupt mask	00100000						4	1	(A) ← (*interrupt mask*)
SIM	set interrupt mask	00110000						4	1	(*interrupt mask*) (A)

Appendix B

Available Back Issues

April 1976	2.25
October 1976	2.25
November 1976	2.25
March 1977	2.25
May 1977	
June 1977	2.50
July 1977	2.50
August 1977	2.50
September 1977	2.50
October 1977	2.50
November 1977	2.50
February 1978	2.75
April 1978	2.75
July 1978	2.75
August 1978	2.75
September 1978	2.75
October 1978	2.75
November 1978	2.75
December 1978	2.75

March 1979	2.75
April 1979	2.75
May 1979	2.75
June 1979	2.75
July 1979	2.75
August/September 1979	2.75

INTERFACE AGE MAGAZINE Dept. BI P.O. Box 1234,
Cerritos, CA 90701

Name (print) ______________________

Address ______________________

City______________ State__________ Zip__________

Price includes 50¢ for postage and handling.

VISA# ______________ M/C# ______________

Exp. Date __________ Signature ______________

Index

INTERFACE AGE™ MAGAZINE

For Businessmen...
Professionals...
Students...
You...

Happiness is...a computer magazine you can understand

Step into the exciting world of computing with INTERFACE AGE Magazine. Written and edited expressly for those who want to get more out of their life and business through the use of computers. Join the 85,000 plus who make reading INTERFACE AGE a priority each month. Enjoy articles that not only tell you how, but show you how. Each issue of INTERFACE AGE contains projects, programs, games and reports on and about people and their computers.

Learn how easy it is to own and operate your own computer system at home or in your business. Explore the many ways a computer can make money for you. Keep up to date with the latest new products and developments. Only INTERFACE AGE brings you all this plus much, much more.

The magazine leading the way. . .
bringing people and technology together

Please enter my subscription to INTERFACE AGE for:

☐ 1 year U.S. $18.00	☐ 2 years U.S. $30.00
☐ 1 year Canada/Mexico $20.00	☐ 2 years Canada/Mexico $34.00
☐ 1 year International Surface Mail $28.00	☐ 1 year International Air Mail $50.00

Make Check or Money Order (U.S. Funds drawn on U.S. Bank) payable to:
INTERFACE AGE Magazine P.O. Box 1234, Dept. IA 4 Cerritos, CA 90701

Charge my: ☐ Visa Card ☐ Master Charge ☐ American Express

Card No. ____________ Expiration Date ________ Signature ____________

Name (Print) ______________________ Title ________

Company ______________________

Address ______________________

City ____________ State ________ Zip ________