MORSE CODE
COMMUNICATION AID
FOR
THE HANDICAPPED

MORSE CODE

COMMUNICATION AID

FOR

THE HANDICAPPED

by

E. G. CALLWAY, B. ENG.

A Project

Submitted to the School of Graduate Studies

in Partial Fulfillment of the Requirements

for the Degree

Master of Engineering

McMaster University

April 1981

MASTER OF ENGINEERING (1981)
Department of Engineering Physics

McMASTER UNIVERSITY
HAMILTON, ONTARIO.

TITLE: Morse Code Communication Aid for the Handicapped

AUTHOR: E.G. Callway, B.Eng.

SUPERVISOR: Dr. Ralph Bloch

NO. OF PAGES 122, vi

ABSTRACT

A microprocessor driven display was built and programmed for the storage and reproduction of Bliss symbols. An explanation is offered for the success of the symbol language in teaching the handicapped.

The hardware was designed to be inexpensive enough for classroom use, but still deliver adequate flexibility and resolution. Due to the complexity and variety of the symbols a method of data compaction was developed to reduce the required storage space.

Initial tests are presented and suggestions are made for continuing the work.

ACKNOWLEDGEMENTS

The author would like to thank Dr. R. Bloch, Chedoke Hospital, for his advice and guidance. Appreciation is also expressed to Ted Bojanowski, Madeleine Harris, Ted Iler and Tony Wallace for their continual advice and support.

Special thanks are due to Dr. J. Russell for the interest he took in the project.

TABLE OF CONTENTS

Page

APPENDICES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.0 Communication Aids for the Handicapped

The inability to communicate with other people can make life difficult and unrewarding. Since the most common form of communication is speech, lack of speech can be seen as a serious handicap.

Communication aids enable a person with a speech, visual, auditory or motor disorder to communicate with other people. Those with handicaps in these areas are often downgraded, as a lack of words is unfairly equated with a lack of ideas.

The purpose of this project was to provide a non-vocal cerebral palsied person with a portable means of communication using Morse code. An appropriate review of cerebral palsy can be found in Day (4). Briefly, it is a neurological disorder, present at birth and nonincreasing, which causes widespread motor disabilities. Its effects vary from slight tremors and weakness to a complete lack of control over voluntary movements, including speech.

In this case, the subject was a young man confined to a wheelchair and possessing extremely limited speech, but of normal intelligence. He had some control over his feet and was able to turn the pages of a book and type slowly with them. For practical reasons of size and weight a typewriter could not be permanently attached to his wheelchair. An electronic keyboard with an eye-level display might have worked, but it would have still required a ruggedized QWERTY keyboard small enough to be unobtrusive, but large enough to be worked with the feet. For use outdoors either unit would be unsuitable as they could not be operated with a shoe.

1.1 The Morse Code as a Communication Aid

It was decided to try the Morse code as a method of communication. Unlike a typewriter, a single switch can be used to send the whole code, thus solving the problems of size and ruggedness. Morse code is fairly compact, with common letters such as E or T being a single dot or dash, while Z is much longer. This fortuitous arrangement is only valid for English.

The main disadvantage is that most people do not understand the code. It was felt that a simple and reliable decoding device could be built using microprocessor technology.

1.2 Previous Work and Basic Requirements

Schemes for the automatic decoding of Morse code have always been popular, but not necessarily cheap or effective.

Most of the research in this area is performed by amateur radio operators and various military organizations, each for their own purposes. Bell (2) presents the military view with its aim of increased surveillance with less manpower. For ham radio operators it becomes possible to communicate more enjoyably be removing some of the hard work. Recently there has been work on microprocessor based decoders, but they tend to require large volumes of code for initialization (1) or require fairly good code (12). The first requirement cannot be met due to the short conversations, and the second because of the nature of the sender. It was felt that a better algorithm could be developed for this use.

Unlike receiving, sending the code is fairly easy, and in fact the subject learned to send it within one week. By contrast, this author still cannot understand the code without the aid of the device developed here!

The items generated by this project were a footswitch interface to send the code, an adaptive software program to interpret it, and recommendations for the hardware required to produce a portable unit.

CHAPTER II

THE MORSE CODE

2.0 Basic Description

Morse code is an internationally recognized binary (on-off) coding scheme for the transmission of letters and numbers. Although it was originally designed to be sent and received by human operators, it can also be used with mechanical or electronic devices.

Each character consists of a unique sequence of one to six MARKS (on) separated by SPACES (off). The marks and spaces, or ELEMENTS, have several different lengths. Marks of time duration 1 are called DOTS, and those of lengths 3 are DASHES, SYMBOL SPACES of length 1 are used inside a letter, CHARACTER SPACES of length 3 separate characters, and WORD SPACES of duration 7 divide words.

The absolute timing of the elements does not matter as long as they maintain the 1:3:7 ratios. The dot is usually designated as the speed determining element. By not fixing the speed, operators can work from 5 to 60 words per minute (Guenther,9), often limited by skill of the receiver. Machine to machine transfers can proceed at higher rates.

Grappel and Hemenway (8) state that the reciprocal of the speed in words per minute corresponds roughly to the dot length in seconds for normal text. Thus, at an average speed of 10 words per minute, the dots and symbol spaces would be 0.1 second long.

Bell (2) was unable to find a standard value for the bandwidth of Morse code, but used 3 times the reciprocal of the dot length as a working value. This figure is important when designing analog input circuitry to reduce noise while retaining information. The bandwidth of machine code could be rigorously calculated, but hand sent code is too dependent on human factors for meaningful results.

2.1 Code Sample

To send a message, such as "I am hot", it is necessary to find the correct code for each letter, and then join the individual codes with the correct spaces. A table of the various character codes is given in Appendix A. There is no provision for upper or lower case in the code.

Individual codes:

```
        I  . .          a  . -          m  - -
        h  . . . .      o  - - -        t  -
```

Complete message:

```
 I         a     m           h          o      t    (text)
. .       . -   - -       . . . .     - - -    -    (code)
1 1       1 3   3 3       1 1 1 1     3 3 3    3    (mark times)
 1    7    1   3       7    1 1 1   3   1 1   3     (space times)
```

It can be seen that information is contained in both the marks and spaces.

2.2 Decoding

To decode a message it is first necessary to take each element and decide its relative length: 1, 3 or 7. For well sent, low noise code, the wide ratios (1:3 and 3:7) make this an easy task, even for a mediocre operator or simple machine.

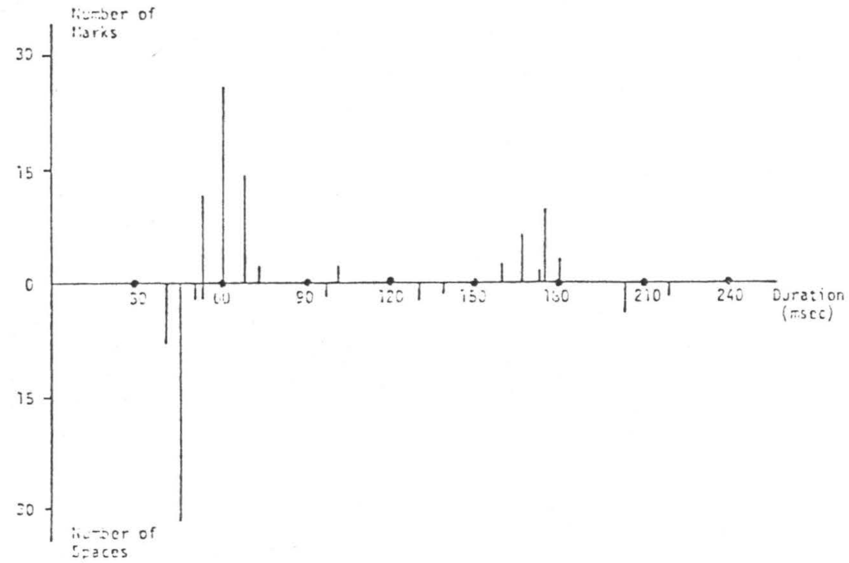
Because the spaces clearly define the beginning and end of letters and words, it merely remains to look up the individual letter codes and produce a copy of the message.

The above tasks can be implemented easily to produce a simple decoder at low cost. The Morse-A-Letter (11) is an example of such a device, available as a kit for $150.

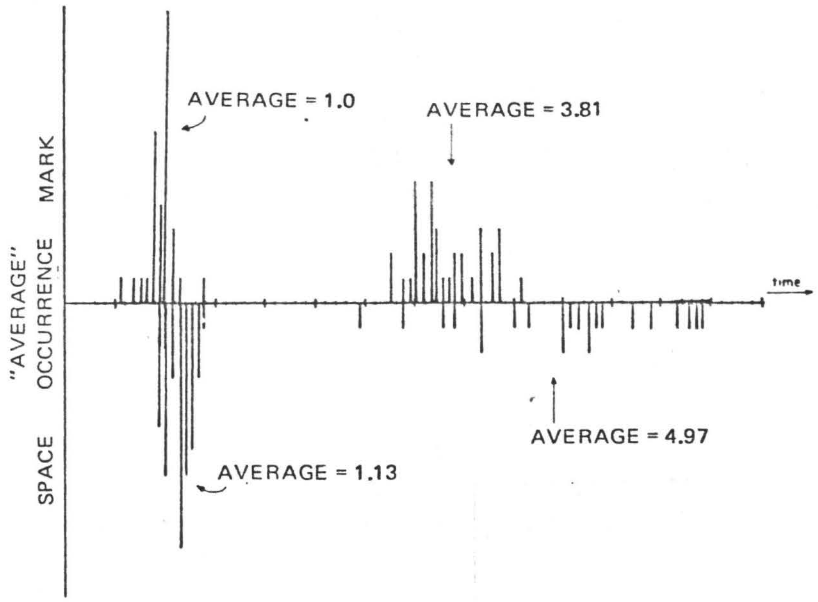
Unfortunately, Morse code is rarely received under ideal conditions. Human operators cannot exactly duplicate the 1:3:7 ratios, and during a long message both the ratios and overall speed will vary. Bell (2), Guenther (9), Freimar (5) and other detail some of these changes, including the variations which may occur inside a single character.

For most applications the signal has been received via a radio link, and is further corrupted by noise and fading.

For reliable decoding of Morse code, the machine must be adaptive in some way. It may only adjust its overall



a) Bell, Reference 2.



b) Hickey, Reference 10.

<u>Figure 1.</u> Histograms

speed, such as the Automatic Fist Follower (7), it may compensate for deviations in the 1:3:7 ratios (3), or it may even perform some textual analysis based on a rudimentary knowledge of English (10).

2.3 Code Characterization Displays

In order to analyze a Morse signal, most researchers have found it helpful to have a graphical representation of the main characteristics.

Hickey (10) and Bell (2) used histograms, shown in Figure 1. The horizontal axis is time, and the positive vertical axis shows the number of marks with that duration. The negative vertical axis displays the space information, inverted for clarity. In both cases, some of the important characteristics of Morse code are shown:

1. There are more dots than dashes, and more symbol spaces than character or word spaces.
2. The dot cluster is narrow with an obvious midpoint.
3. The dash cluster is wide, and the midpoint may not be clearly defined (flat top, rather than peak).
4. The symbol space cluster is as well-defined as the dot cluster, with nearly the same midpoint.
5. The character spaces are poorly grouped.
6. The word spaces are not grouped at all.

Point 1 is inherent in any Morse transmission of text, but 2 to 6 are peculiar to hand sent code. It can be seen that the 1:3:7 ratios are not exactly followed.

Guenther (9) used scatter distribution plots which are able to show the changes when one element follows another. For the example shown in Figure 2, it can be seen that dashes following word or character spaces (groups D and E) are longer than dashes after symbol spaces (group F). This particular type of information cannot be easily seen on the histogram displays. The frequency of occurence information is slightly obscured by the scatter plots as it changes from height to dot density.

For this project, the histogram method was chosen as it offers a good display of information with minimal hardware and software requirements. The output was first presented on a Teletype printer (Figure 3), but the program was changed to use a Tektronix graphics terminal (Figure 4). This gave a higher quality display and much faster output.

2.4 Decoder Functions

Any decoding device must perform several basic functions:

1. Separate the marks into dots and dashes.

This is the least difficult operation as the dots and dashes are usually well sent, preserving the 1:3 ratio or even making it larger (3).

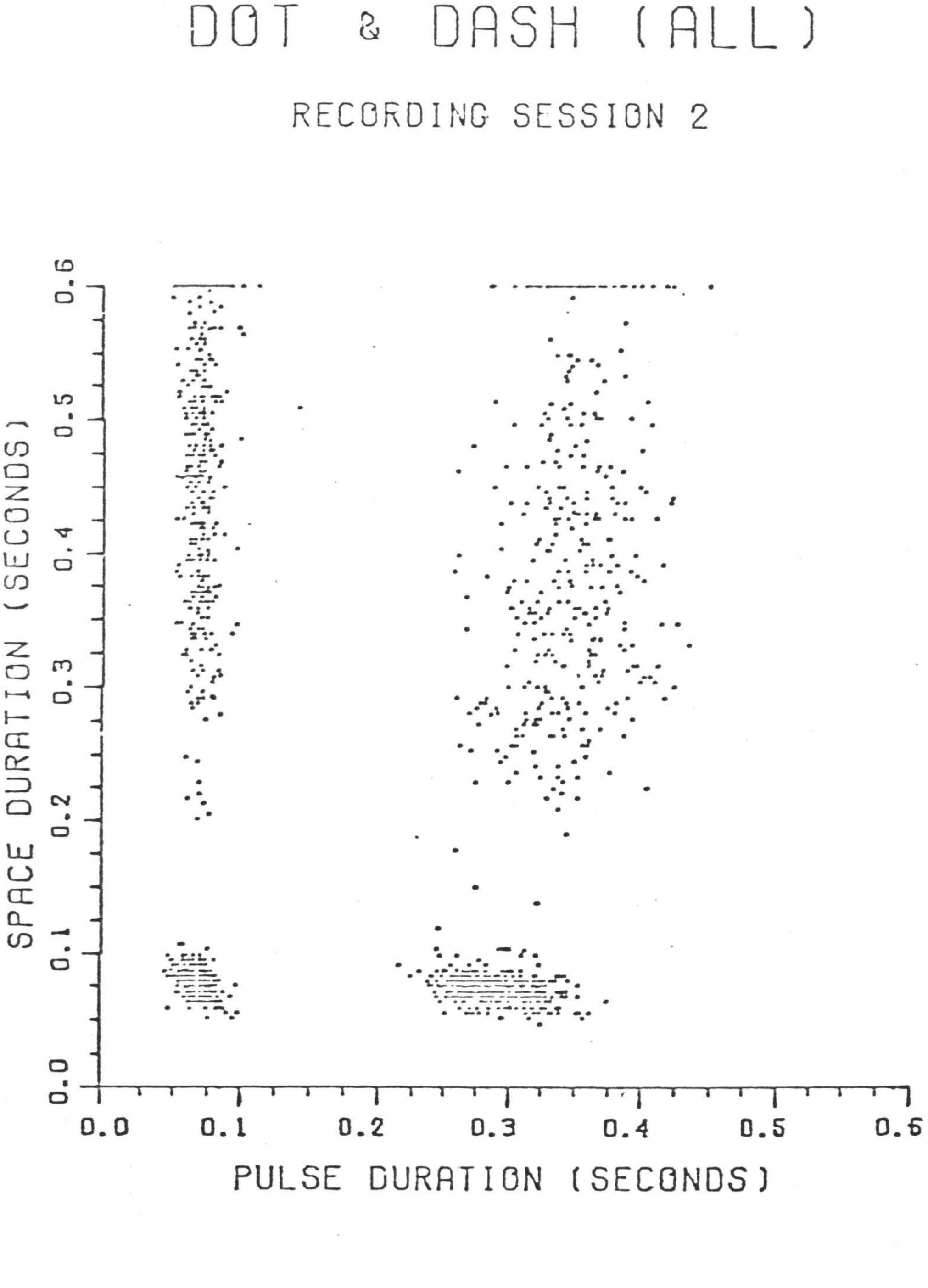


Figure 2. Scatter Plot. (Guenther, Ref. 9)

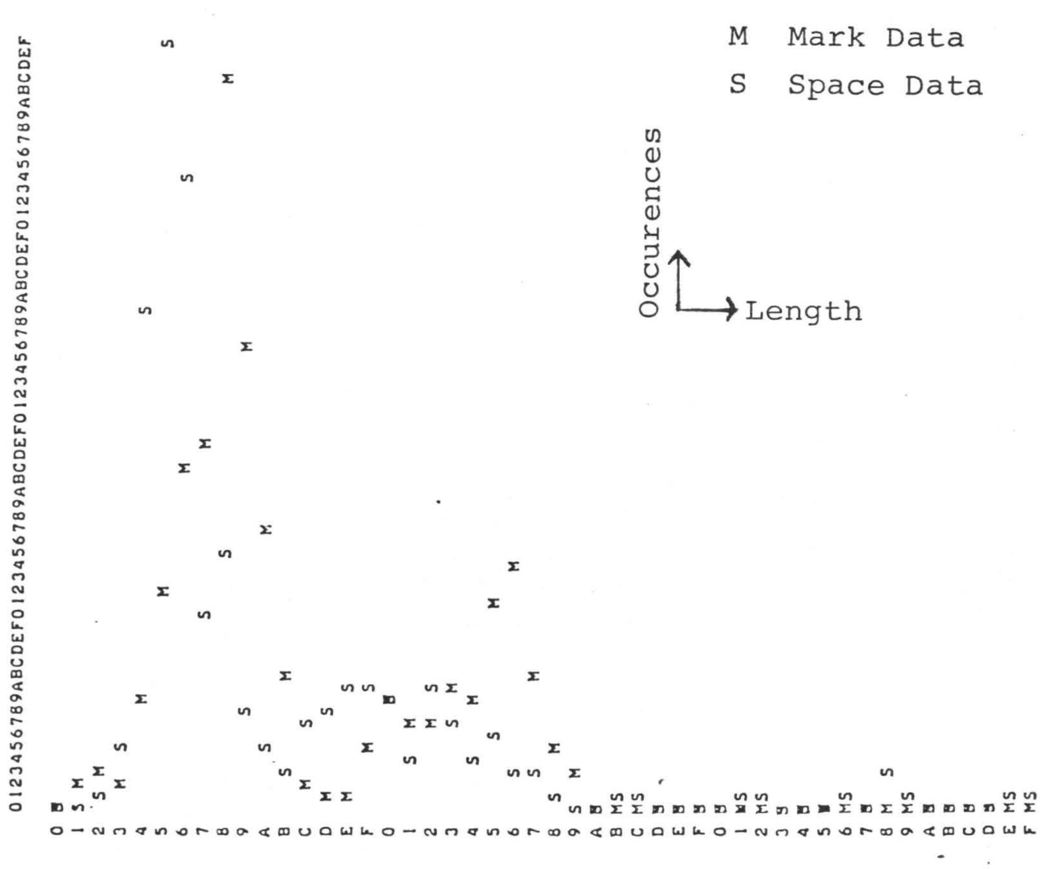


Figure 3. Teletype Histogram

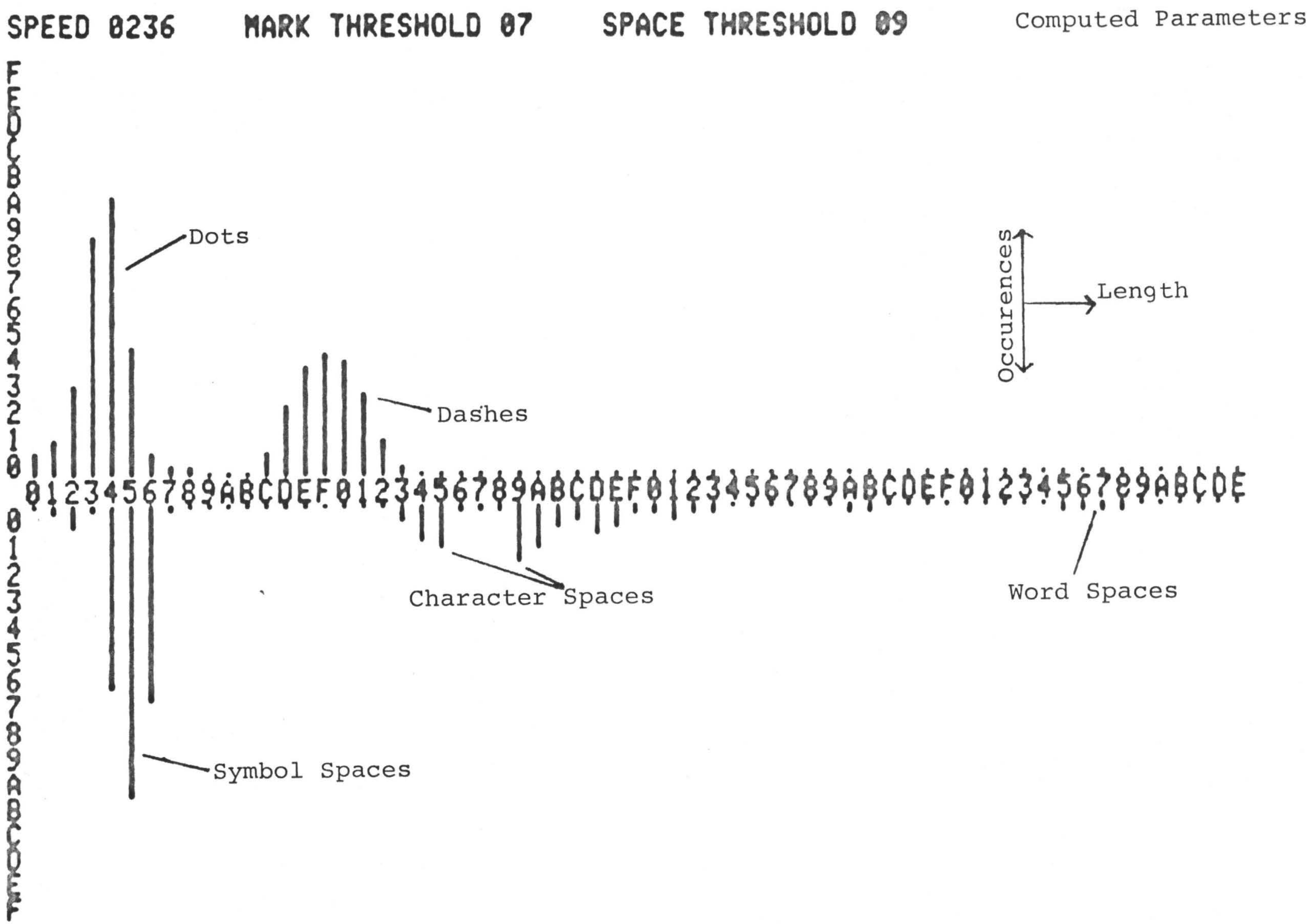


Fig. 4   Annotated Tektronix Histogram

2. Separate the spaces into symbol, character and word.

The spaces are never sent with as much care as the marks. Blair (3) and Guenther (9) agree that a more complicated decision process is needed for the spaces.

3. Based on the character spaces which have been found, divide the data stream into characters, decode and output them.

If the character spaces have been correctly found, and if the marks were properly identified, the final conversion from Morse to text is a simple operation using a look-up table. When the above processes produce an invalid character, there is a choice between printing an error message, and reassigning the various elements using a modified decision technique in an attempt to find a valid character. This correction process may be quite involved in the larger devices (11).

4. Based on the word spaces, output spaces between words.

Although word spaces are the most variable type of space, their identification is not critical, as a missing space between words rarely causes a loss of understanding.

2.5 General Decision Strategies

Hickey (10) noted that there are three basic approaches to the manual Morse problem:

1. Macro: You can accumulate statistical information on an

operator and use this data to make decoding decisions.

2. Micro: You can make your decisions on a mark to mark basis, sometimes called the "idiot dot" method.

3. Hybrid: You can compromise these two methods and come up with a hybrid algorithm.

Method 1 requires a memory large enough to contain all the timing information in a message, but should give the best results. Method 2 requires very little computer time or space, but is the least reliable. Successful devices, such as MAUDE (6), use the hybrid approach.

A common method is to keep an average speed parameter which varies slowly, and an average dot time which uses only the last few characters. The decision thresholds are calculated from empirical formulas using the dot average. The short "time constant" on the dot average allows tracking of fast speed changes, while the slower responding speed parameter inhibits locking on t.. noise (8).

2.6 Decision Problems and Approaches

1. Decoding Delays:

The methods which build up a large body statistical information about an operator require a large sample of code to be effective, thus delaying decoding. Guenther (9) tried to quicken the process by decoding initially with one set of rules, then switching to a better set when more code had been received.

For use as a conversational communication aid, the long initialization times of most methods are unacceptable. Since the code will be sent very slowly (possibly 1 w.p.m.) it is important to have each letter decoded as quickly as possible.

2. Non-Code Activation:

Due to the nature of the sender, there are periods when the code switch is involuntarily activated. If some of the simpler averaging techniques are used, after several non-code "words", the program will have locked on to that style, losing its memory of the sender's true characteristics. This is similar to the problems caused by noise and interference in the radio case, and is why "idiot dot" strategies fail.

Since a non-code sequence is obvious to an observer, (because of the sender's physical involvement) no attempt was made to block decoding during one. It would require a complex program to have the computer recognize invalid input. The decoding algorithm was designed so that inputs which are not similar to previously received code do not affect the decoding thresholds unless prolonged.

3. Space Problems:

Problems arise when there is poor distinction between the symbol and character spaces. If a long group of marks is received with no obvious character space in the string, a good receiver can separate the string into characters using his knowledge of valid letter codes, combined with his expectations based on context.

Machines find this difficult unless they contain rules and vocabulary from the language being decoded. An easier method is to mix timing and simple language rules such as in the MAUDE decoder. Rules such as "The longest of 6 successive spaces is almost always a character space" allowed the machine to divide a string of poor code into letters. If the division resulted in an invalid character, a modified set of rules was tried. (6)

The bad grouping of the various space types not only causes incorrect decoding (due purely to the bad sending), but also makes it difficult to calculate best values for the space thresholds, compounding the errors. In this program the threshold setting process does disregard poorly grouped information as mentioned above, but in addition the initially calculated symbol/character threshold is averaged with the dot/dash threshold, which is more reliable. For perfectly sent code these two thresholds should be equal.

4. Correction Routines:

McElwain and Evens (11) reported good success with a "degarbling" program to correct machine received code. It was given a vocabulary list of every word which might be used in the message, along with the received, partially decoded text, and some timing information. Based on this it could correct many of the errors in the received text. This would be useful in a military situation where the context is unknown but the vocabulary prescribed. It allows large volumes of readable text to be produced with little operator attention.

In ordinary conversation the vocabulary is large, but the context is known - such as replies to questions. It was found in preliminary tests that a reply never had to be completely spelled out, as the listener could correctly guess the full statement after a few letters or words. For this reason no error correction routines were included in this design. This works well in a conversational environment, but might be inadequate for other uses.

5. Effects of Incorrect Decisions:

Most of the schemes examined use variations of the following method. The element being processed (on line or from memory) is examined and a decision made as to its nature (dot/dash, symbol/character/word). This decision is used as part of the character currently being assembled for decoding. If the element was classified incorrectly, the current character will be incorrect. Error correction schemes may help if an invalid character was produced.

One incorrect character is not a problem, but each element is then used to update a running average, as in:

DOT AVG. = DOT AVG. + NEW DOT/8 - DOT AVG./8

DASH AVG. = DASH AVG. + NEW DASH/8 - DASH AVG./8

(This keeps running averages of the typical dot and dash, with each new element given a weight of 0.125.)

The various averages are used to calculate the thresholds, as in:

MARK AVG. = DOT AVG./4 + DASH AVG./2

(For perfectly weighted 1:3 code, this produces a threshold at 1.75, an empirically better value than value 2, which might have been expected.)(after 1)

Unfortunately, any incorrectly classified element is averaged into the wrong place.

The effect of this is that a period of bad code or noise is not only incorrectly decoded (which may be unavoidable), but is also used to calculate incorrect thresholds which prolong decoding errors into periods of good code.

Most of the element classification errors occur when an element is received with a timing value near a threshold value. These elements are poorly grouped with respect to the average of their intended value. If these elements are included in the average calculations, even if they are correctly classified, poor threshold values must result.

Elements near present threshold values cannot however be explicitly and permanently excluded from calculations as they may be important if the operator has changed speed.

The decoding algorithm outlined in the next chapter tries to use only well-grouped data, ignoring data which may represent noise or odd code.

CHAPTER III

THE DECODING PROGRAM

Some of the main features of this program are:

3.0 Timing

A software timer is used instead of external counters. This gives maximum adaptability and minimizes the hardware requirements. At the slow code speeds anticipated, the non-timing portions of the program do not significantly distort the timing, so the delays through various paths were not equalized, but ignored. A single speed parameter is used, with an estimated useful range of 300 to 1.

All timing in the program is in the arbitrary, dynamically variable units determined by the speed parameter.

3.1 Input

The code input is sampled under program control, instead of using an interrupt. It was felt that an interrupt based system was too susceptible to noise, as it must respond to every input change, however short. When fully adapted to a code style, an average length dot is sampled 8 times, a dash 24 times. While adapting, the program will decode correctly if a dot is sampled from 2 to 20 times.

The program will lock on to code over a much wider range.

3.2 Histograms

As each mark or space finishes, its length (in arbitrary timing parameter units) is stored in a histogram type table. There are separate tables for marks and spaces. The tables allow for a length from 1 to 64 timing units, with a maximum count in each entry bin of 64 occurences. When any bin in either the mark or space table reaches 64 counts, the data in both tables is divided by 2. This prevents software overflows and allows new data to slowly take over from the old. The histograms are smoothed with a moving window and used for threshold calculations.

The smoothed data can be displayed on a graphics terminal as an actual histogram. This is useful for checking the overall quality of the code and the validity of the threshold setting algorithm.

3.3 Morse/ASCII Conversion

As each element comes in, it is shifted into a character holding register, using "0" for dots and "1" for dashes. When a character space is found, the data in the character holding register is used directly as a memory address pointing to the equivalent ASCII in a look-up table. This method is simple and fast while requiring a minimum of space for the table.

Two 6 mark characters produce ambiguous addresses and are dealt with separately.

If it is desired to put out more than one character per Morse input, such as a "CR/LF", then a special case subroutine must be written.

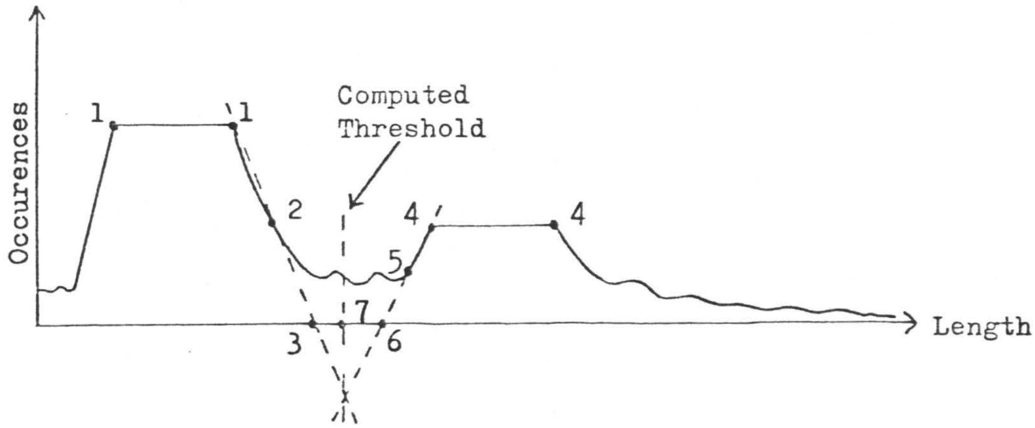
3.4. Slow Adaptability

After the thresholding process has decided dot or dash each mark time is compared to 8 or 24 respectively. If the time was short, the overall speed is increased by a small increment, and vice versa. This action slowly forces the program into the correct speed range. As it works slowly, noise or bad code will not really affect the speed unless prolonged for dozens of marks, an unlikely event. Due to its poor quality, the space data is not used.

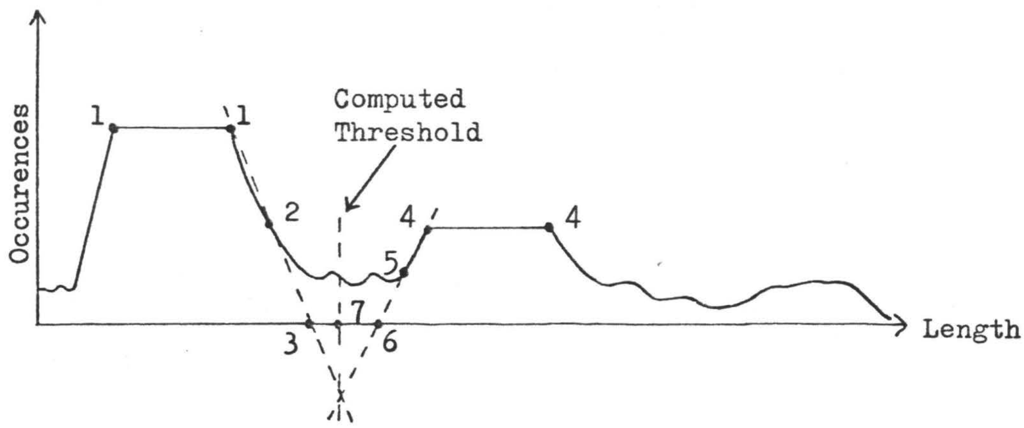
Initially this was the only adaptation mechanism in the program, but it could not be made fast-responding without the program becoming unstable during bad code. It also required perfect 1:3 ratio code, which is rarely sent.

3.5 Threshold Setting

Figure 4 shows a typical histogram and 5 shows an idealized version. If point 7 can be reliably found, then the critical dot/dash and element/character



a) Marks



b) Spaces

Figure 5. Threshold Histograms.

decisions can be properly made.

Blair (3) used histogram data, by assuming an initial threshold value, then calculating an empirical "goodness of separation" based on various statistical parameters. The threshold estimate was moved until the "goodness of separation" was maximized. This method requires a lot of computation and would not be suitable for real time application, but the basic idea is excellent.

The stylized histogram in Figure 5 provides the basis for the method chosen. It was found that code from different sources always had similarly shaped histograms. The most important similarities are that the dot peak is consistently higher than the dash peak, and the symbol space peak higher than the character space peak. The work space peak is not well-defined and often cannot be found. The "noise level" between peaks is lower than the peaks, and consists of elements which were poorly timed. The peaks are shown with flat tops, but this is not essential.

To find the threshold (dot/dash or symbol/character):

1. Find point 1:

An entire histogram is searched for the largest value, point 1. If two points have this value (1 and 1'), then the rightmost one is taken because it should be closer to the threshold. This largest peak is always the dot or symbol space peak. The position and value of point 1 are noted.

2. Find point 2:

The histogram is searched from point 1 to the right for the data point which is 1/2 or less than the value at point 1. If this point does not occur before the dash peak the code is of very bad quality and cannot be decoded. Only its position is noted, as its value was used to find it.

3. Find point 3:

The first estimate of the threshold is found by calculating point 3, which is the position as far to the right of point 2 as point 2 is to the right of point 1. Its value is ignored, because it contains the bad data that this method was designed to avoid.

4. Find point 4:

The histogram is now searched for the largest data value from point 3 to the end. This will be the dash or character peak. If two bins contain this value the leftmost one is chosen as this will be closer to the threshold. The position and value are noted.

5. Find point 5:

The histogram is searched from point 4 to the left for the first bin whose value is 1/2 or less than that of point 4. If the code is very bad the noise floor could be higher than this 1/2 value, which would extend the search to the left side of the first peak. To avoid this, the search will stop at point 3, even if the desired value was not found. Code that bad would be unreadable anyway, but a good threshold value must still be chosen to aid the slow adaptation. Only the position of this point is used.

6. Find point 6:

The second estimate of the threshold is found by calculating point 6, which is as far to the left of point 5 as point 5 is to the left of point 4. Just the position is taken as the value contains questionable data.

7. Find point 7:

Point 7 is the arithmetic average of points 3 and 6. This is the estimate of the threshold. If either line 1-2-3 or 4-5-6 has a shallow slope it is possible for point 3 to be to the right of point 6. This does not affect the choice of point 7, as it just means that 1-2-3 crosses 4-5-6 above rather than below the horizontal axis.

8.8. Space Correction:

The above seven operations are performed separately for the mark and space data, so that the only tie between the thresholds should be the overall program speed. This allows for different distributions of the marks and spaces. It also is useful for very slow code (less than 1 w.p.m.) so that long spaces can be used without long marks.

It was found however that the space data is never as well-grouped as the mark data. This becomes a problem when small code samples are obtained, as it may take a long time to build up the space peaks properly. For this reason the space threshold estimate obtained from steps 1 to 7 is averaged with the value for the mark threshold to obtain the final space value. It was found that this helped decoding in almost all cases.

9. Character/Word Threshold:

The space between words in regular Morse code is 7 dot times long. These word spaces should appear as a third peak in the space histogram, but rarely do, so a continuation of the above methods cannot be used to find it.

The character/word threshold is set at 3.5 times the symbol/character threshold, or 7 dot times (assuming a symbol/character value of 2 dot times). Normal code would give a value between 3 and 7, but it was found that for this application the character spaces were as long as ordinary word spaces. This operation is not critical for conversational use.

The threshold setting method has been shown to find a good threshold value. No math other than integer addition, subtraction, and division by shifting is used, giving a high speed. It can be seen that a simple curve fit is actually done to find the horizontal position of the intersection of lines 1-2-3 and 4-5-6, which should be at the low point between the two distributions. Only very basic assumptions are made about the shape of the histograms, without fixing the time ratios at 1:3 or the valley at a certain distance between peaks. Any time bins which do not have counts to a value of at least 50% of the closest peak are automatically excluded from the calculations, as are all bins on the far sides of the peaks which also do not contain information about the threshold position.

The most important objective has been achieved, and that is any elements which were incorrectly classified when first received are not averaged into the wrong place to perpetuate the initial error. Instead they are put into the histogram where they belong in time-relation to previously received elements. If they were part of a legitimate speed change, new peaks will form in the correct place to give a correct threshold, and if they were errors of low occurence they will be completely ignored.

If reasonable quality code is received, a useable threshold is usually found after the receipt of less than a dozen marks, giving the required fast initialization time.

3.6 Out of Range Correction

If the program is running far too quickly, then the dashes will time out to the maximum of 64, rather than the ideal 24. If a mark of length 64 is detected, the timing constant is doubled (to slow the program and halve the counts) and the program is restarted with clear histograms.

This allows one to start without having to make an initial guess at the code speed. The program will restart on each mark until the timing constant is within range. This mechanism can be fooled by holding down a key to produce a long mark, so an automatic keyer should be used which cannot produce marks longer than those required. Spaces are not checked in this manner as a long space could be just a pause in sending, not a change of speed.

If the program is far too slow, a mark may only be sampled once. There is no explicit mechanism to deal with this case, as a typical key bounce would also be sampled once and a speed halving mechanism (as above) would cause the program to lock onto that noise. The normal threshold setting process can deal with this condition, for if all the marks go into the 1 bin rather than being spread around 8 and 24, the histogram fills up and empties quickly. This lets the slow adaptation and threshold mechanisms function effectively by quickly removing data stored at an earlier speed. This is a slower correction than for the opposite case above, but it is stable unless the number of noise marks greatly exceeds the number of true marks. Code of such bad quality is not expected in this application.

Flowcharts are given in Appendix B and the assembled code is in Appendix D.

CHAPTER IV

HARDWARE - AIMS AND DEVELOPMENT

4.0 Hardware Requirements

The hardware falls into three main categories:

1. Patient Interface - physical movement to audible code
2. Digital Interface - audio signal to digital input
3. Processor and Display - digital input to visible letters

The ideal system would consist of a single switch connected directly to an input port on a one chip microprocessor. An output port would drive a speaker with clean code for audio feedback, while another drove a display with the reconstructed characters. Very little other hardware would be required, as all the timing and switch debouncing would be performed in software. The computer and the display would be lightweight and low powered. The system used for development was more complex, but at all stages decisions were made with the above criteria in mind.

4.1 Patient Interface

In any manual Morse system a key is required to translate the operator's movements into a string of dots and dashes, providing an electrical signal for transmission and an audible one for the operator. The audio feedback may come from a directly connected buzzer, monitoring of the transmitted signal, or the mechanical action of the key. Keys vary from a simple switch to dual switches with mechanisms for automatically timing perfect dots, dashes and spaces.

Normally code is sent with the hand, but in this case the subject only had reliable control over his right foot, so this point was chosen for the interface. During the largest muscle spasms it was unuseable, but good action was available at most times.

At an early point in the project, it was decided not to use a direct electrical connection between the subject and the processor. In its place the audio signal from the keyer was picked up by a microphone and conditioned for use by the computer. This approach allowed all test sessions to be recorded on a cassette recorder in any location suitable to the subject. Whether the code was "live", taped in the subject's home, or from a shortwave radio, it could all be processed the same way and stored easily for reevaluation. It also solved the problem of electrically isolating the subject from the equipment, which was line-powered except for the keyer.

One final benefit was that verbal comments on the progress of the session were recorded at the appropriate times and were thus more valuable than written notes. This was especially useful in analyzing the earlier sessions, as the character sent could rarely be recognized but was often framed by comments which described the true intent and the mistake.

The first key to be tried was a "Rancho"[1] footswitch switch connected directly to a 12-volt battery and a Mallory Sonalert (Figure 6). The Sonalert was chosen because of its low power consumption, high efficiency and sine wave audio output. The last quality was found to be useful in triggering the phase locked loop detector described below. Ordinary buzzer sources produced complex audio waveforms which were hard to detect against background noise.

The switch could be placed in a loose-fitting shoe or sandal and was cosmetically the most pleasing, but it had a large amount of contact bounce. This was not a problem in gait studies due to the higher pressure available for contact closure, but the effect was intolerable for machine read Morse code.

It became apparent that the subject was making a valiant effort to send correct code, but he could not time out the dots and dashes properly with the single switch. It was decided that a dual action switch was needed, with

---

[1]Developed at Rancho Los Amigos for gait studies

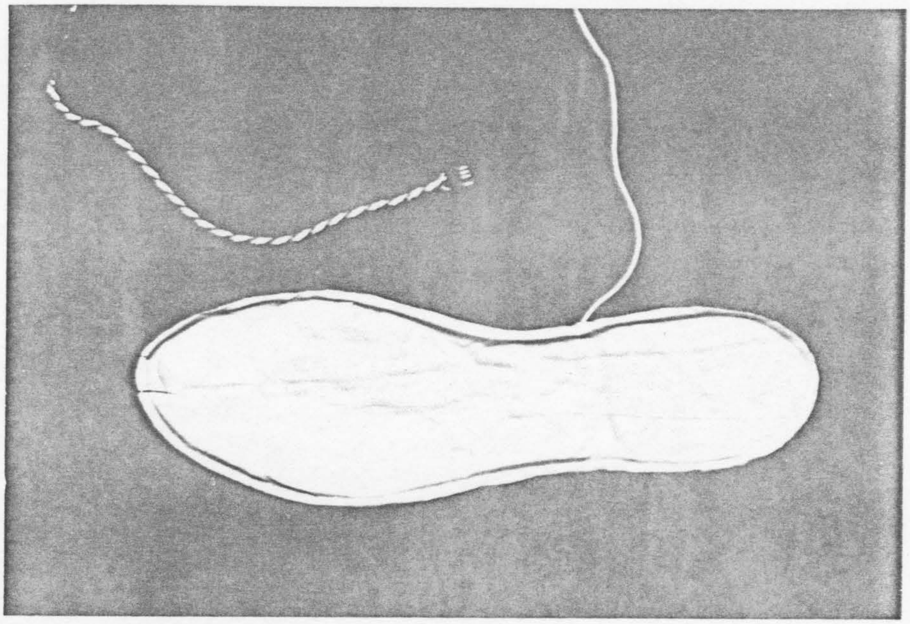


Figure 6. Rancho Footswitch

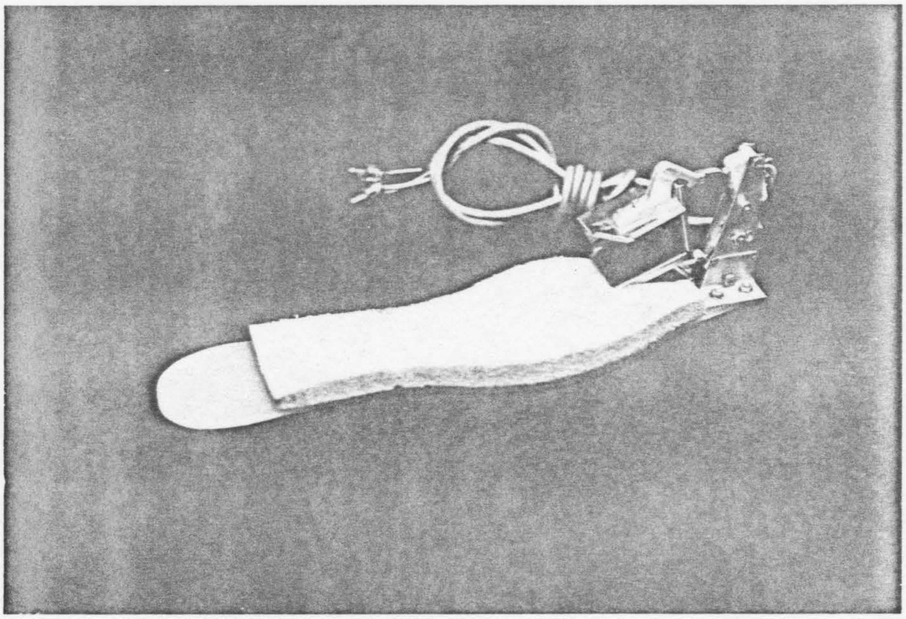


Figure 7. Metal Footswitch.

one position for dots and another for dashes. The author designed and built the footswitch in Figure 7. The metal loop enclosed the big toe of the right foot and pivoted in a similar way. Two microswitches sensed motion up or down from the rest position to trigger a dash or dot respectively. This orientation was chosen as the subject could press down more easily and more dots are used than dashes. The restoring force from each direction could be varied independently by changing the microswitches.

To complement this, a keyer was designed by Tony Wallace[2] with two separate inputs which would result in an audio tone of dot or dash length when activated.

Further trials were conducted with this equipment and new problems appeared. The footswitch required the foot to remain motionless while the toe pivoted up and down. The subject however found it easier to curl his toes down and push his foot up and down, which did not activate the switch properly. The switch also had to be flexible like the Rancho switch or else much stronger, as it was easily damaged.

The keyer had no automatic space delays, so although perfect length marks were being sent, a dot and dash could be sent simultaneously. Holding a switch down produced one mark, but another often appeared on release due to contact bounce. The square wave oscillator and small speaker pro-

---

[2]Electronic Technologist, Chedoke Rehabilitation Centre

duced an excellent signal for the ear, but it was not recognized over background noise by the equipment. The length of the tones, and hence the overall speed, could not be easily varied.

Another footswitch was designed and constructed. (Figure 8) Two microswitches were mounted into the wooden frame and activated by pressing the clear plastic cover plates. This design was more rugged and easier to activate.

4.2 Automatic Keyer

The author designed and built a fully automatic keyer which can send perfect dots or dashes, but with the addition of a perfect symbol space between them. If the footswitch is held down the keyer will produce a string of dots and spaces. If it is then suddenly pushed up, the dot in progress is finished, the symbol space timed out, and the switches sampled. If the dash switch is still activated, a dash is begun and the switch can then be released. The dashes are also auto-repeating if required.

There is a semi-automatic mode in which a switch has to be released and reactivated before a mark is repeated, although the minimum space timing is still produced.

In the manual mode either switch activates the tone for as long as it is closed with no automatic timing.

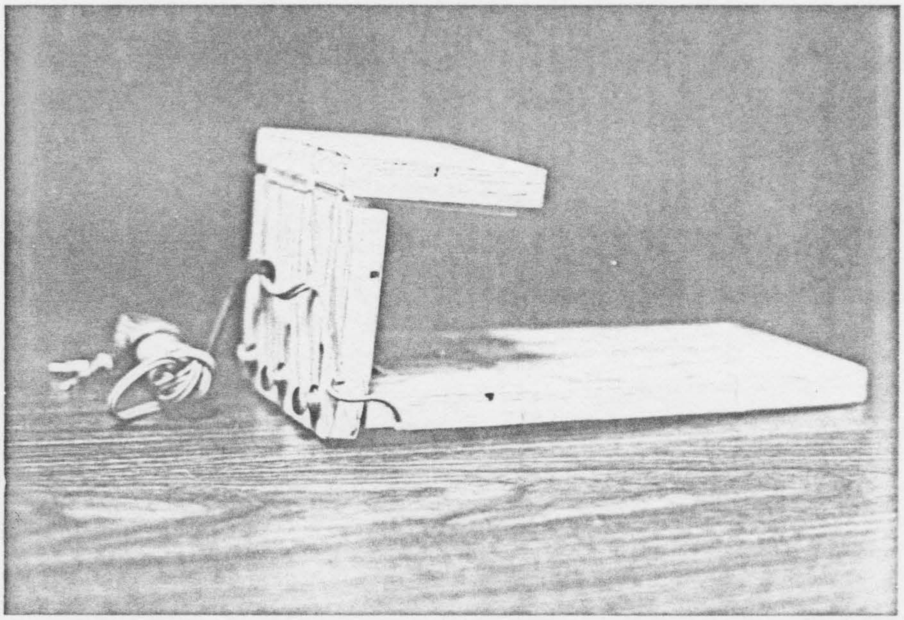


Figure 9. Wooden Footswitch.

Since the design is fully synchronous without using monostables, the speed can be varied over a wide range with all timings (dot, dash, space) in the correct ratio. The Sonalert was used again for the advantages outlined above. The unit was built with CMOS technology and has a quiescent power drain of microwatts. For this reason and to simplify operation no power switch was included. The battery can be from 4 to 9 volts with no change in operation.

4.3 Detailed Keyer Operation

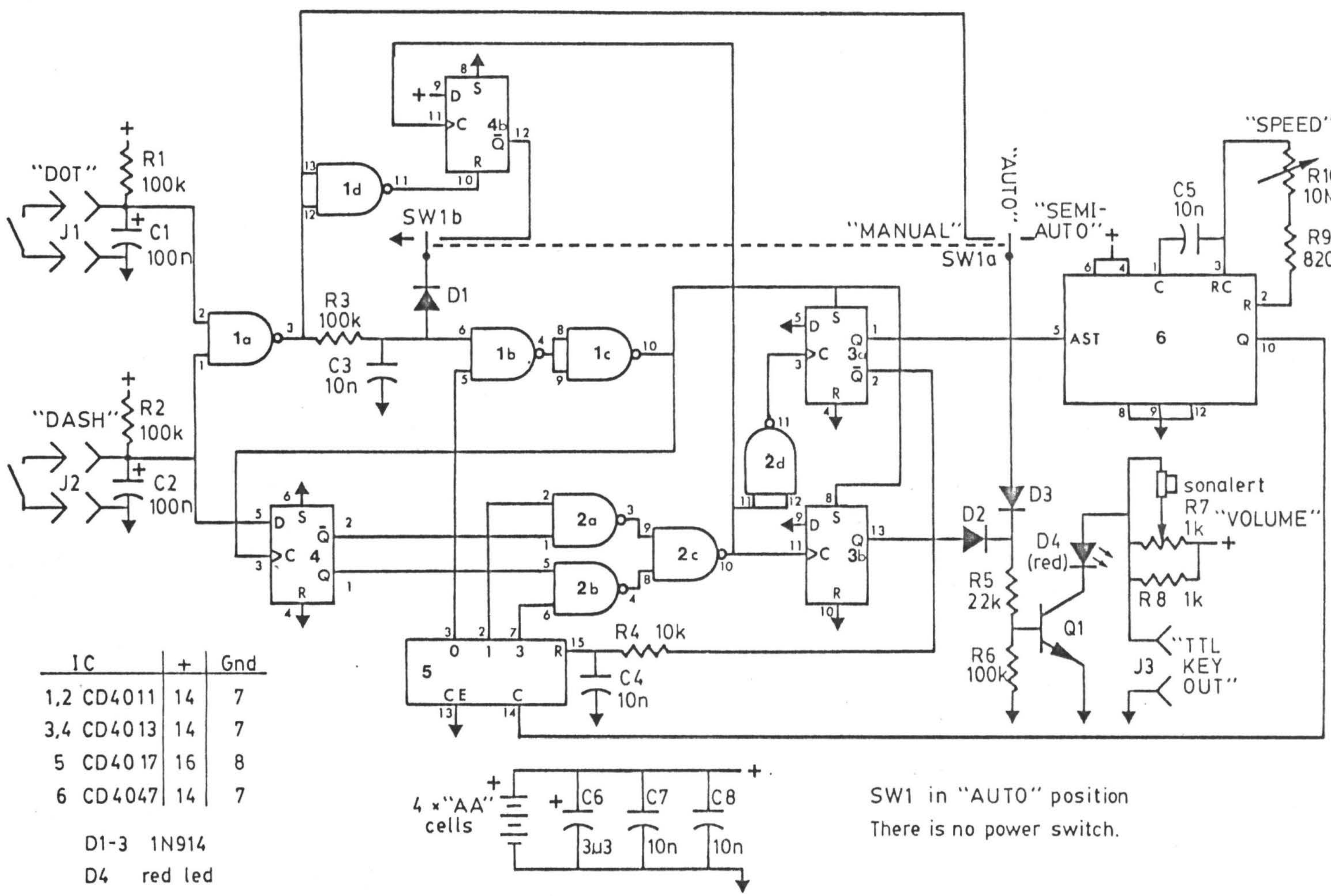
In the manual mode either switch can directly activate the output through ICIa,D3 and Q1. The trigger signal to the timing circuits is shorted to ground through D1 and SW1b.

In the automatic mode IC6 is the timing oscillator which sets the code rate. IC5 is a decade counter with 10 active high decoded outputs, clocked by IC6. If no switches have been closed:

1. IC5 is in the "0" state, closing the trigger sampler formed by IC1b and IC1c.
2. IC3a is off turning off IC6, the clock.
3. IC3b is off, turning off the output.

When a switch is closed, the trigger signal gated through IC1b and IC1c:

1. Sets IC3a to turn on IC6 and start the clock.



IC	+	Gnd
1,2 CD4011	14	7
3,4 CD4013	14	7
5 CD4017	16	8
6 CD4047	14	7

D1-3 1N914

D4 red led

Q1 2N2222

SW1 in "AUTO" position

There is no power switch.

Figure 9. Keyer Schematic

2. Sets IC3b to turn on the output through D2 and Q1.

3. Clocks IC4a to latch the input - 1 for dash, 0 for dot. Because the input is latched, a change in the switches will not change the timing.

IC2a, b and c select the count from IC5 which will stop the output, based on the switch latched in IC4a. A dot is stopped by the "1" count, a dash is stopped by the "3" count, giving the correct timing.

Using a dot as an example:

1. The output is on during the "0" count after triggering This is one clock cycle long.
2. The rising edge of the "1" count, gated by IC2a, b and c, clocks IC3b to the zero state, turning off the output and ending the dot.
3. During the "1" count, the output is off and the input is locked out. This gives the required symbol space.
4. One clock cycle later, the falling edge of the "1" count clocks IC3a to the zero state, which turns off the clock and resets IC5 to the "0" state.
5. As IC5 is reset to "0" the switch sampler of IC1a and b opens again to check the switches, ending the cycle.

The dash sequence is identical, except the "3" count is used instead of the "1" count.

In the semi-automatic mode, the signal which ends the mark also clocks IC4b to the "1" state. The trigger signal is then grounded out through the $\overline{Q}$ output and D1. When both keys have been released, IC4b is reset through IC1d. The effect of this is that a switch must have been released after the completion of the mark (i.e. during the symbol space or later) and then reclosed to send another mark.

4.4 Footswitch/Keyer Results

Sequences such as "dot dash dot" (R) became "down up down" with little regard for timing. The only requirement was that a switch had to be held until the mark began, and the switch for the next mark had to be activated (or the last one released) by the time the previous symbol space was finished. Due to the very slow speeds available and the fact that the switches were locked out when not being sampled, good "noise immunity" from involuntary movements was achieved.

This equipment was tried and the only improvements needed were a reorientation and height adjustment of the upper switch, but its overall performance was finally good enough to test the software.

A final footswitch was built incorporating the needed adjustments((Figure 10) and it performed satisfactorily.

4.5 Audio Input Board

The basis of this circuit, shown in Figure 11, is IC3, a phase locked loop tone detector. This approach was chosen

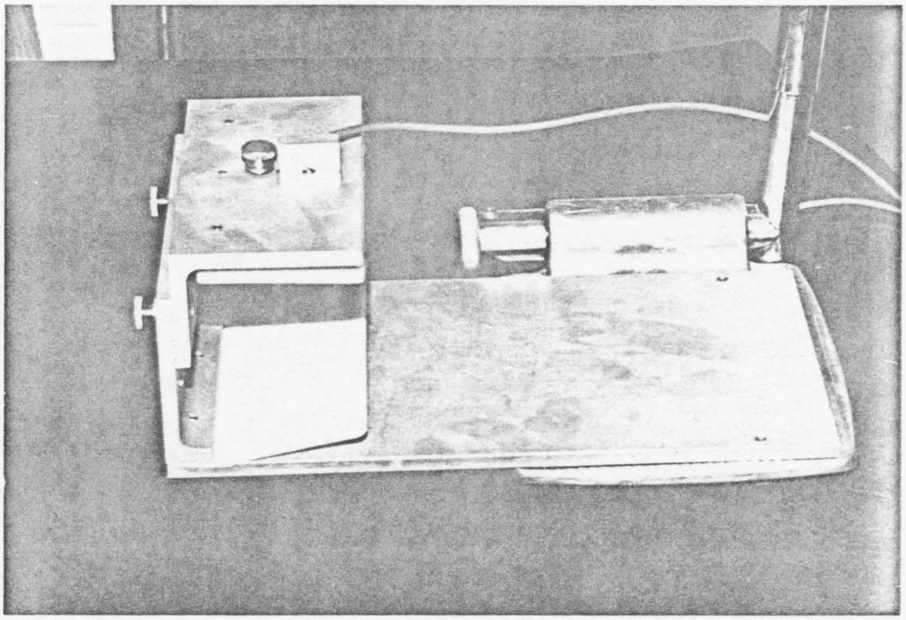


Figure 10. Final Footswitch (on wheelchair mounting).

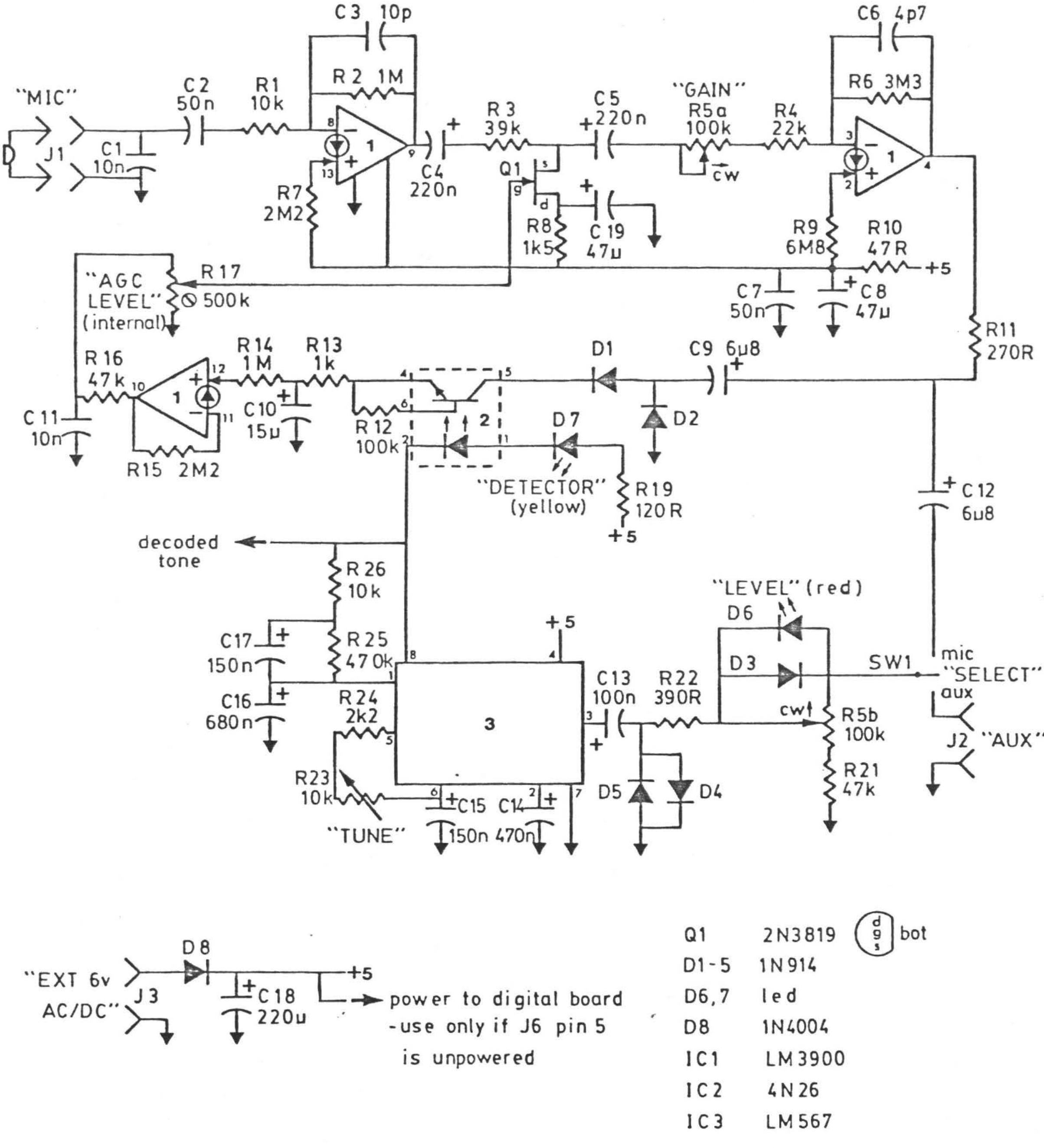


Figure 11. Audio Input Schematic

over simple amplitude detection as it gives excellent rejection of background noise. As outlined in reference 13, there are design tradeoffs involving detection bandwidth, speed of response, and noise immunity.

Switch SW1 selects a high gain microphone input or a direct input for tape playback. Gain control R5b and D3, D6 form a nonlinear input network to aid noise rejection. At low gain levels (large inputs) the signal must overcome the forward voltage drop on the diodes, so low level noise is partially inhibited. D6 was also intended to function as a level indicator, but the signal required to activate it was too large.

R22 and D4-5 clip any input to 1 volt peak to peak. If larger signals are allowed IC3 tends to detect harmonics of low frequency signals. C15 and R23-24 determine the centre detection frequency, allowing tuning from 700 Hz to 3000 Hz. C14 affects detection bandwidth and loop response time, while C16 is part of a filter on the output switch. These two capacitors were varied considerably from design values to optimize switching times and dropout rejection. R25-26 add some hysterisis to the output, and C17 increases this feedback during switching transitions.

IC1-2 are an AGC (automatic gain control) amplifier for microphone inputs. IC1a provides a gain of 100, with 3db rolloffs at 300 Hz (R1, C2) and 16 KHz (R2, C3). Q1 and

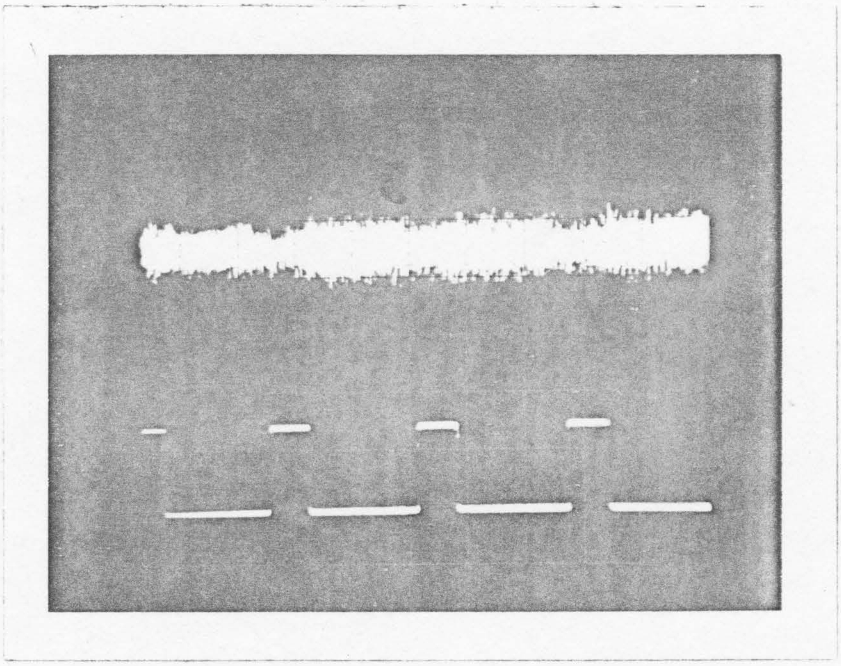
R3 are the variable gain attenuator necessary for AGC action. IC1b is the amplifier inside the AGC loop, and it also lowers the upper 3db point to 10 KHz. The external gain control varies the gain of this section from 27 to 150, exclusive of the attenuator. It may seem odd to have this control inside the loop, but it limits the maximum gain for low level signals when low gain is selected. Combined with the nonlinear network of R5b, D3, D6, at low gain levels low level signals are not well amplified by IC1b and are further rejected at the nonlinear network. By reducing these unwanted signals IC3 performs more reliably. High gain is available when required, but it should only be used in quiet surroundings.

D1-2 and C9-10 form a voltage doubler for AGC detection. Attack time is approximately 20ms (R11+R13, C10) and decay time 15s (R14, C10). IC1c adds a dc gain of 2 to the AGC control signal.

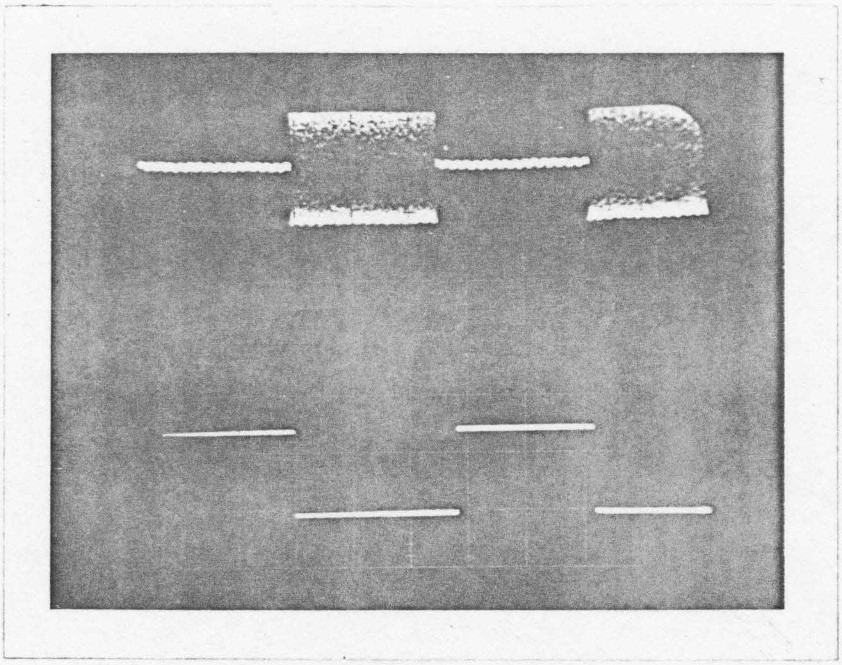
IC2 gates the AGC signal to C10 so that it is only increased to lower the gain when a tone has been detected by IC3. This stops high room noise from lowering the gain of the system and relates it more closely to the level of the desired signal.

Tests were done in a noisy room (fan noise) using the Sonalert Keyer and a microphone pickup several feet away. As Figure 12 shows, good digital signals were produced for

Figure 12. Phase Locked Loop Tests.



a) High ambient noise (short space, long mark)



b) Low ambient noise (long space, long mark) both low and high noise inputs.

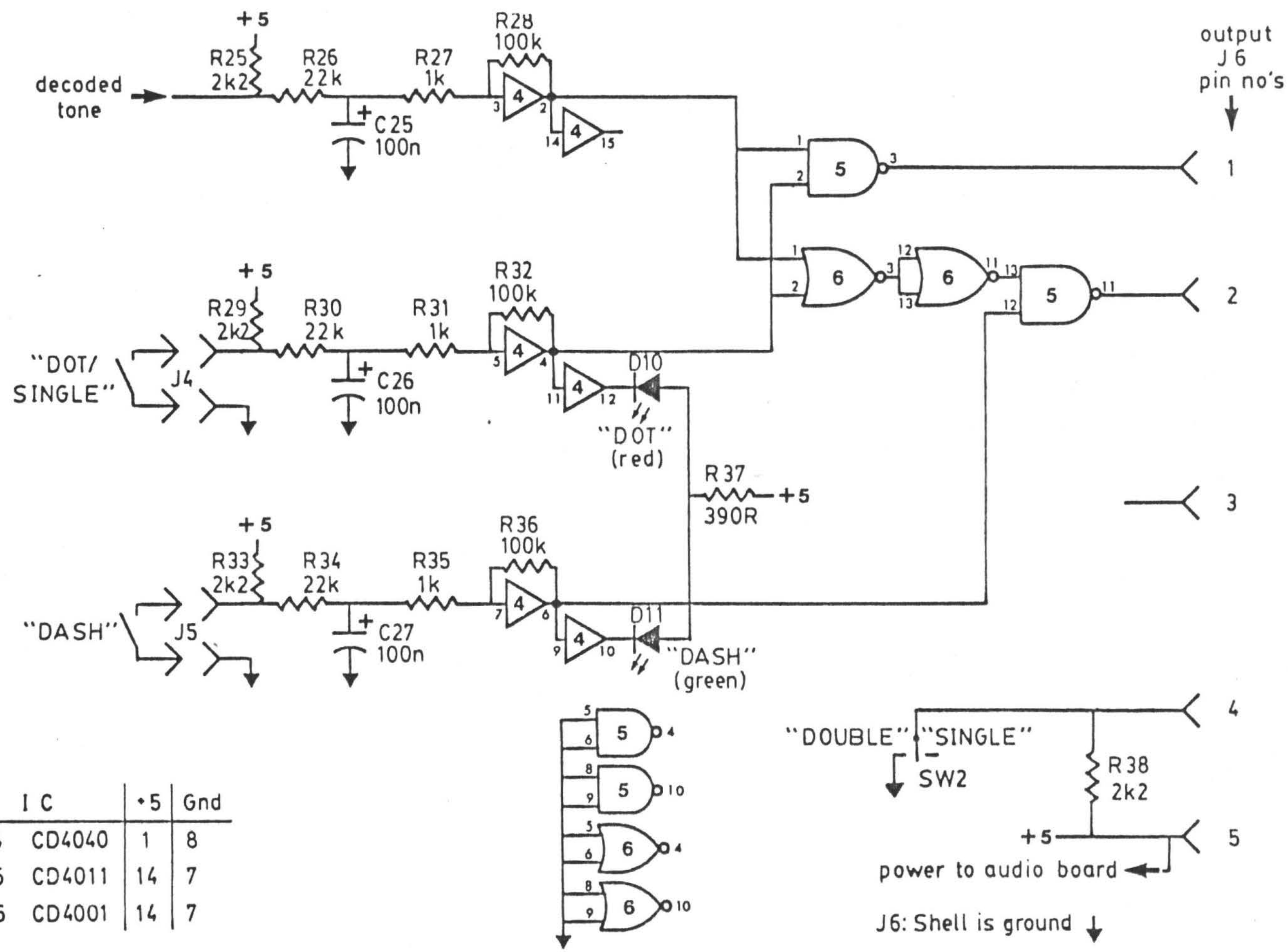
For tracking of noisier or faster code, one could add a bandpass filter in front of the microphone amplifier with its tuning coupled to the phase locked loop. This would greatly increase noise rejection and allow other time constants to be relaxed for better tracking of fast code.

4.6 Digital Input Board

This is a simple circuit (Figure 13) to condition the detected tone from the audio board and allow the direct use of single or double keys.

The three input circuits are identical and require a switch closure to ground for a mark. R25 is a pullup resistor and R26, C25 form a low pass filter with a cutoff near 70 Hz to reduce noise from IC3 or external keys. R27-28 and IC4 form a Schmitt trigger to square the signal for digital use. D10-11 give a visual indication of external switch activation.

IC5-6 combine the three possible inputs into two data lines. Pin 1 is high for a tone mark or the dot switch and Pin 2 is high for the dash switch. Both pins high indicates an error. SW2 informs the software whether a single or double switch is being used. Pins 1, 2 and 4 connect to data lines 0, 1 and 2 respectively on an input port on the microprocessor. The dual key and SW2 sense software was not



IC		+5	Gnd
4	CD4040	1	8
5	CD4011	14	7
6	CD4001	14	7

<u>Figure 13.</u> Digital Input Schematic

needed or written during development, but its addition would be trivial.

4.7 Microprocessor System

The microcomputer used for development is shown in Figure 14. It is based on the single board demonstration unit marketed by Tektron Inc. using the RCA 1802 CMOS microprocessor. It was chosen as its power requirements are a few milliamps at 5 volts, and it is thus suited for the ultimate goal of a portable device. It is an 8-bit machine with sixteen general purpose 8-bit registers and a suitable instruction set. Apart from RAM and ROM it is nearly a one chip processor, as the clock oscillator, DMA, interrupt and I/O ports are provided on the chip. Further details can be found in reference 16.

The system used for development was more complex than the capabilities of the chip might imply. The unit had to drive TTL circuits, which required buffering the data and address lines. The final system would be completely CMOS and the microprocessor could drive everything directly. Two serial I/O ports, including a cassette interface, were included, as well as a hexadecimal led display and an ASCII keyboard. On the demonstration board itself was an 8-bit latched output port driving eight leds and a hexadecimal keypad which was enlarged and brought out to the front

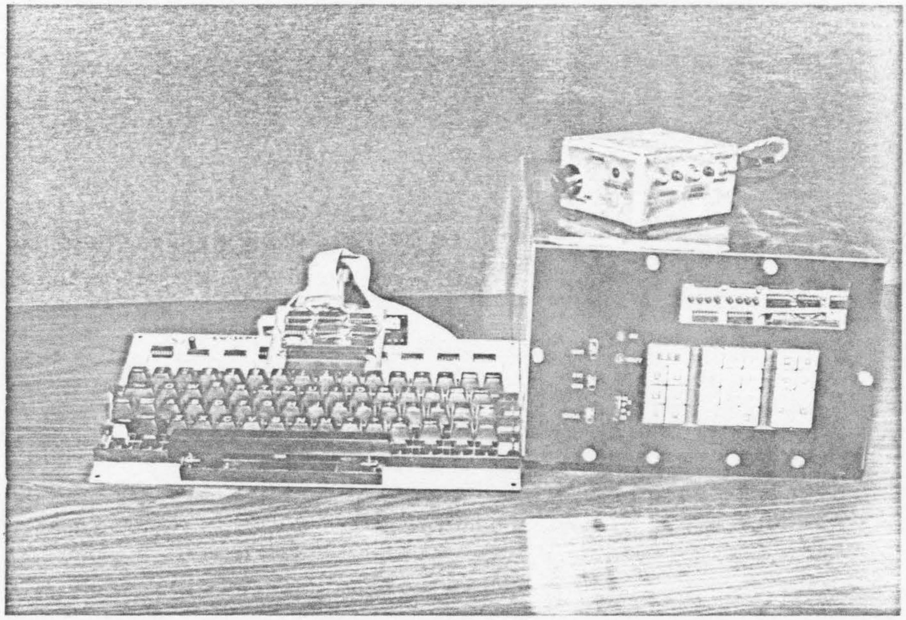


Figure 14. Development System.

panel. Memory consisted of 2K of RAM, but the first 512 bytes were replaced with EPROM containing the monitor.

The monitor program given in Appendix C allowed editing and execution of programs, and also contained utility routines for driving the various interfaces. Operation is based on the ASCII keyboard and the hexadecimal display, but it can be run from a terminal with less flexibility.

Circuitry for this system is not given as it does not relate directly to the final system. As pointed out above, that would consist of a one board computer without the extra buffering and peripheral capabilities.

CHAPTER V

RESULTS AND CONCLUSIONS

5.0 Testing and Results

Testing involved two main areas, testing with the patient and without. Both the hardware and the software were initially developed using the general rules for Morse code, and then modified as required.

The footswitch progressed from a single switch and a buzzer to a dual switch designed for the feet, plus an automatic keyer. The final combination of an adjustable wooden frame and microswitches covered in plexiglass proved to be functionally adequate and quite rugged. Required pressure for activation was easily varied by inserting foam rubber under the plexiglass. For a patient with weaker movements and more control it should be possible to place a single or dual switch entirely within a shoe using the Rancho type switch.

The automatic keyer was found to be necessary due to the subject's relatively poor control compared to a normal Morse operator. In this respect Morse code with its timing requirements might not be the ideal code for the cerebral palsied. Again, for a subject with better control the keyer may not be needed, or could be incorporated into the software.

The program began as a short decoder of ideal code with fixed dot/dash ratios. This performed perfectly in tests using perfect code from another 1802 system, but failed miserably using hand sent code. The final program was much longer than anticipated (as usual!), but produced readable code. Some tests were conducted using code from a shortwave radio using the phase locked loop detector described in the hardware section. This code was being sent at rates up to 20 times those anticipated for this project, but readable code was still produced. One caution to others trying this test is that amateur radio operators often use a bewildering array of abbreviations which at first appear to be bad decoding. Due to international regulations, it is unfortunately illegal to reproduce here any of those tests.

Tests with the patient were quite brief due to logistic problems. These sessions were taped however and used repeatedly. It was found that for use as a conversational aid extra non-standard characters had to be added to do things such as blank the display and perform a carriage return. Due to the very long times between letters the automatic letter space should be deleted and replaced with a special space code, although this was not done.

5.1 Trial Unit

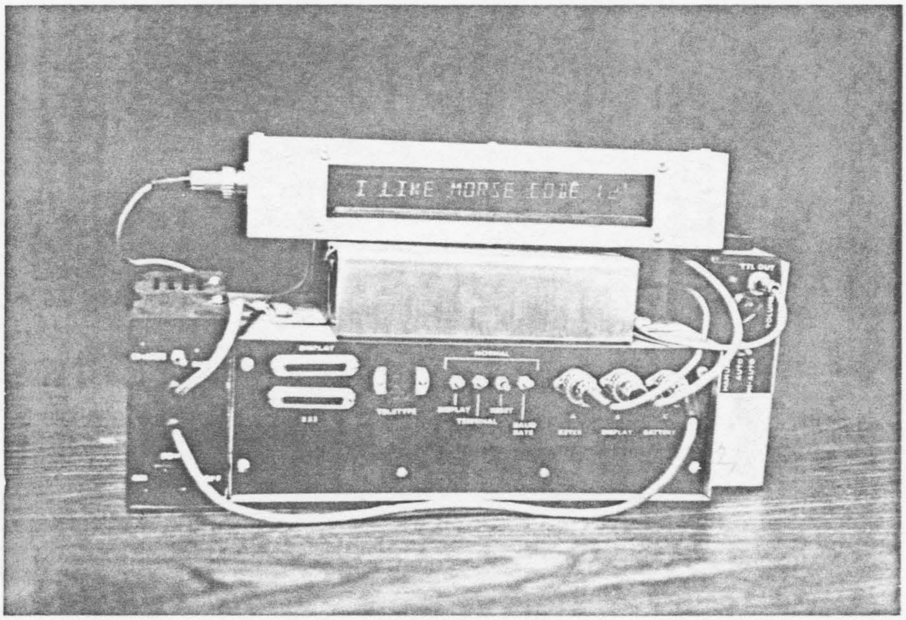
Near the end of the project a summer student built a portable decoder based on the work presented here. (15) The result, shown in Figure 15, consisted of the footswitch and keyer, the processor, the battery and charger, and a 20 character alphanumeric display. The size of the battery was due mainly to the fluorescent display.

Several trials were conducted with this apparatus on a wheelchair and it fulfilled the requirements. At the present time, intermittent electrical failures are preventing its use, but a more reliable unit will be produced.

5.2 Conclusions and Recommendations

The main objectives of the project were met, in that Morse code was shown to be a feasible means of portable communication for a cerebral palsied person.

Improvements can now be made all round, due to a better understanding of the problem, and more compact and lower power technology. Studies of the histograms generated by the user should be made so that the decoding algorithm can be tailored to the user. Convenience features can be included, such as the facility for the unit to operate as a computer terminal. A liquid crystal display would allow week long operation on a single charge, or a smaller battery. The actual switch required depends on the individual user, as with



<u>Figure 15.</u> Trial Unit.

all aids, but a lighter and more visually pleasing switch could be designed.

It should be noted that most of the above improvements are aesthetic (editing,lower weight, etc.) rather than functionally essential. It is the author's experience however that unless an aid is almost invisible in its active and passive states it is apt to be rejected by the user.

In closing, an aid is only useful if it is used! It is the author's hope that further work will be done in this area, both in clinical trials and final placing of devices with the handicapped.

APPENDIX A

MORSE CODE CHARACTERS

AND

PROGRAM COMMANDS

MORSE CODE CHARACTERS

A . -	N - .	1 . - - - -	Period . - . - . -
B - . . .	O - - -	2 . . - - -	Comma - - . . - -
C - . - .	P . - - .	3 . . . - -	Question . . - - . .
D - . .	Q - - . -	4 . . . . -	Error . . . . . .
E .	R . - .	5	Colon - - - . . .
F . . - .	S . . .	6 -	Semicolon - . - . - .
G - - .	T -	7 - - . . .	Bracket - . - - . -
H	U . . -	8 - - - . .	Backslash - . . - .
I . .	V . . . -	9 - - - - .	
J . - - -	W . - -	0 - - - - -	
K - . -	X - . . -		
L . - . .	Y - . - -		
M - -	Z - - . .		

PROGRAM COMMANDS FROM ASCII KEYBOARD

/ Restart at 03E0, call monitor for new sp ed

H Output histogram immediately, wait for more commands

K Send "COPY" command to graphics terminal, wait

M Restart at 03F6, some histogram, speed and thresholds

N Set non-auto mode, restart at 03F6 (cancel a "Z")

X Clear histograms and continue

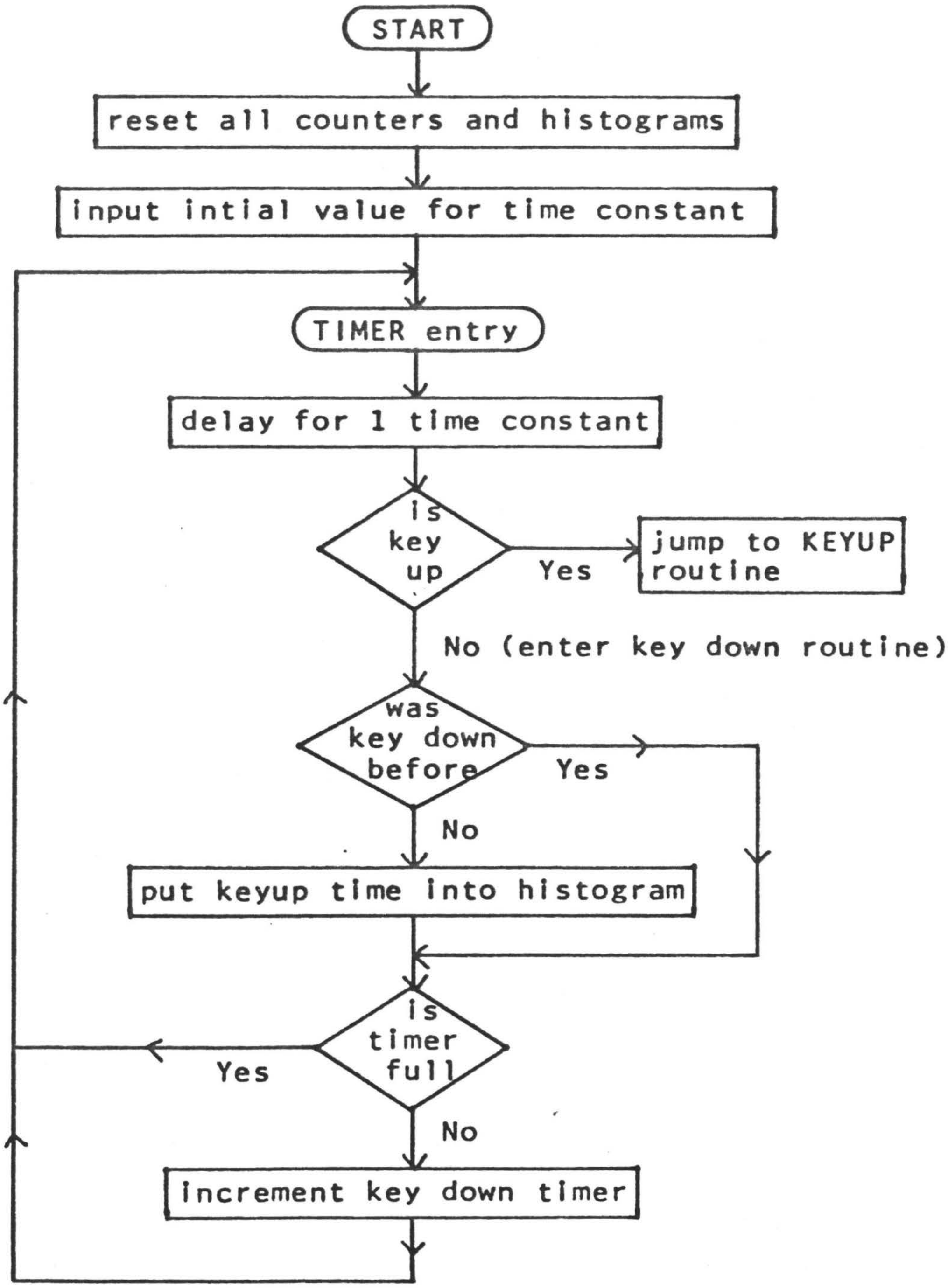
Y Divide histogram data by 2 and continue

Z Set automatic mode (output a histogram on any overflow)

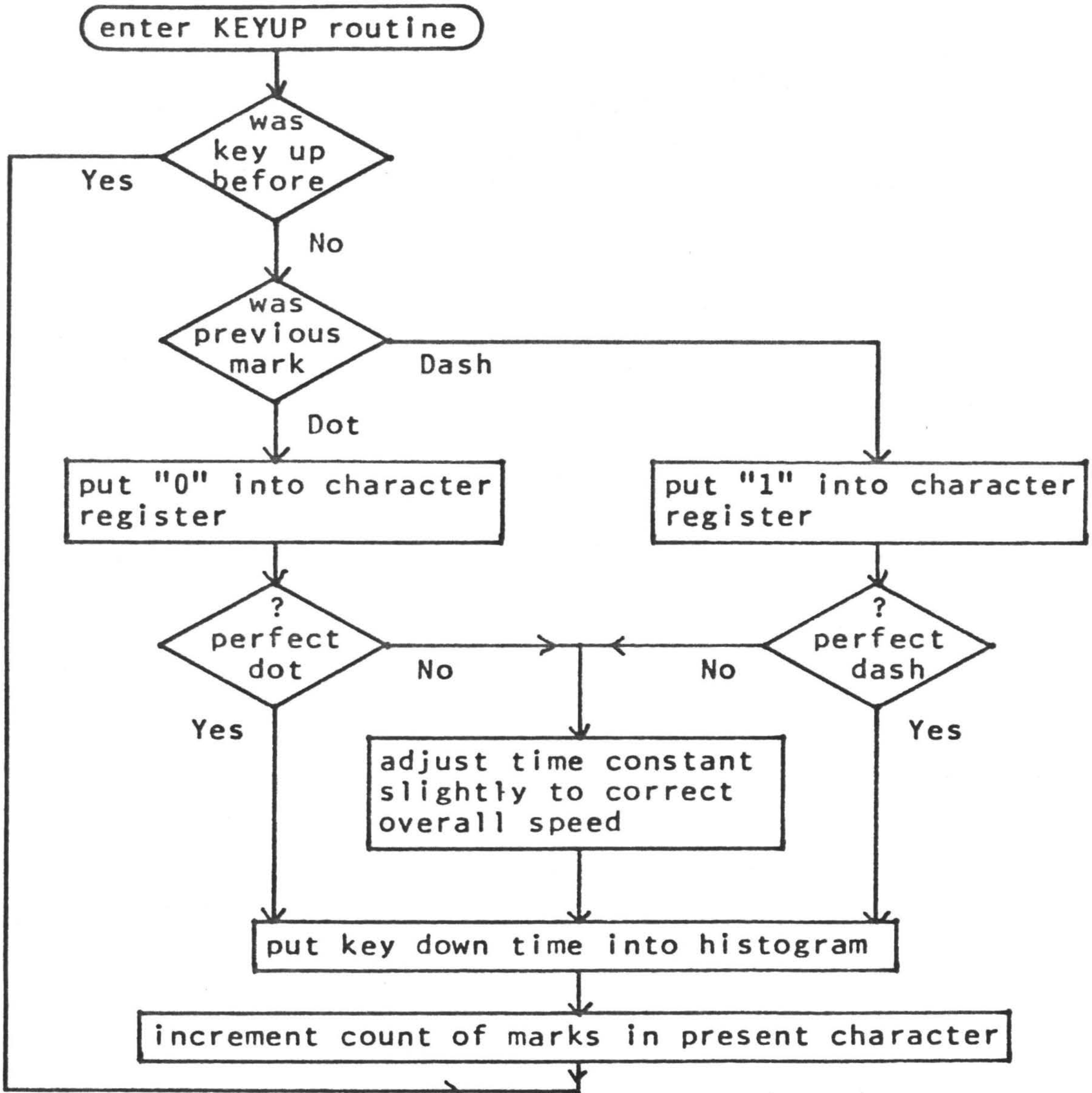
APPENDIX B

FLOWCHARTS

PROGRAM OVERVIEW

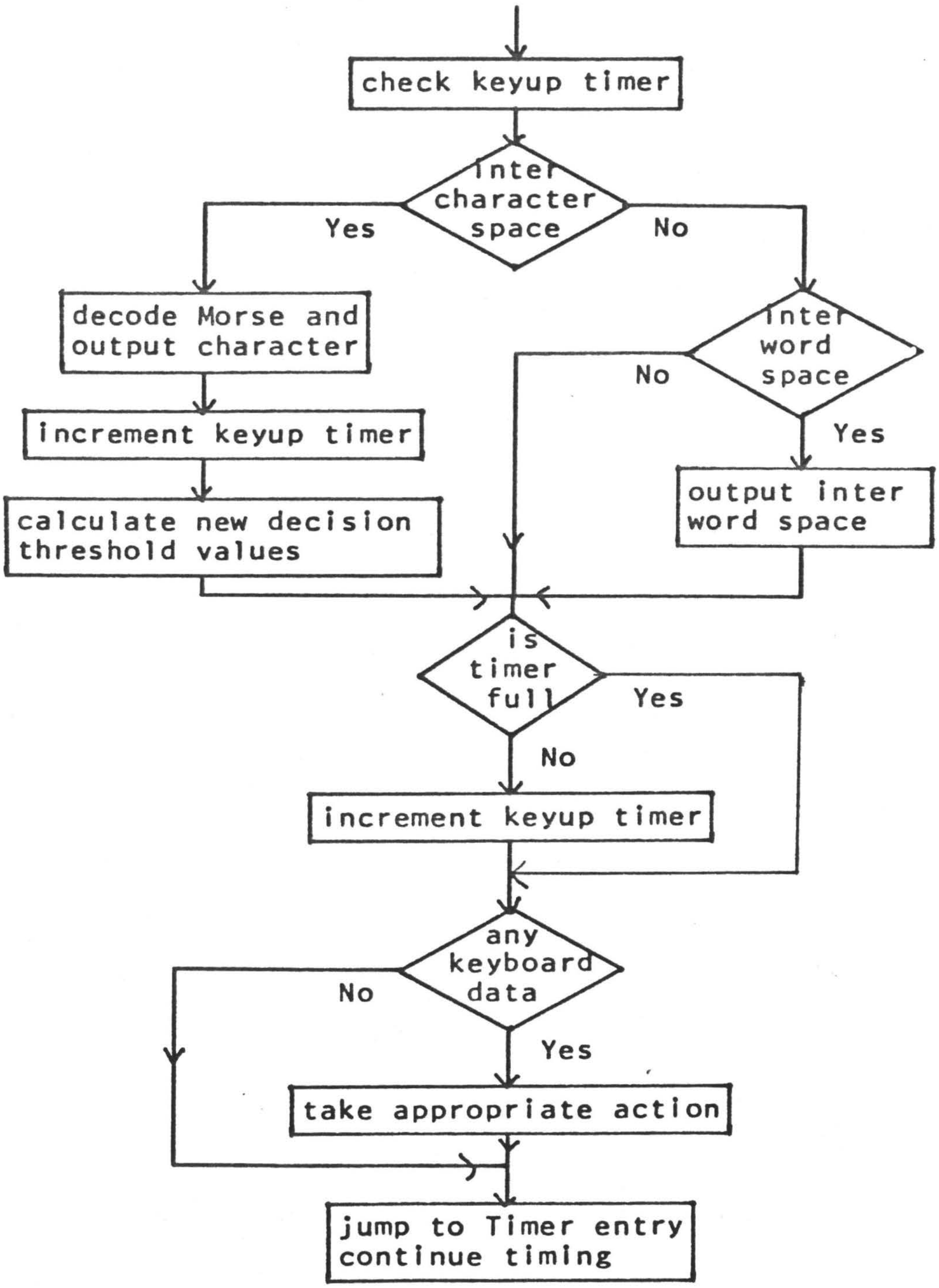


PROGRAM OVERVIEW (continued)

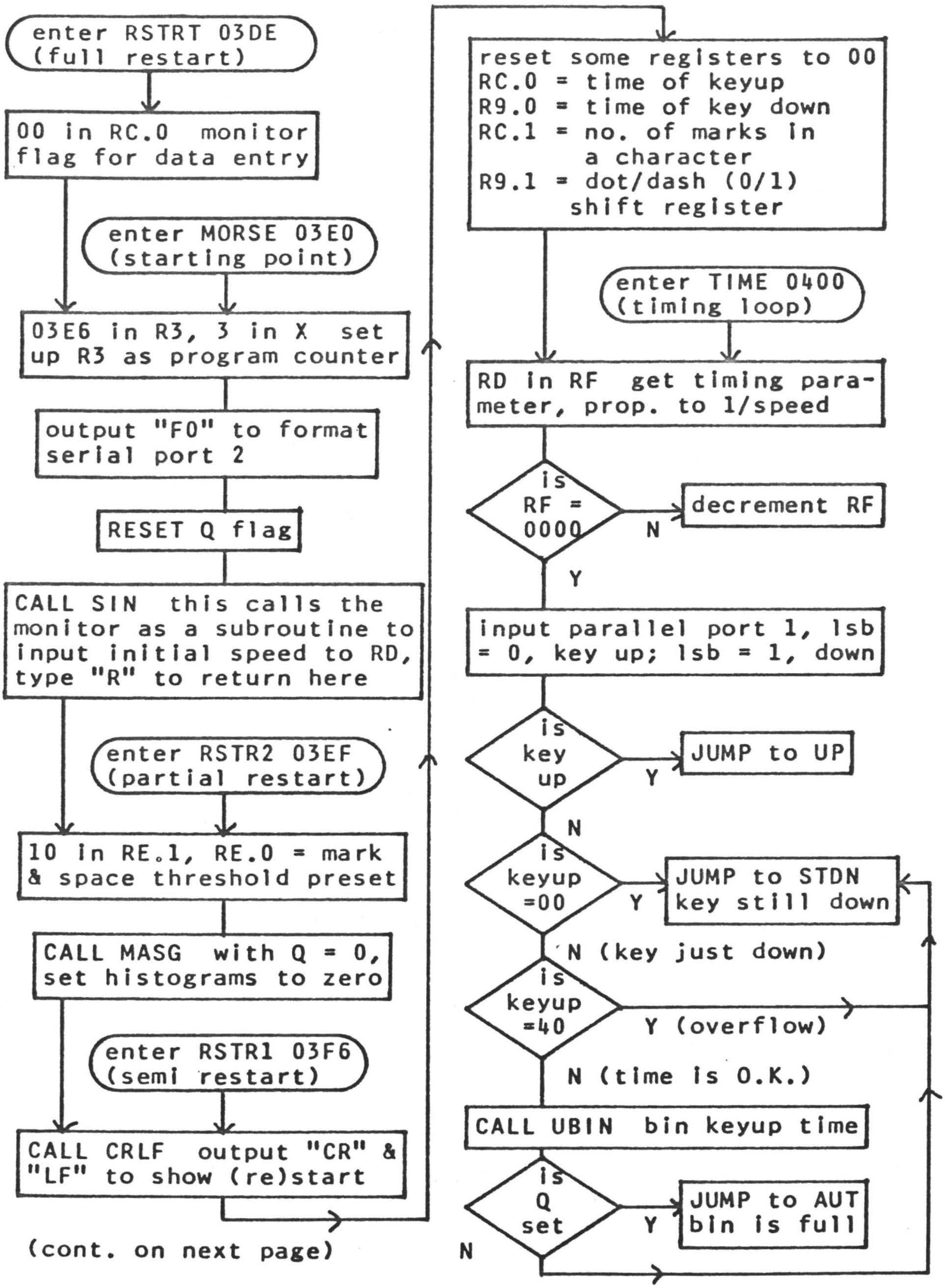


(continued next page)

PROGRAM OVERVIEW (continued)

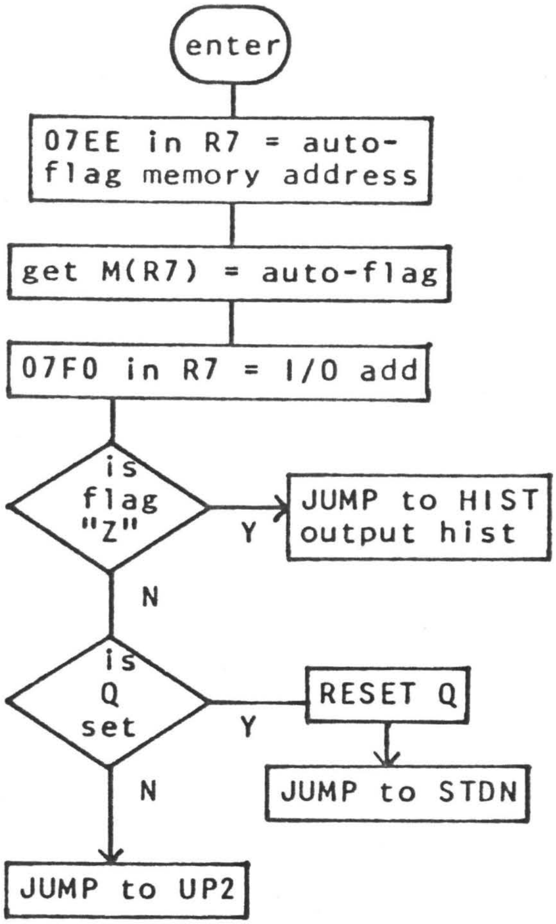


MORSE 03DE-03FF

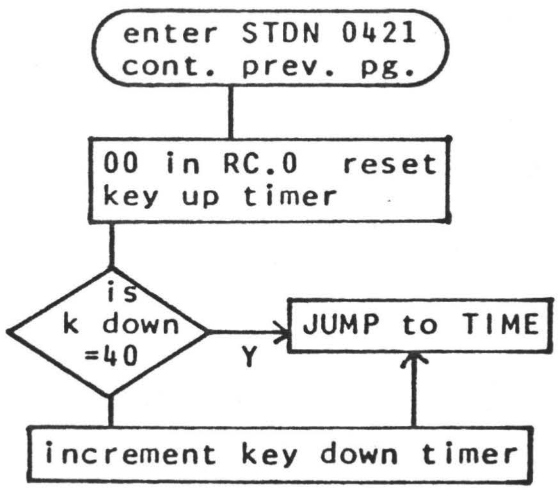


(cont. on next page)

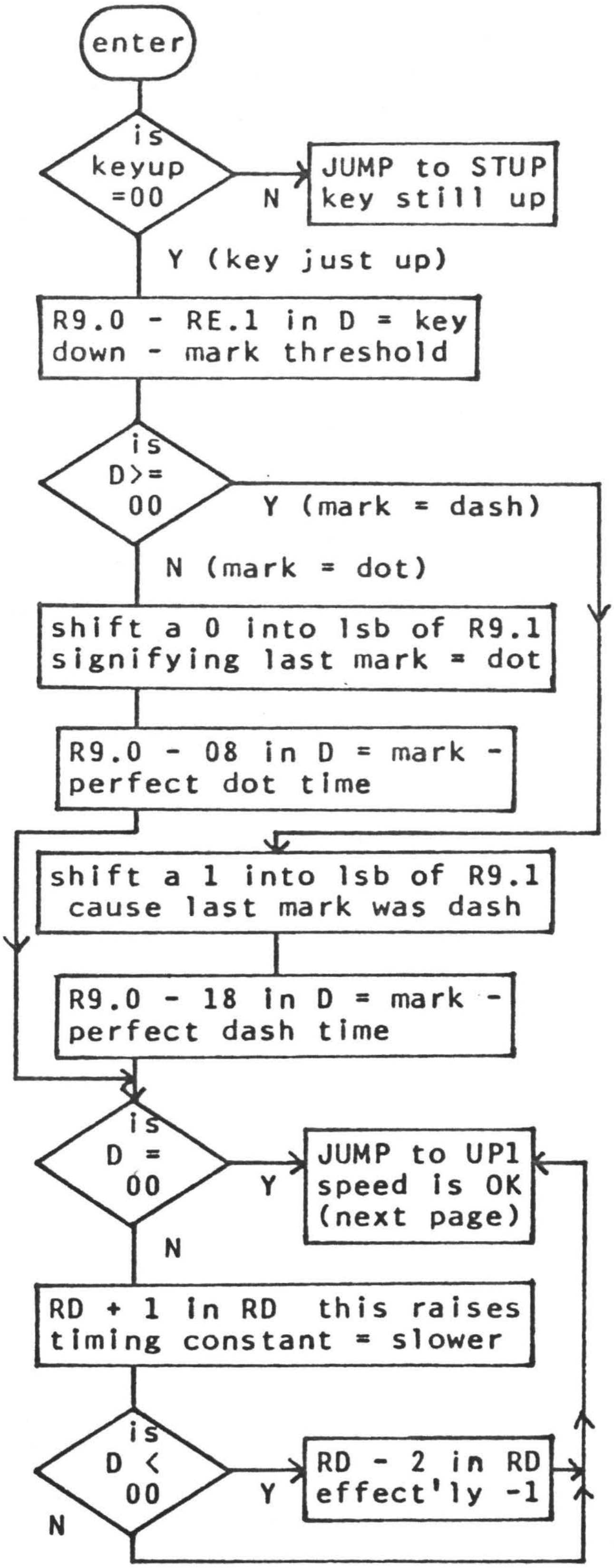
AUT? 0568-057F

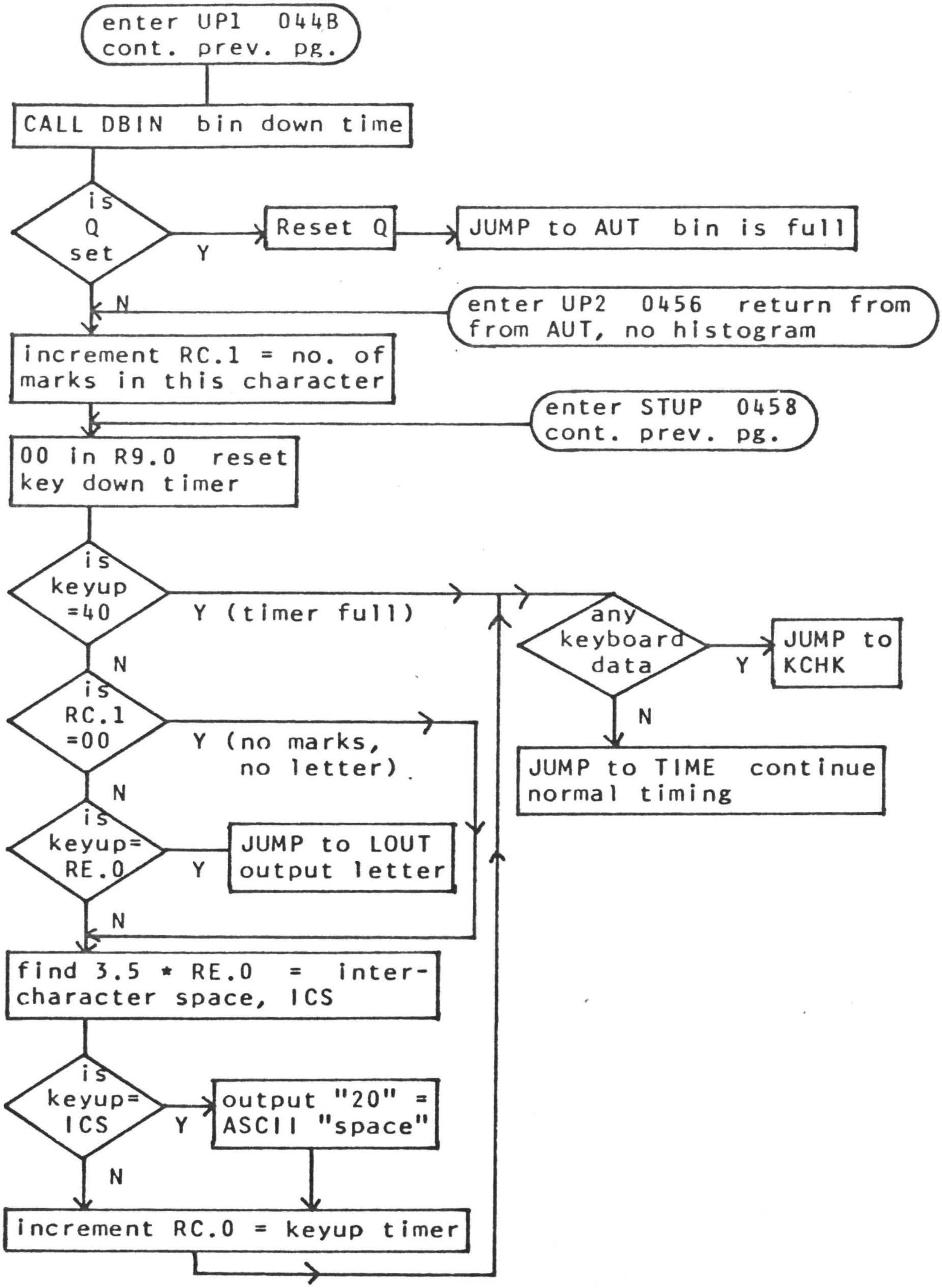


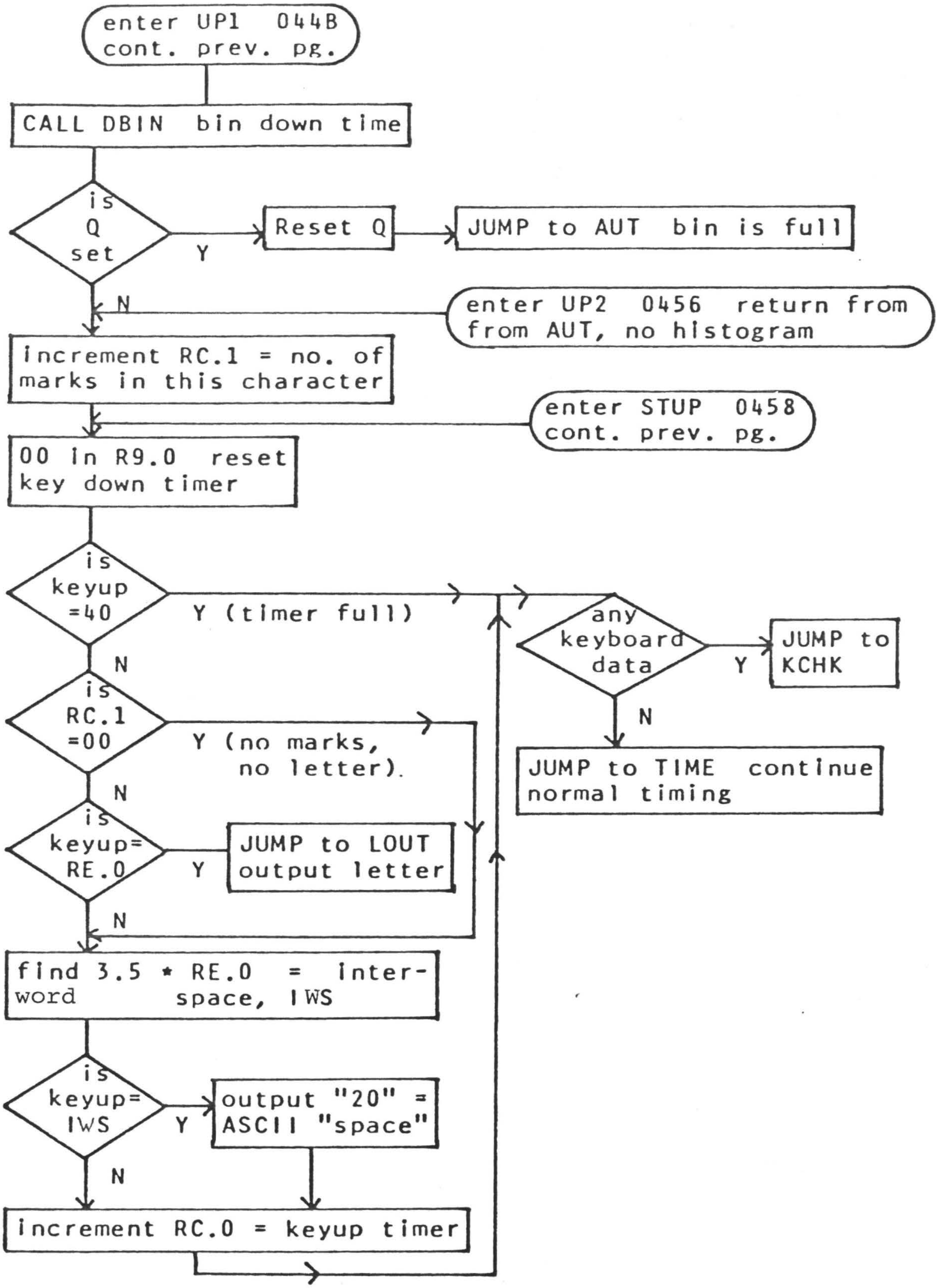
The "AUT" section of code is in a difficult position. It acts like a subroutine, but in fact "Q" is used to determine the "return" address.



UP 042C-047F

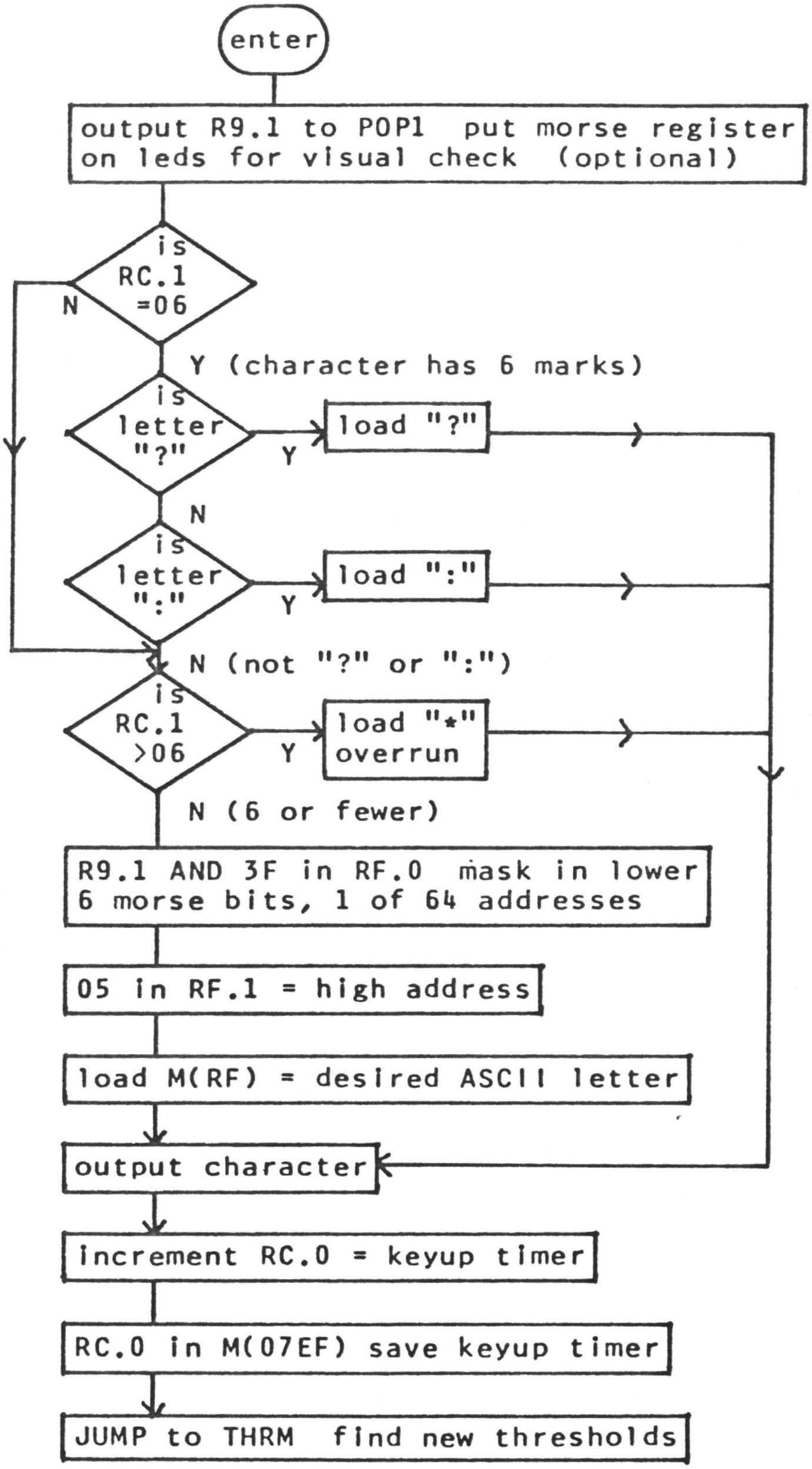



enter UP1 044B
cont. prev. pg.
CALL DBIN bin down time
is Q set
Y
Reset Q
JUMP to AUT bin is full
N
enter UP2 0456 return from
from AUT, no histogram
increment RC.1 = no. of marks in this character
enter STUP 0458
cont. prev. pg.
00 in R9.0 reset key down timer
is keyup =40
Y (timer full)
any keyboard data
Y
JUMP to KCHK
N
JUMP to TIME continue normal timing
N
is RC.1 =00
Y (no marks, no letter)
N
is keyup= RE.0
Y
JUMP to LOUT output letter
N
find 3.5 * RE.0 = inter-character space, ICS
is keyup= ICS
Y
output "20" = ASCII "space"
N
increment RC.0 = keyup timer

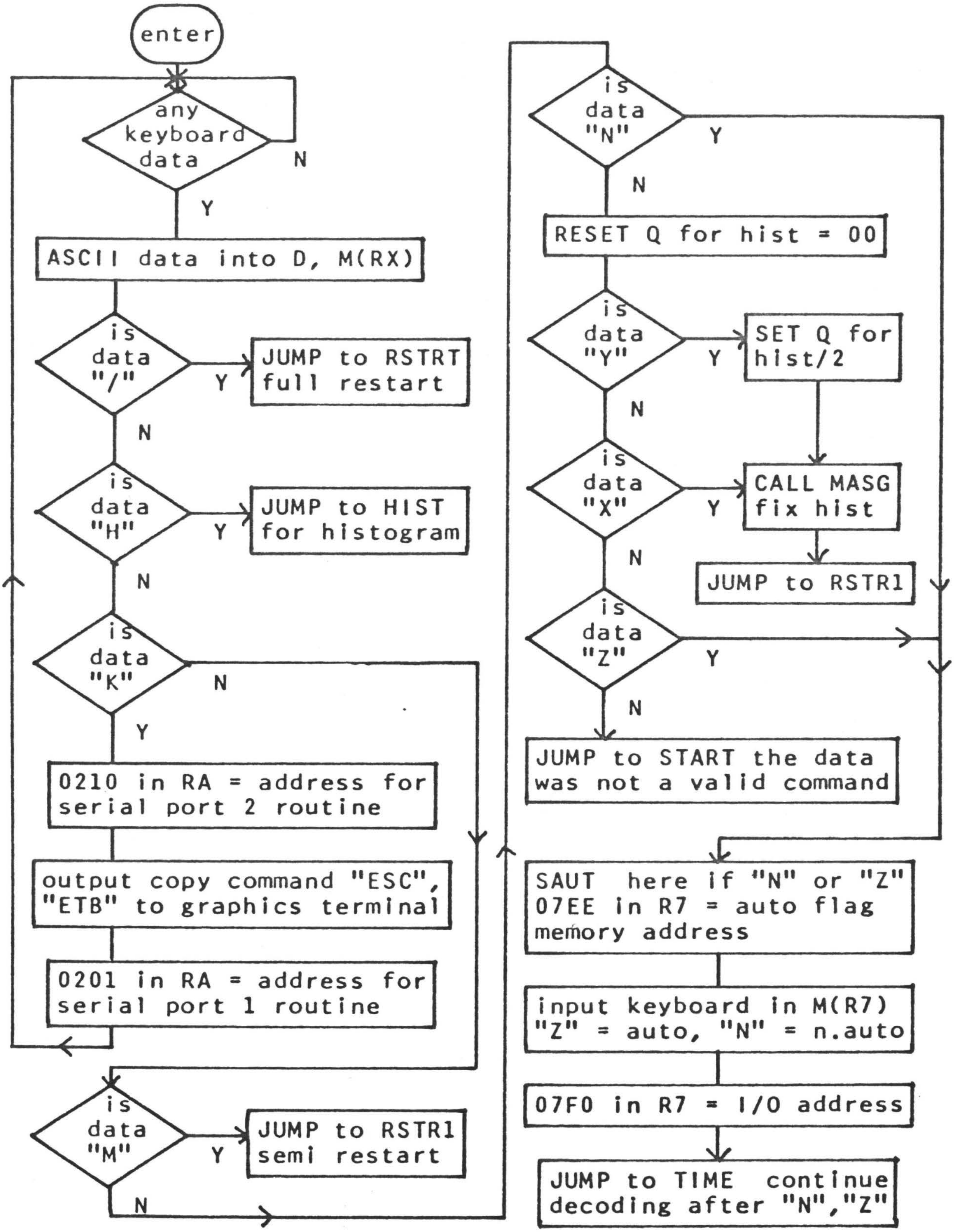


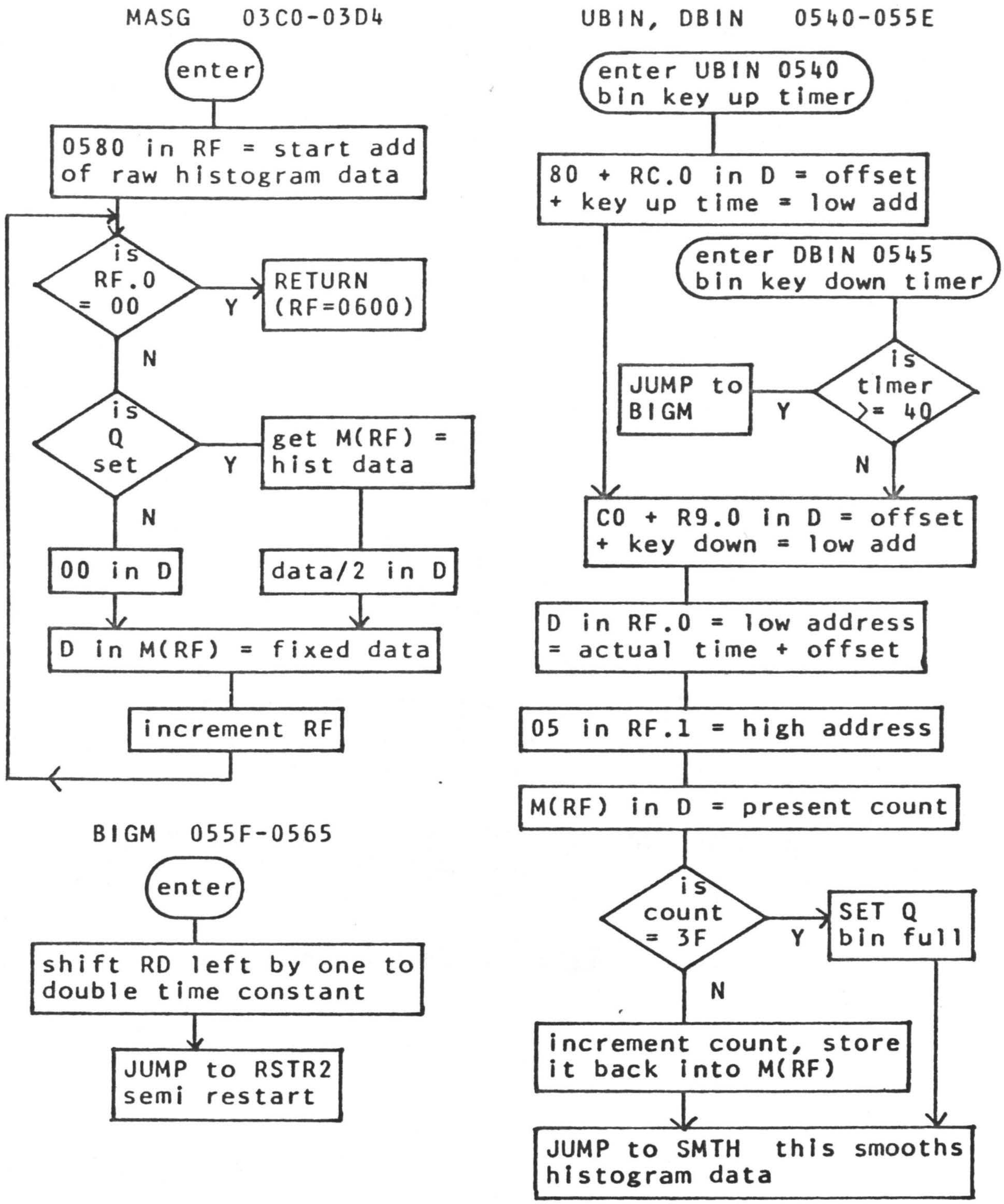
enter UP1 044B
cont. prev. pg.
CALL DBIN bin down time
is Q set
Y
Reset Q
JUMP to AUT bin is full
N
enter UP2 0456 return from
from AUT, no histogram
increment RC.1 = no. of
marks in this character
enter STUP 0458
cont. prev. pg.
00 in R9.0 reset
key down timer
is keyup =40
Y (timer full)
any keyboard data
Y
JUMP to
KCHK
N
JUMP to TIME continue
normal timing
N
is RC.1 =00
Y (no marks,
no letter).
N
is keyup= RE.0
Y
JUMP to LOUT
output letter
N
find 3.5 * RE.0 = inter-
word space, IWS
is keyup= IWS
Y
output "20" =
ASCII "space"
N
increment RC.0 = keyup timer

LOUT 0480-04B0

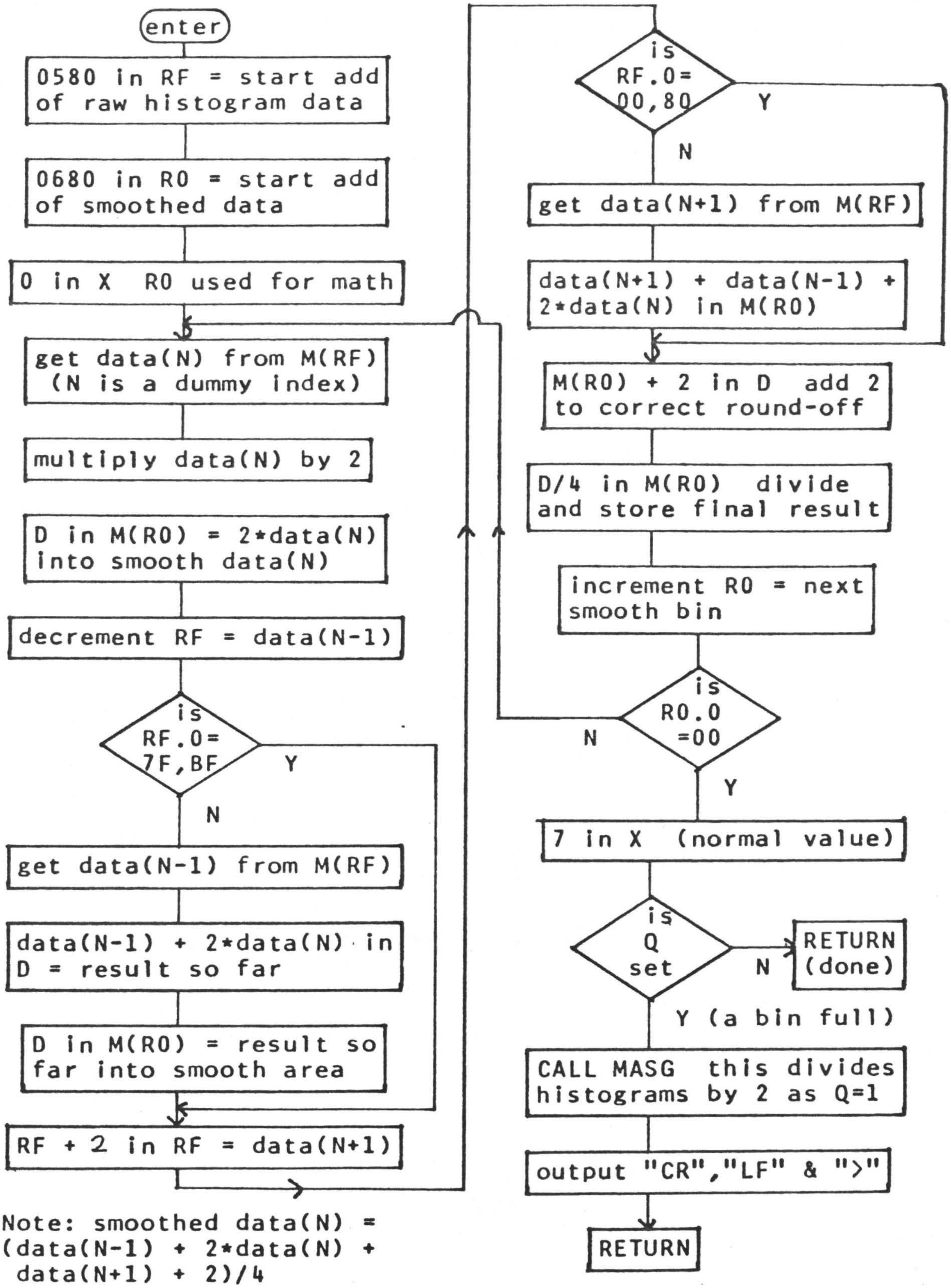


KCHK 04B8-04FC




MASG 03C0-03D4
enter
0580 in RF = start add of raw histogram data
is RF.0 = 00
Y
RETURN (RF=0600)
N
is Q set
Y
get M(RF) = hist data
N
00 in D
data/2 in D
D in M(RF) = fixed data
increment RF
BIGM 055F-0565
enter
shift RD left by one to double time constant
JUMP to RSTR2 semi restart
UBIN, DBIN 0540-055E
enter UBIN 0540 bin key up timer
80 + RC.0 in D = offset + key up time = low add
enter DBIN 0545 bin key down timer
JUMP to BIGM
Y
is timer >= 40
N
C0 + R9.0 in D = offset + key down = low add
D in RF.0 = low address = actual time + offset
05 in RF.1 = high address
M(RF) in D = present count
is count = 3F
Y
SET Q bin full
N
increment count, store it back into M(RF)
JUMP to SMTH this smooths histogram data

SMTH 0760-079F

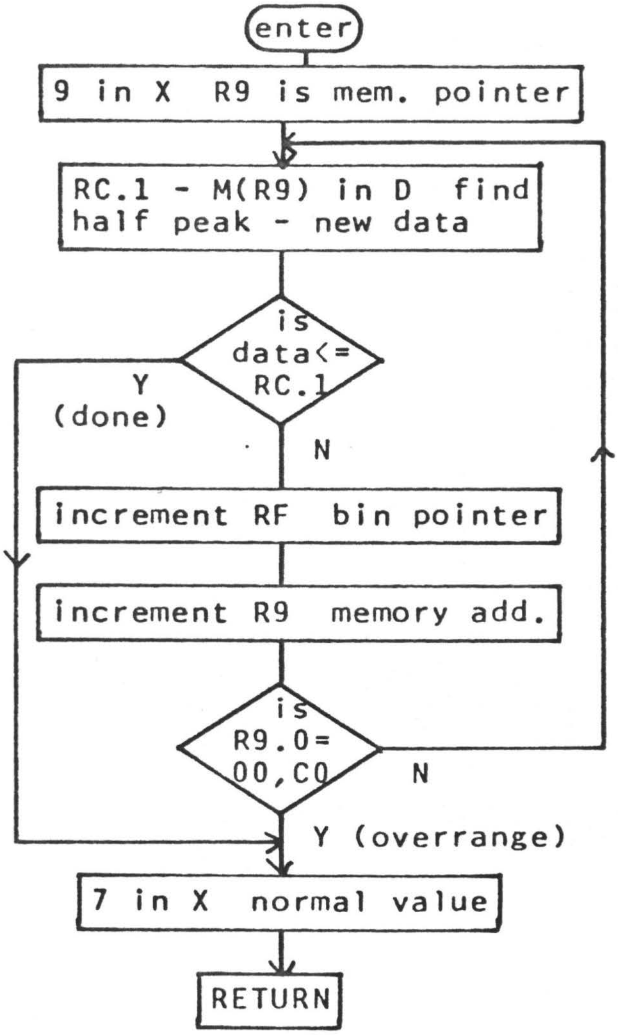


Note: smoothed data(N) = (data(N-1) + 2*data(N) + data(N+1) + 2)/4

LTOR 0710-0723

entry: half peak in RC.1
start address of search
in R9
00 in RF.0

exit: RF.0 = # of bins
from peak to first bin
<= 1/2 peak from left
to right

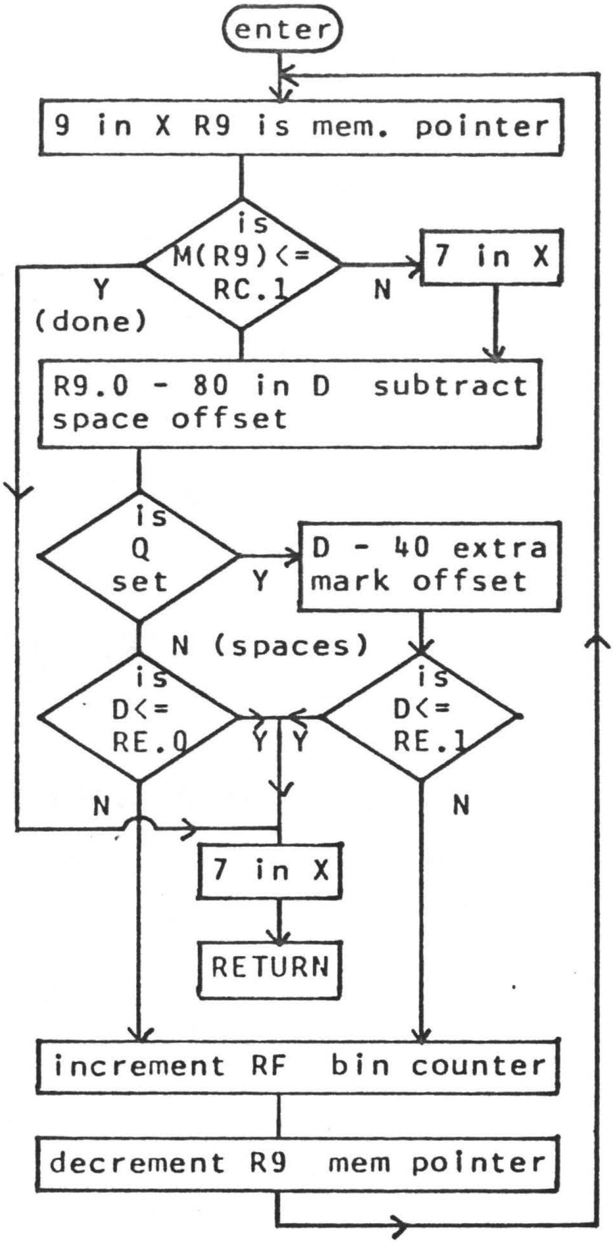


Note: LTOR and RTOL are
very similar and differ
in direction of scan and
overrange criteria.

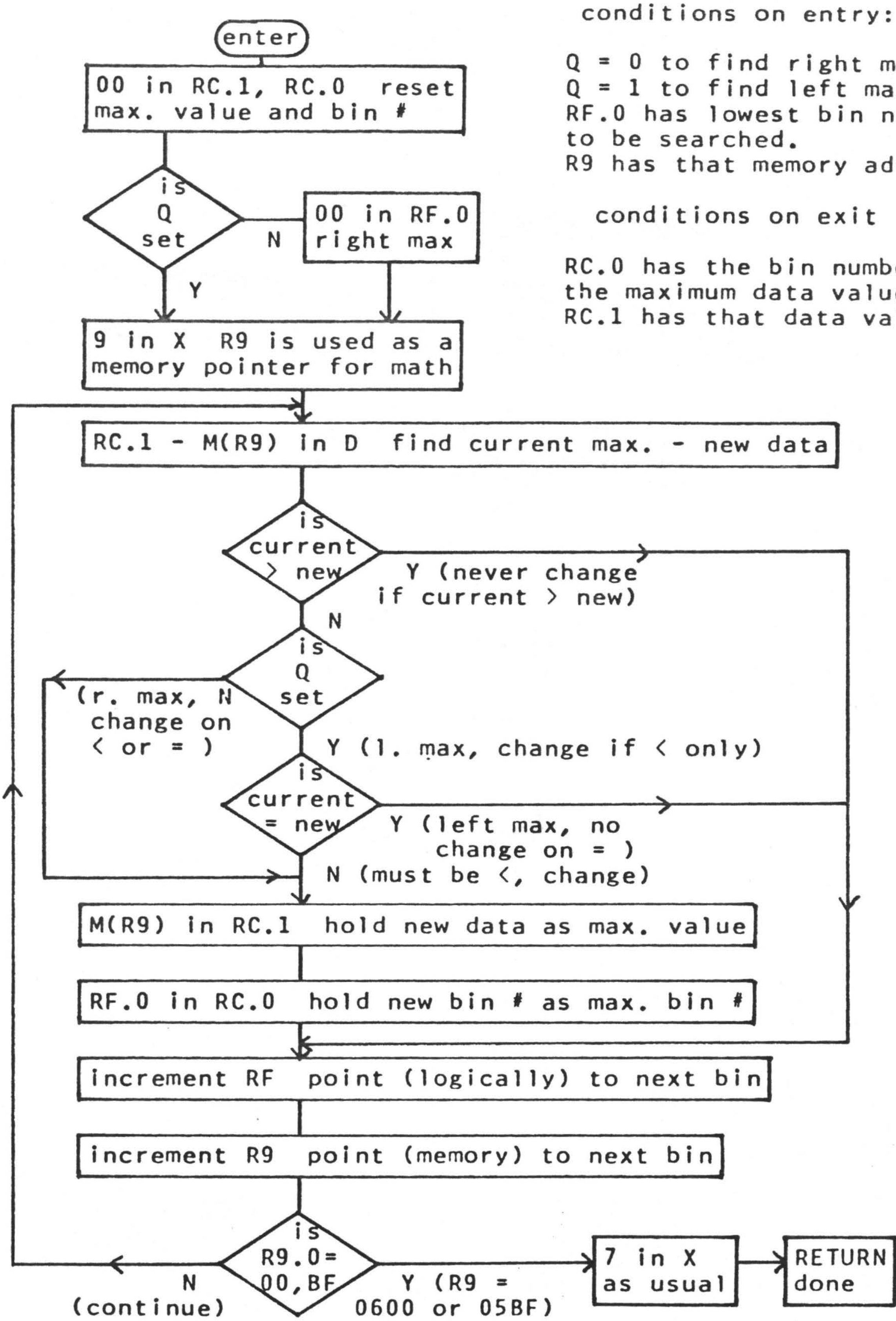
RTOL 0728-073F

entry: half peak in RC.1
start address in R9.0
00 in RF.0
lowermost bin # in RE.0 for
spaces, RE.1 for marks
Q = 0, spaces; Q = 1,marks

exit: RF.0 = # of bins from
peak to first bin <= peak
from right to left



FMAX 0740-075D



conditions on entry:

Q = 0 to find right maximum
Q = 1 to find left maximum
RF.0 has lowest bin number to be searched.
R9 has that memory address.

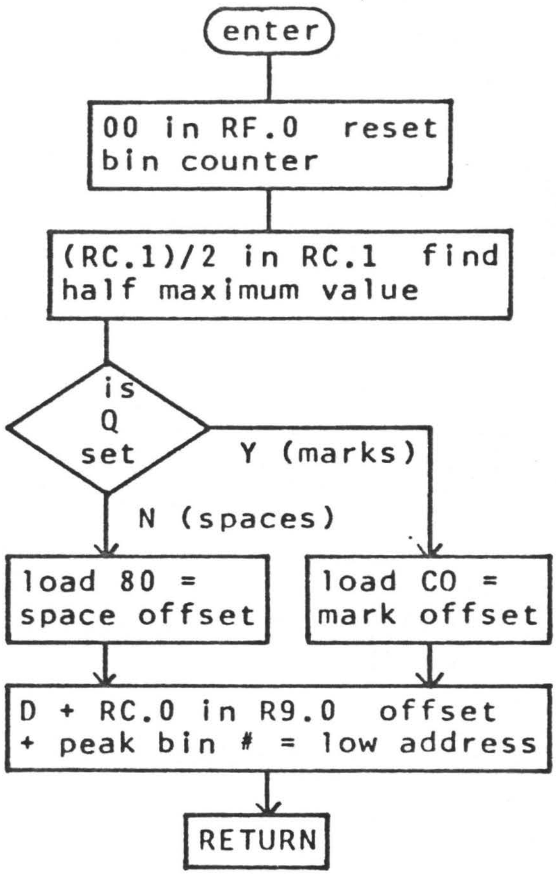
conditions on exit

RC.0 has the bin number with the maximum data value.
RC.1 has that data value.

ADD50 0700-070F

entry: peak value in RC.1
peak bin # in RC.0
Q = 1 for marks
Q = 0 for spaces

exit: half peak in RC.1
actual memory add.
of peak in R9.0



THRM 0600-063B

enter

06C0 in R9 start address of smoothed mark data

reset Q to find right maximum in FMAX

CALL FMAX find right maximum (dot peak, point 1)

set Q for mark routine in ADD50

CALL ADD50 fix data from FMAX

SUB LTOR find bin 50% or less than dot peak (point 2)

RC.0 + 2*RF.0 in D, RF.0 (counter) & RE.1 (left stop) find bin twice as far from peak as 50% bin, estimate of threshold based on dot peak and start point for dash peak search (point 3)

D + C0 in R9.0 convert this to a memory address

set Q to find left maximum in FMAX

CALL FMAX find left maximum (dash peak, point 4)

set Q for mark routine in ADD50

CALL ADD50 fix data from FMAX

SUB RTOL find bin 50% or less than dash peak (point 5)

RC.0 - 2*RF.0 in D find bin twice as far from dash peak as 50% bin, estimate of threshold based on dash peak, (point 6)

→(continued next page)

THRM (continued previous page)

(RE.1 + D)/2 in RE.1 average two estimates (dot, dash) for mark threshold, store it in RE.1 (point 7)

JUMP to THRS

THRS 0640-067F

This routine is very similar to THRM, and differs in detail only. IRCS refers to the intercharacter space, and IACS to the intracharacter space.

enter

0680 in R9 start address of smoothed space data

reset Q to find right maximum in FMAX

CALL FMAX find right maximum (IACS peak, point 1)

reset Q for space routine in ADD50

CALL ADD50 fix data from FMAX

SUB LTOR find bin 50% or less than IACS peak (point 2)

RC.0 + 2*RF.0 in D, RF.0 (counter) & RE.0 (left stop) find bin twice as far from peak as 50% bin, estimate of threshold based on IACS peak and start point for IRCS peak search (point 3)

D + 80 in R9.0 convert this to a memory address

set Q to find left maximum in FMAX

CALL FMAX find left maximum (IRCS peak, point 4)

(continued next page)

THRS (continued previous page)

↓

reset Q for space routine in ADD50

CALL ADD50 fix data from FMAX

SUB RTOL find bin 50% or less than IRCS peak (point 5)

RC.0 - 2*RF.0 in D find bin twice as far from IRCS peak as 50% bin, estimate of threshold based on IRCS peak, (point 6)

(RE.0 + D)/2 in RE.0 average two estimates (IACS,IRCS) for space threshold, store it in RE.0

(RE.0 + RE.1)/2 in RE.0 average this estimate for the space threshold with estimate for the mark threshold, due to the low "quality" of the space information, store it in RE.0 (point 7)

↓

JUMP to CLUP

↓

CLUP 07B0-07BD

↓

SUB DISPY display new thresholds

00 in R9.0 reset R9.0 for use as key down timer

00 in RC.1 reset RC.1 to count number of marks/character

01 in R9.1 preset morse shift register with leading 1

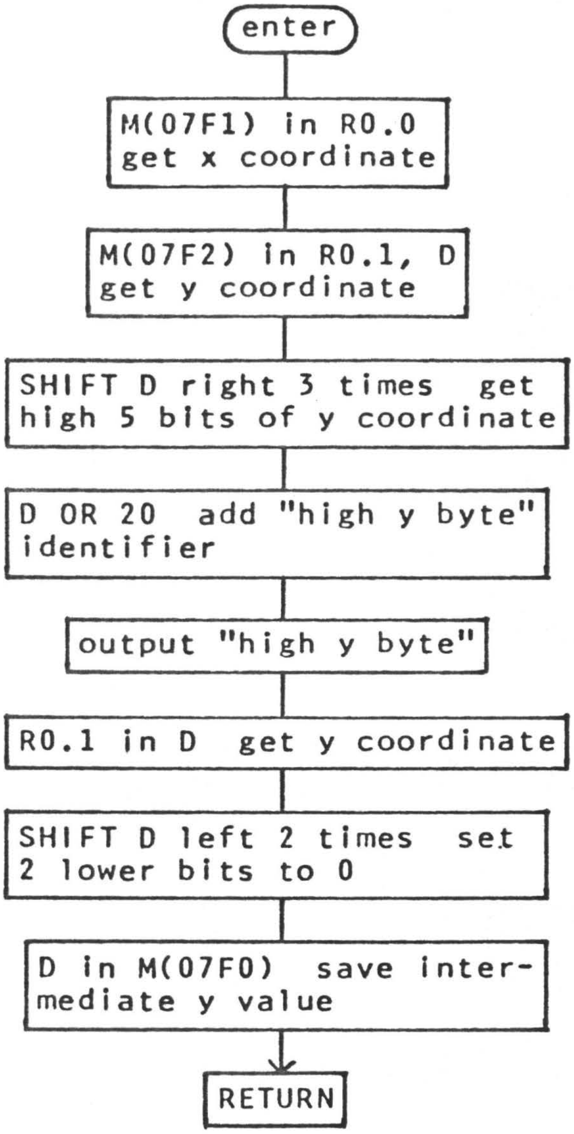
M(07EF) in RC.0 restore key up time value to RC.0

↓

JUMP to TIME thresholds found, continue regular timing

XYOU1 02E0-02F1

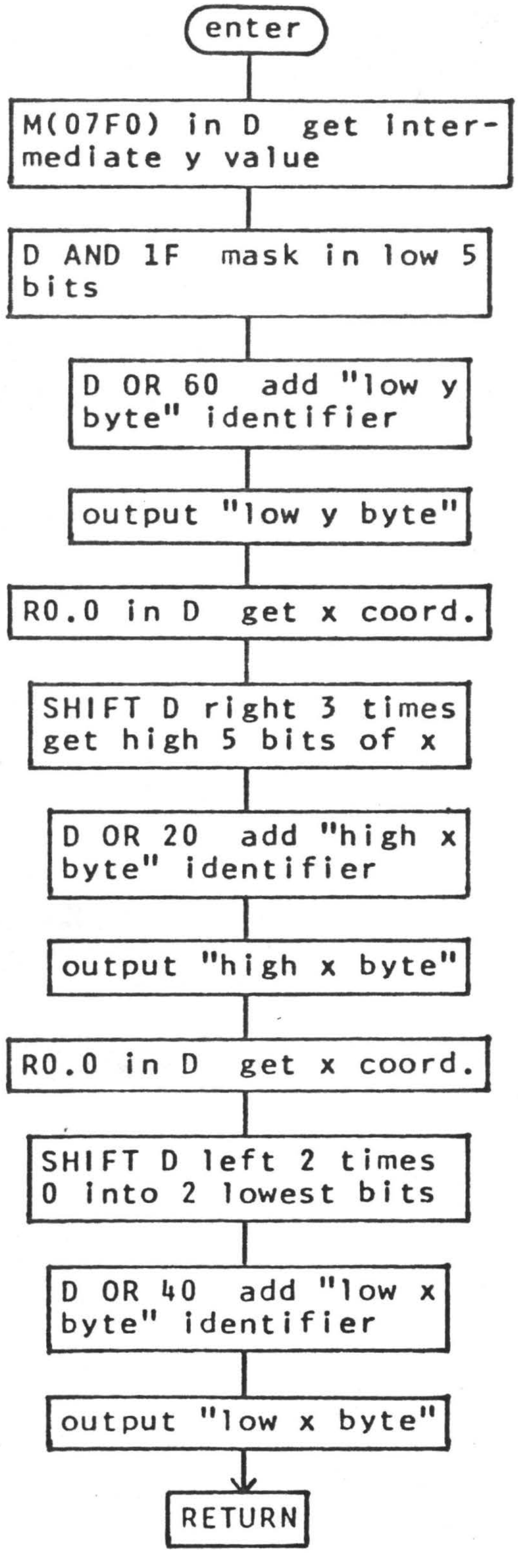
entry: 8 bit x coordinate in M(07F1), y in M(07F2)



At this point the intermediate y value in M(07F0) can have the two lowest bits set by the main program. This would allow the full 10 bit accuracy of the graphics terminal to be used.

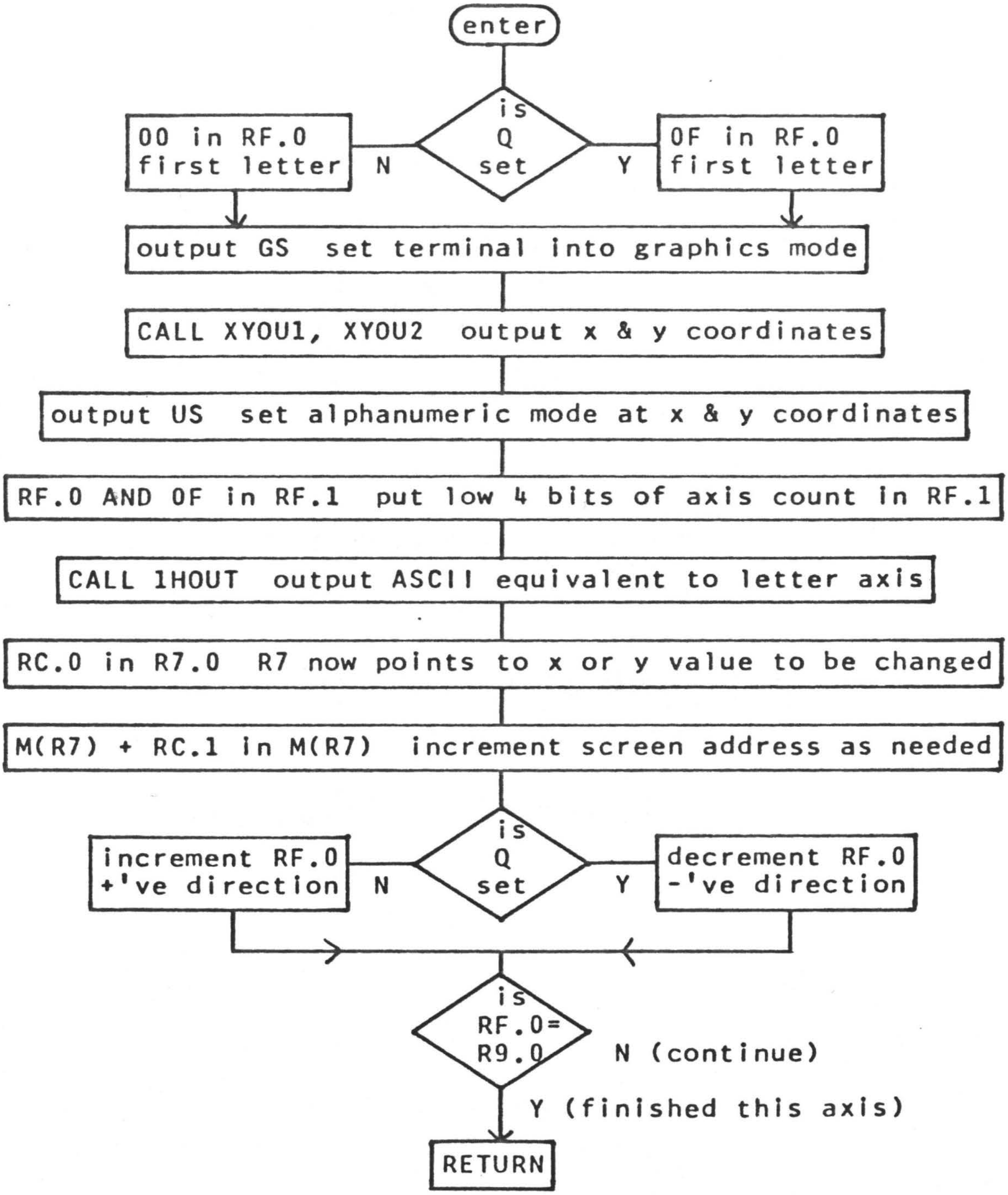
XYOU2 02F8-030D

entry: intermediate y value in M(07F0)

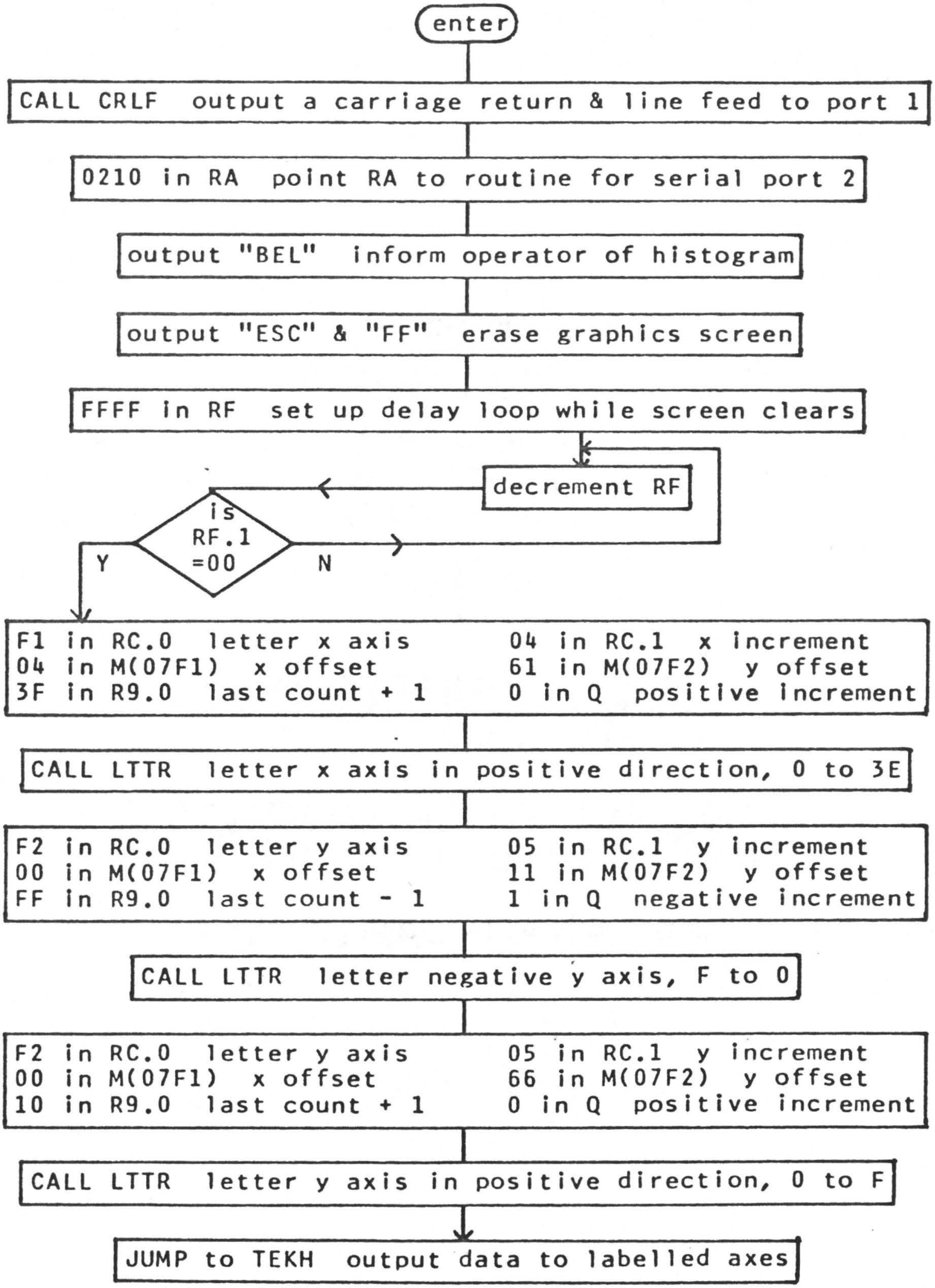


LTTR 0220-024B

entry: RC.0 = F1 for x axis, F2 for y axis; a low mem. add.
Q = 0 to letter in +'ve direction (x or y) and v.v.
RC.1 = screen address increment per letter
M(07F1) = x coordinate, M(07F2) = y coordinate
R9.0 = last position to be lettered, +/- 1

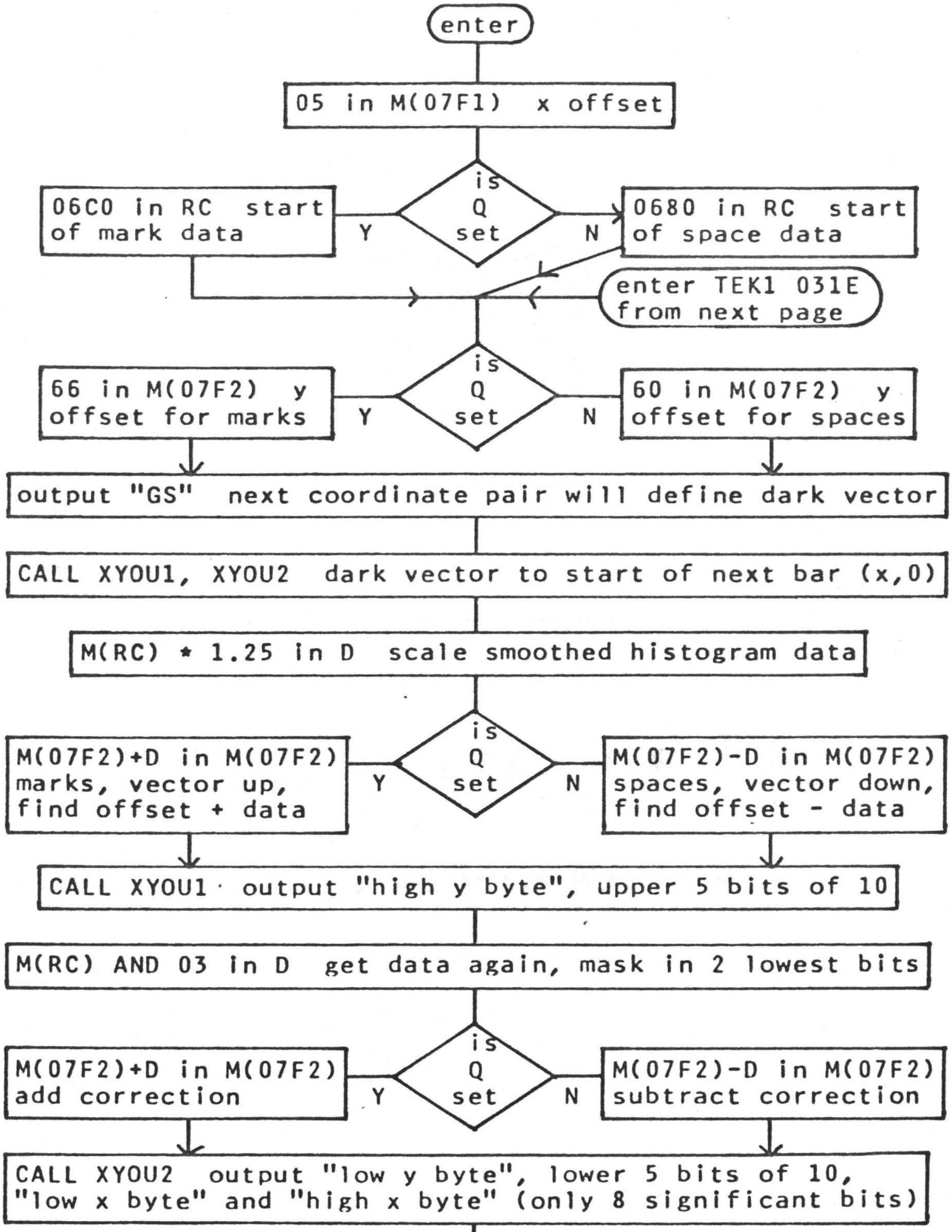


HIST 0250-036E

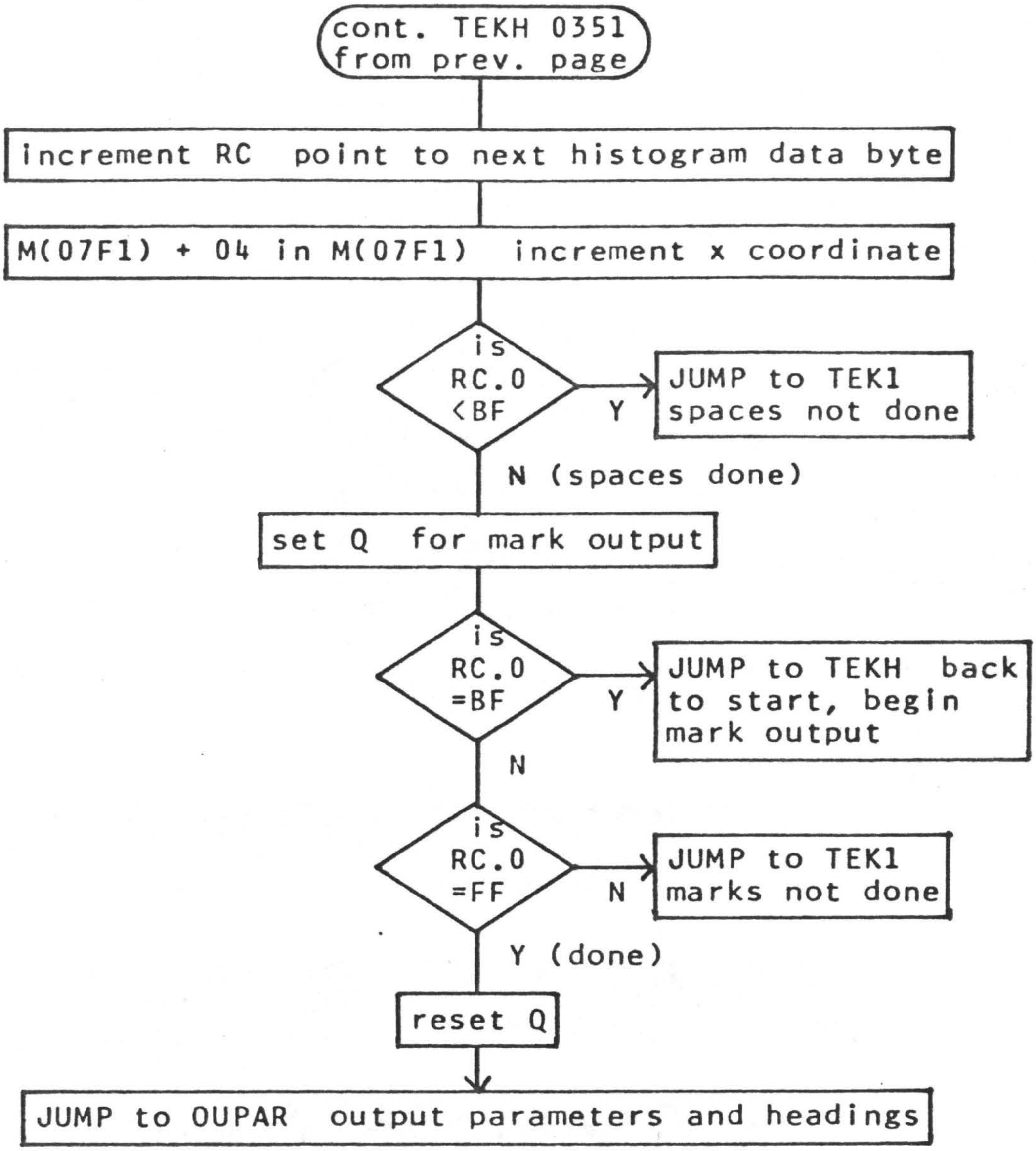


TEKH 0310-036E

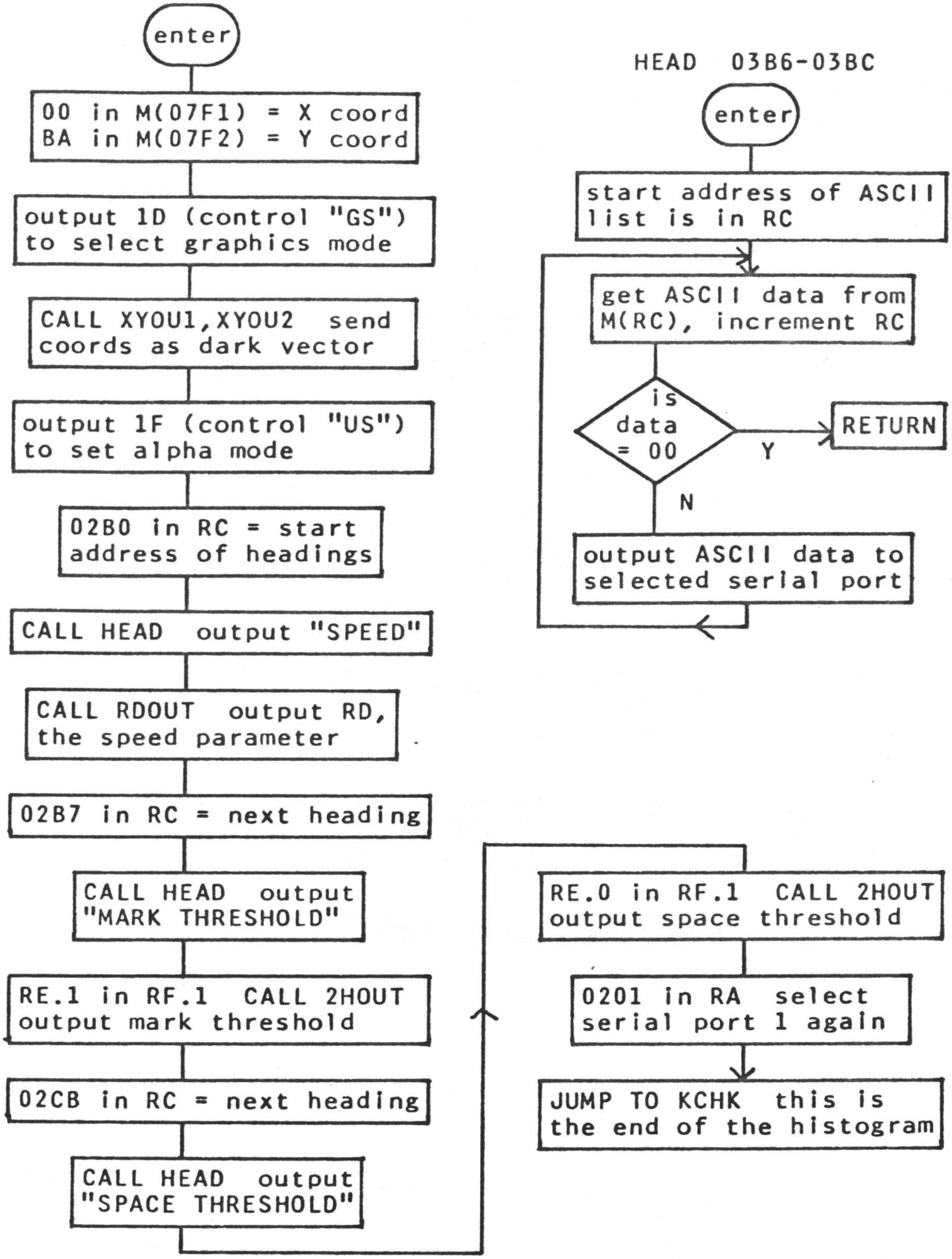
entry: Q = 0, space data is output first



(continued next page)


cont. TEKH 0351
from prev. page
increment RC point to next histogram data byte
M(07F1) + 04 in M(07F1) increment x coordinate
is RC.0 <BF
Y
JUMP to TEK1
spaces not done
N (spaces done)
set Q for mark output
is RC.0 =BF
Y
JUMP to TEKH back to start, begin mark output
N
is RC.0 =FF
N
JUMP to TEK1
marks not done
Y (done)
reset Q
JUMP to OUPAR output parameters and headings

OUPAR 0370-03AF



APPENDIX C

MONITOR PROGRAM

MONITOR OPERATION

Most of the monitor resides in EPROM from 0000 to 01FF. The two serial output drivers sit in RAM from 0200 to 021F and must be reloaded before use. To start the monitor hit "Reset" and "Run" on the front panel. The available commands are all single key commands which are immediately acted upon, no carriage return is required. All addresses or data are displayed on the 8 digit hexadecimal display above the ASCII keyboard. RD is displayed in the left four digits and RE in the right.

```
 Key            Function
 break      ;Interupts the processor and restarts the
;monitor at 0000. This may be used at any time to regain
;control from a bad program which has NOT used R1.
 space      ;Normally numeric data goes to RD, but after the
;space bar is pressed data goes to RE. After the next
;command data flow returns to RD.
 0-9, A-F   ;Enters hexadecimal numbers to RD or RE.
 ?          ;The data in M(RD) is displayed in RE.0, RE.1=00
 I          ;Use after "?" to display successive locations.
 !          ;The data in RE.0 is entered into M(RD). If "!"
;is hit again, RD is advanced by 1 and RE.1=00. New data
;can now be entered in RE.
 $          ;Start execution of a program at M(RD) with R0
;as RP. R3 should be made RP at the start of that program.
 P          ;All the data from M(RD) to M(RE) is moved down
;by one to M(RD+1) to M(RE+1). This is useful but any jump
;addresses must be corrected by hand.
 G          ;Any data from serial port 1 is dumped into
;M(RD) to M(RE). When (RE-RD) bytes have been collected or
;if there were any errors the program returns to the
;monitor. This is usually used for loading from a cassette.
 W         ;All the data from M(RD) to M(RE) is dumped out
;serial port 1. This is usually used to record a cassette.
 R         ;If the monitor was called as a subroutine by
;a running program this command is used to return to that
;program. This is useful for entering data during execution
 TT        ;This is a two key command. The data from M(RD)
;to M(RE) is output to serial port 1 as double hexadecimal
;ASCII characters, with each pair followed by "CR" & "LF".
;This is used to obtain listings on the Teletype.
 TP        ;This is the same as "TT" except that the address
;is output at the beginning and at every nnn0 address.
```

Many useful routines are hidden in the monitor and can be called by other programs. The whole monitor can be called, with return effected by "R". Some routines will return automatically if "Q" is set.

I/O PORTS & FLAG ASSIGNMENTS

The three "N" lines from the 1802 are decoded to seven lines (called "DEC N") to allow direct access to seven I/O ports.

```
 Code   Mnemonic                Comments
  61    OUT SER1       ;This is serial port 1, a UART which
;drives an FSK cassette, 20 ma loop TTY, and EIA RS 232
;interfaces. Baud rate is 110 or 600, format is 7 or 8 bits
;with optional parity, selectable by front panel switches.
;EF3 = 1 signifies transmitter busy, EF3 = 0, ready.
  69    INP SER1       ;This is serial input port 1, and it
;is configured the same way as the output port. The three
;input interfaces are OR'd, so only one should be used at
;time. Input and output may proceed simultaneously. UART
;error flags are OR'd onto a flag: EF1 = 1 means error,
;EF1 = 0, no error. EF2 = 0 means new data ready, EF2 = 1
;no data. There is a front panel switch to connect EF2 to
;the DMA so tapes may be loaded with no bootstrap at 0000.
  62    OUT POP1       ;Parallel output port 1, a set of 8
;leds on the CPU card and a dip connector.
  6A    INP HEX        ;This is the hex keyboard on the front
;panel. If the CPU is in the load mode, each pair of hex
;digits will be DMA'd into M(R0). It can be read with 6A.
  63    OUT DISP       ;Output to the hex led display. The
;data must be formatted by subroutine DISPY. This display
;is used extensively by all programs.
  6B    INP ASKEY      ;Input the ASCII encoded keyboard.
;EF4 = 1 means new data, EF4 = 0, no data.
  64    OUT POP2       ;This is a one bit (D0) output port
;used by the morse program to output the sampled key as
;a check on sampling rate. It is not essential.
  6C    INP 4          ;Not used.
  65    OUT 5          ;Not used.
  6D    INP 5          ;Not used.
  66    OUT SER2       ;Output data to serial port 2. This
;port has 20 ma TTY and EIA RS 232 interfaces. The baud
;can be set on the back panel from 75 to 4800.
  6E    INP SER2       ;Input data from serial port 2.
  67    OUT CON2       ;This sets the control register on
;serial port 2. D7=PI, D6=SBS, D5=WLS1, D4=WLS2, D3=PS.
;D2-D0 are not used. Output "F0" to set 8 data bits, no
;parity and 2 stop bits.
  6F    INP PIP1       ;D7=1 if serial port 2 is ready to
;send, D6=1 for new data ready, D5=1 for any errors. D4
;and D3 are not used. D2 - D0 connect to the morse inter-
;face, D0=1 for key down.
```

LABELS

;Some labels are entry points of significance, but most
;are merely for internal branches and deserve no comment.

Add.	Label	Comments
0000	START	;Beginning of monitor, resets all required ;registers. The interrupt is not enabled ;until this is complete. Many routines and ;the interrupt come to this point.
0030	CHECK	;If Q=1, executes an SCRT RETURN from here.
0033	SIN	;Enter monitor here as a subroutine.
0036	DIN	;Enter here if input already in RF.1.
003E	S1	;
004B	S1B	;
0070	ATOF	;Jump here if monitor recognizes hex A to F
0074	NUM	;and here if 0 to 9.
007D	MOSH	;
008D	LRE	;
0098	SHOV?	;
00A4	LOGAN	;Here from look routine to examine next add.
00A5	LOOK	;Routine to examine memory. Not a subroutine.
00B8	RENM	;
00C0	MOD	;Routine to modify memory. Not a subroutine.
00DB	SOVER	;
00DC	SRX4	;SEP subroutine to shift D right four places.
00E2	HOVER	;
00E3	HXOUT	;SEP subroutine to output to hardware display
00F4	INPUT	;SCRT subroutine waits for input from serial ;port 1 or the ASCII keyboard. Data returned ;in M(RX) and RF.1.
00F9	KB?	;
00FC	GOTIT	;
0100	EXITC	;
0101	SUB	;SEP subroutine for subroutine calls by SCRT.
0112	EXITR	;
0113	RETRN	;SEP subroutine for SCRT subroutine exits. ;See RCA 1802 Users Manual for SEP and SCRT.
0120	WRITE	;Cassette or paper tape dump. Not subroutine.
0121	TFRE?	;
0134	PUSH	;Routine to make a hole. Not a subroutine.
0136	MPUSH	;
0147	PDONE	;
0150	GET	;Routine to input tapes. Not a subroutine.
0157	DELAY	;
015D	SER?	;
0161	GMORE	;
016C	MSER?	;
0170	GDONE	;

LABELS

```
Add.   Label               Comments
0178   TTY     ;Routines for hex/ASCII dumps. Not subroutine
0188   MORTY   ;
018B   NOTTP   ;
01A1   NOADD   ;
01CE   RDOUT   ;SCRT subroutine outputs RD as 4 hex/ASCII.
01DA   CRLF    ;SCRT subroutine outputs an ASCII "CR" & "LF"
01E2   2HOUT   ;SCRT output subroutine,2 hex/ASCII from RF.1
01F2   1HOUT   ;SCRT output subroutine,1 hex/ASCII from RF.1
0200   S1DON   ;
0201   SOUT1   ;SEP subroutine outputs D to serial port 1.
0202   S1OK?   ;
020F   S2DON   ;
0210   SOUT2   ;SEP subroutine outputs D to serial port 2.
0214   T2OK?   ;
```

MONITOR REGISTER ASSIGNMENTS

```
Register Initial Value          Use
  R0         0000       ;Reset internally by 1802; initial
;program counter (RP)/ RP after "$" command/ DMA pointer.
  R1         0000       ;RP after interupt to restart mon-
;itor, reset before interrupt reenabled. To ensure correct
;interrupt action, R1 should NEVER have other uses.
  R2         07E0       ;Stack pointer for SCRT addresses.
  R3         0007       ;RP for most programs.
  R4         0101       ;RP for SEP routine CALL for SCRT.
  R5         0113       ;RP for SEP routine RETURN for SCRT.
  R6         ----       ;Scratchpad for CALL and RETURN.
  R7         07F0       ;RX for most programs. 07F0 is a
;free location for I/O and R7 should point here when not
;in use. The area from 07E1 to 07FF is available as a stack
;for data, but only 07F0 is used by the monitor.
  R8         00DC       ;RP for SEP routine SRX4.
  R9         ----       ;Scratchpad
  RA         0201       ;RP for SEP routine SOUT1.
  RB         00E3       ;RP for SEP routine HXOUT.
  RC         --00       ;Scratchpad, RC.0 is used as a flag.
  RD         ----       ;Basically scratchpad, but since RD
;is output to the led display, it is used by many routines
;for addresses or data. The display should be updated each
;time RD is modified.
  RE         ----       ;Same as RD.
  RF         ----       ;Scratchpad.
```

MONITOR PROGRAM

```
Add.  Code    Label   Mnemonic  Comments
0000  F8 00   START   LDI 00    ;Load 0007 in R3 for use as
0002  B3              PHI 3     ;the program counter (RP).
0003  F8 07           LDI 07    ;
0005  A3              PLO 3     ;
0006  D3              SEP 3     ;Make R3 = RP
0007  7A              REQ       ;If monitor restarted from
0008  F8 00           LDI 00    ;jump or interupt, Q flag
000A  A1              PLO 1     ;may need to be reset.
000B  B1              PHI 1     ;Initialize the registers
000C  B8              PHI 8     ;as required.
000D  AC              PLO C     ;
000E  BB              PHI B     ;0000  R1
000F  F8 01           LDI 01    ;07E0  R2
0011  A4              PLO 4     ;0101  R4
0012  B4              PHI 4     ;0113  R5
0013  B5              PHI 5     ;07F0  R7
0014  AA              PLO A     ;00DC  R8
0015  F8 02           LDI 02    ;0201  RA
0017  BA              PHI A     ;00E3  RB
0018  F8 E0           LDI E0    ;  00  RC.0
001A  A2              PLO 2     ;
001B  F8 DC           LDI DC    ;
001D  A8              PLO 8     ;
001E  F8 E3           LDI E3    ;
0020  AB              PLO B     ;
0021  F8 F0           LDI F0    ;
0023  A7              PLO 7     ;
0024  F8 07           LDI 07    ;
0026  B2              PHI 2     ;
0027  B7              PHI 7     ;
0028  F8 13           LDI 13    ;
002A  A5              PLO 5     ;
002B  E3              SEX 3     ;Make R3 = RX. This enables
002C  70 73           RET 73    ;the RET to load the immed-
002E  C4              NOP       ;iate byte and enable the
002F  C4              NOP       ;interrupt. R3=RP, R7=RX
```

MONITOR PROGRAM

```
Add.  Code       Label  Mnemonic      Comments
0030  39 33      CHECK  BNQ SIN     ;Jmp 1 if Q=0,else return
0032  D5                RETURN      ;to calling program
0033  D4 00F4    SIN    SUB INPUT   ;Get input from device
0036  9F         DIN    GHI F       ;Input into D
0037  FF 20             SMI 20      ;Subtract 20, 0 if "space"
0039  3A 3E             BNZ S1      ;Jmp if not
003B  1C                INC C       ;Make RC.0 > 0, data to RE
003C  30 30             BR CHECK    ;Back for more input
003E  FF 01      S1     SMI 01      ;Subtract 01, 0 if "!"
0040  32 C0             BZ MOD      ;Jmp to modify memory if so
0042  FF 03             SMI 03      ;Subtract 03, 0 if "$"
0044  3A 4B             BNZ S1B     ;Jmp if not
0046  8D                GLO D       ;Put RD into R0, and begin
0047  A0                PLO 0       ;execution of a new program
0048  9D                GHI D       ;with R0 as the PC. The PC
0049  B0                PHI 0       ;should be changed back to
004A  D0                SEP 0       ;R3 by the new program.
004B  FF 1B      S1B    SMI 1B      ;Subtract 1B, 0 if "?",
004D  3B 74             BM NUM      ;If neg, was 0 to 9, input it
004F  32 A5             BZ LOOK     ;Jmp to look at memory if "?"
0051  FF 08             SMI 08      ;Subtract 08, 0 if "G"
0053  3B 70             BM ATOF     ;If neg, was A to F, input it
0055  C2 0150           LBZ GET     ;Jmp to GET if "G"
0058  FF 09             SMI 09      ;Subtract 09, 0 if "P"
005A  C2 0134           LBZ PUSH    ;Jmp to PUSH if "P"
005D  FF 02             SMI 02      ;Subtract 02, 0 if "R"
005F  C6                LSNZ        ;Skip 2 if not
0060  D5                RETURN      ;Return to the program
0061  00                IDL         ;which called the monitor
0062  FF 02             SMI 02      ;Subtract 02, 0 if "T"
0064  C2 0178           LBZ TTY     ;Jmp to TTY routines if "T"
0067  FF 03             SMI 03      ;Subtract 03, 0 if "W"
0069  C2 0120           LBZ WRITE   ;Jmp to write if "W"
006C  D1                SEP 1       ;If no command recognized,
006D  00                IDL         ;there was an error, so
006E  00                IDL         ;restart monitor.
006F  00                IDL         ;
```

MONITOR PROGRAM

```
Add.  Code      Label Mnemonic          Comments
0070  F8 09     ATOF  LDI 09      ;Add 09 to the ASCII from
0072  F4              ADD         ;A to F to remove offset
0073  57              STR 7       ;Restore it like 0 to 9
0074  F0        NUM   LDX         ;Here if it was 0 to 9
0075  FE              SHL         ;Shift left by 4 to remove
0076  FE              SHL         ;last of offset. There is
0077  FE              SHL         ;now a hex number in D4 to D7
0078  FE              SHL         ;of the D register.
0079  57              STR 7       ;Save it
007A  F8 04           LDI 04      ;Load R9.0 with the number of
007C  A9              PLO 9       ;shifts to be done.
007D  8C        MOSH  GLO C       ;If RC.0=0, shift hex to RD
007E  3A 8D           BNZ LRE     ;Else jmp to load RE
0080  8D              GLO D       ;This section of code shifts
0081  FE              SHL         ;RD left by one in a 16 bit
0082  AD              PLO D       ;shift. This leaves a 0 in
0083  9D              GHI D       ;the lsb.
0084  7E              SHLC        ;
0085  BD              PHI D       ;Shift done here
0086  F0              LDX         ;Get the hex character
0087  FE              SHL         ;Shift the msb into DF
0088  3B 98           BNF SHOV?   ;If 0, check if shifting done
008A  1D              INC D       ;Else inc RD, ie set lsb to 1
008B  30 98           BR SHOV?    ;Check if shifting done
008D  8E        LRE   GLO E       ;This is the same as 0080 to
008E  FE              SHL         ;008A, but the hex is shifted
008F  AE              PLO E       ;into RE because RC.0 > 0.
0090  9E              GHI E       ;
0091  7E              SHLC        ;
0092  BE              PHI E       ;
0093  F0              LDX         ;
0094  FE              SHL         ;
0095  3B 98           BNF SHOV?   ;
0097  1E              INC RE      ;
0098  57        SHOV? STR 7       ;Save the shifted hex
0099  29              DEC R9      ;Dec the shift counter
009A  89              GLO 9       ;Get the counter
009B  3A 7D           BNZ MOSH    ;Jmp for more if not 0
009D  D4 01 AB        SUB DISPY   ;If done, display result,
00A0  30 30           BR CHECK    ;then back for more.
00A1  00              IDL         ;
00A2  00              IDL         ;
```

MONITOR PROGRAM

```
Add.  Code     Label  Mnemonic          Comments
00A4  1D       LOGAN  INC D       ;Point to next mem. location.
00A5  0D       LOOK   LDN D       ;This is the beginning of the
00A6  AE              PLO E       ;routine to examine memory.
00A7  F8 00           LDI 00      ;Put the data in RE.0 and
00A9  BE              PHI E       ;blank RE.1.
00AA  D4 01AB         SUB DISPY   ;Display M(RD) in RE.0.
00AD  C5              LSNQ        ;Skp 2 if Q=0, else since one
00AE  D5              RETURN      ;location was displayed,
00AF  00              IDL         ;return to calling program.
00B0  D4 00F4         SUB INPUT   ;Go and get an input,
00B3  9F              GHI F       ;put it into D.
00B4  FF 49           SMI 49      ;Subtract 49, 0 if "I".
00B6  32 A4           BZ LOGAN    ;If so, jump to look again.
00B8  F8 00    RENM   LDI 00      ;If not, reset RC.0 (not
00BA  AC              PLO C       ;always needed), and jump to
00BB  30 36           BR DIN      ;monitor to check last input.
00BD  00              IDL         ;
00BE  00              IDL         ;MOD is the routine to modify
00BF  00              IDL         ;a memory location. The piece
00C0  8E       MOD    GLO E       ;of data in RE.0 is put into
00C1  5D              STR D       ;M(RD).
00C2  F8 00           LDI 00      ;Blank RE.1.
00C4  BE              PHI E       ;
00C5  D4 01AB         SUB DISPY   ;Display that.
00C8  C5              LSNQ        ;Skp 2 if Q=0, else since one
00C9  D5              RETURN      ;location was modified,
00CA  00              IDL         ;return to calling program.
00CB  D4 00FA         SUB INPUT   ;Go and get an input,
00CE  9F              GHI F       ;put it into D.
00CF  FF 21           SMI 21      ;Subtract 21, 0 if "!".
00D1  3A B8           BNZ RENM    ;If not, back to monitor,
00D3  1D              INC D       ;else point to  next location
00D4  D4 01AB         SUB DISPY   ;and display that address.
00D7  30 30           BR CHECK    ;Go back to the monitor and
00D9  00              IDL         ;enter data for that location
00DA  00              IDL         ;Note that data goes to RE.

00DB  D3       SOVER  SEP 3       ;Return to calling program.
00DC  F6       SRX4   SHR         ;This is an SEP subroutine
00DD  F6              SHR         ;which shifts the D register
00DE  F6              SHR         ;right by four places. It
00DF  F6              SHR         ;puts 0's into D4 to D7.
00E0  30 30           BR SOVER    ;Over and out
```

MONITOR PROGRAM

```
Add.  Code     Label  Mnemonic          Comments
00E2  D3       HOVER  SEP 3       ;Return to calling program
00E3  FA 0F    HXOUT  ANI 0F      ;This is an SEP subroutine
00E5  FB 0F           XRI 0F      ;which outputs a single hex
00E7  57              STR 7       ;digit to the correct place
00E8  89              GLO 9       ;in the led display. Enter
00E9  FC 10           ADI 10      ;with the digit in D0 to D3
00EB  A9              PLO 9       ;of D, and the digit # minus
00EC  F1              OR          ;one in D4 to D7 of R9.0, so
00ED  57              STR 7       ;F0 is leftmost digit, and
00EE  63              OUT DISP    ;60 is the rightmost digit.
00EF  27              DEC 7       ;This digit # is upped by 10.
00F0  30 E2           BR HOVER    ;Over and out. Note that this
00F2  00              IDL         ;routine depends heavily on
00F3  00              IDL         ;the display hardware.
;This is the input subroutine. It loops until it finds an
;input from the ASCII keyboard or serial interface 1.
00F4  35 F9    INPUT  B2 KB?      ;Jmp if no serial input
00F6  69              INP SER1    ;Else input it
00F7  30 FC           BR GOTIT    ;Jmp a bit
00F9  3F F4    KB?    BN4 INPUT   ;Loop if no keyboard either.
00FB  6B              INP ASKEY   ;Else input it
00FC  BF       GOTIT  PHI F       ;Salt it away
00FD  D5              RETURN      ;Over and out
00FE  00              IDL         ;
00FF  00              IDL         ;
;This is the SUB subroutine. It is used to call subroutines
;via the RCA SCRT (Standard Call and Return) conventions.
0100  D3       EXITC  SEP 3       ;Go to the called subroutine.
0101  E2       SUB    SEX 2       ;R2 points to the return
0102  96              GHI 6       ;address stack.
0103  73              STXD        ;Push R6 onto stack and
0104  86              GLO 6       ;leave stack at a free
0105  73              STXD        ;location.
0106  93              GHI 3       ;Copy old RP (R3) into R6
0107  B6              PHI 6       ;to save it.
0108  83              GLO 3       ;R6 now points to two byte
0109  A6              PLO 6       ;inline address in calling
010A  46              LDA 6       ;program. Get it and put it
010B  B3              GHI 6       ;into R3 as it is the start
010C  46              LDA 6       ;of the called routine.
010D  A3              PLO 3       ;
010E  E7              SEX 7       ;Put R7 back as RX as R2 only
010F  30 00           BR EXITC    ;points to the subroutine
0111  C0              IDL         ;address stack.
```

MONITOR PROGRAM

This is the RETURN subroutine. It is used to return from a subroutine called by standard RCA SCRT techniques. This and the above routines are not used for SEP subroutines.

```
Add.  Code     Label  Mnemonic          Comments
0112  D3       EXITR  D3           ;Return to calling program.
0113  96       RETRN  GHI 6        ;Copy R6 into R3 so it will
0114  B3              PHI 3        ;contain the return program
0115  86              GLO 6        ;counter.
0116  A3              PLO 3        ;
0117  E2              SEX 2        ;R2 points to the return
0118  60              IRX          ;address stack. Decrement
0119  72              LDXA         ;it from the free location so
011A  A6              PLO 6        ;it points to the previously
011B  F0              LDX          ;stored value of R6 and put
011C  B6              PHI 6        ;it back into R6.
011D  E7              SEX 7        ;Put R7 back as RX.
011E  30 12           BR EXITR     ;Over and out

;This is the "W" (Write) routine. It is used to dump
;memory locations via serial output port 1. The data is
;sent out as a stream of 8 bit bytes, but final format
;depends on the UART control switches. It is not converted
;to hex or ASCII, so this is the routine for writing to
;cassette. The data from M(RD) to M(RE) is sent out.

0120  ED       WRITE  SEX D        ;RD is used for output (RX).
0121  36 21    TFRE?  B3 TFRE?     ;Loop 'till transmitter free.
0123  61              OUT SER1     ;Output byte and advance RD.
0124  E7              SEX 7        ;R7 = RX for display.
0125  D4 01AB         SUB DISPY    ;Display advanced address.
0128  D4 01C4         SUB SRERD    ;Sub for 16 bit RE-RD and
012B  33 20           BGE WRITE    ;back for more if RE>=RD.
012D  C5              LSNQ         ;Skp 2 if Q=0, else this was
012E  D5              RETURN       ;called as a subroutine, so
012F  00              IDL          ;return to calling program.
0130  D1              SEP 1        ;Done, so back to monitor.
0131  00              IDL          ;
0132  00              IDL          ;
0133  00              IDL          ;
```

MONITOR PROGRAM

```
;This is the "P" (Push) routine. All the data from M(RD) to
;M(RE) is pushed down by one location. This is useful for
;inserting an instruction, but all jumps must be corrected.
Add.  Code     Label Mnemonic         Comments
0134  4D       PUSH  LDA D      ;Get the first byte & advance
0135  B9             PHI 9      ;Save it.
0136  0D       MPUSH LDN D      ;Get the second byte.
0137  A9             PLO 9      ;Save that too.
0138  99             GHI 9      ;Get the first one again.
0139  5D             STR D      ;Put it in new location.
013A  89             GLO 9      ;Get the second one again.
013B  B9             PHI 9      ;Save it as new first byte.
013C  D4 01AB        SUB DISPY  ;Display advanced address.
013F  D4 01C4        SUB SRERD  ;Sub for 16 bit RE-RD.
0142  3B 47          BL PDONE   ;Jump ahead if done
0144  1D             INC D      ;Else increment RD
0145  30 36          BR MPUSH   ;and go back for more.
0147  C5       PDONE LSNQ       ;Skp 2 if Q=0, else this was
0148  D5             RETURN     ;called as a subroutine, so
0149  00             IDL        ;return to calling program.
014A  D1             SEP 1      ;Done, so back to monitor.
014B  00             IDL        ;IDL from 014B to 014F inc.
;This is the "G" (Get) routine. It is used to input a
;string of data bytes into memory from M(RD) to M(RE) from
;serial port 1. Parity depends on the front panel switches.
;This is the routine for loading a cassette or paper tape.
0150  69       GET   INP SER1   ;Input to clear UART flags.
0151  F8 FF          LDI FF     ;Load FFFF into R9 for use as
0153  B9             PHI 9      ;a timing constant. A delay
0154  F8 FF          LDI FF     ;is used so the cassette
0156  A9             PLO 9      ;interface can stabilze.
0157  29       DELAY DEC 9      ;Decrement timing constant.
0158  B9             GHI 9      ;Get the high 8 bits.
0159  C4             NOP        ;Extra delay.
015A  C4             NOP        ;
015B  3A 57          BNZ DELAY  ;Loop until R9.1 = 00.
015D  35 5D    SER?  B2 SER?    ;Tiny loop until data appears
015F  34 50          B1 GET     ;Start again if error.
0161  69       GMORE INP SER1   ;Get some data (finally!!)
0162  5D             STR D      ;and store it.
0163  1D             INC D      ;Increment memory pointer.
0164  D4 01AB        SUB DISPY  ;Display pointer.
0167  D4 01C4        SUB SRERD  ;Sub for 16 bit RE - RD
016A  3B 70          BL GDONE   ;and jump ahead if done.
016C  35 6C    MSER? B2 MSER?   ;Wait for more serial.
016E  3C 61          BN1 GMORE  ;Back for more if no error.
0170  C5       GDONE LSNQ       ;Skp 2 if Q=0, else this was
0171  D5             RETURN     ;called as a subroutine, so
0172  00             IDL        ;return to calling program.
0173  D1             SEP 1      ;All done, back to monitor
```

MONITOR PROGRAM

;These are the two teletype routines, "TT" and "TP".
;TT takes sequential 8 bit memory locations and outputs
;them as two hex ASCII characters - so 11000101 would
;be sent out as C5. Each pair of characters is followed by
;a line feed/carriage return. TP is similar except that the
;starting address and every nn00 address is also output.
;These routines are used for teletype listings. The memory
;from M(RD) to M(RE) is output.

```
Add.  Code     Label  Mnemonic       Comments
0174  00              IDL          ;
0175  00              IDL          ;
0176  00              IDL          ;
0177  00              IDL          ;
0178  D4 00F4  TTY    SUB INPUT    ;Get second letter of inst-
017B  9F              GHI F        ;ruction and put it into D.
017C  FF 50           SMI 50       ;Subtract 50, 0 if "P", put
017E  BC              PHI C        ;result in RC.1 for flag use.
017F  D4 01DA         SUB CRLF     ;Output a CR and LF.
0182  9C              GHI C        ;Get flag and
0183  3A 8B           BNZ NOTTP    ;jump a bit if not TP.
0185  D4 01CE         SUB RDOUT    ;Output RD to serial port 1.
0188  D4 01DA  MORTY  SUB CRLF     ;Output a CR and LF.
018B  4D       NOTTP  LDA D        ;Get the data, put it in RF.1
018C  BF              PHI F        ;and advance the pointer.
018D  D4 01E2         SUB 2HOUT    ;Sub to output two characters
0190  D4 01AB         SUB DISPY    ;Display advanced pointer.
0193  9C              GHI C        ;Get flag and
0194  3A A1           BNZ NOADD    ;don't output add. if not TP.
0196  8D              GLO D        ;Get low 8 bits of pointer,
0197  FA 0F           ANI 0F       ;mask in lower 4 bits only.
0199  3A A1           BNZ NOADD    ;Jump if address not = nn00.
019B  D4 01DA         SUB CRLF     ;Output a CR and LF.
019E  D4 01CE         SUB RDOUT    ;Output RD.
01A1  D4 01C4  NOADD  SUB SRERD    ;Sub for 16 bit RE - RD.
01A4  33 88           BGE MORTY    ;Go back for more if not done
01A6  C5              LSNQ         ;SKp 2 if Q=0, else this was
01A7  D5              RETURN       ;called as a subroutine, so
01A8  00              IDL          ;retrun to calling program.
01A9  D1              SEP 1        ;All done, back to monitor
01AA  00              IDL          ;
```

MONITOR PROGRAM

```
;This is the display subroutine. When it is called the
;current contents of RD and RE are output to an eight digit
;hexadecimal led display. The display is self scanning and
;refresh by the program is not required. The display should
;be updated whenever RD or RE is changed.

Add.  Code    Label Mnemonic        Comments
01AB  F8 F0   DISPY LDI FO     ;Preset R9.0 for use by HXOUT
01AD  A9            PLO 9      ;
01AE  9D            GHI D      ;Get the high 8 bits of RD.
01AF  D8            SEP 8      ;Sub to shift right by four.
01B0  DB            SEP B      ;Sub to output first (left-
01B1  9D            GHI D      ;most) digit, get RD.1 again.
01B2  DB            SEP B      ;Output second digit.
01B3  8D            GLO D      ;Get the low 8 bits of RD.
01B4  D8            SEP 8      ;Shift right by four.
01B5  DB            SEP B      ;Output third digit.
01B6  8D            GLO 8      ;Get RD.0 again.
01B7  DB            SEP B      ;Output fourth digit.
01B8  9E            GHI E      ;Get the high 8 bits of RE.
01B9  D8            SEP 8      ;Shift right by four.
01BA  DB            SEP B      ;Output fifth digit.
01BB  9E            GHI E      ;Get RE.1 again.
01BC  DB            SEP B      ;Output sixth digit.
01BD  8E            GLO E      ;Get the low 8 bits of RE.
01BE  D8            SEP 8      ;Shift right by four.
01BF  DB            SEP B      ;Output seventh digit.
01C0  8E            GLO E      ;Get RE.0 again.
01C1  DB            SEP B      ;Output eighth digit.
01C2  D5            RETURN     ;Return to calling program.
01C3  00            IDL        ;

;This routine performs a 16 bit subtraction, RE - RD.
;Only the sign, in DF, is returned.

01C4  8D      SRERD GLO D      ;Get low 8 bits of RD and put
01C5  57            STR 7      ;it in a free location.
01C6  8E            GLO E      ;Get low 8 bits of RE.
01C7  F7            SM         ;D-M(RX) = RE.0-RD.0.
01C8  9D            GHI D      ;Get high 8 bits of RD and
01C9  57            STR 7      ;put it in a free location.
01CA  9E            GHI E      ;Get high 8 bits of RE.
01CB  77            SMB        ;RE.1 - RD.1 - Borrow.
01CC  D5            RETURN     ;Return to calling program.
01CD  00            IDL        ;
```

MONITOR PROGRAM

```
;This is a routine to output RD as 4 ASCII characters.

Add.   Code     Label Mnemonic          Comments
01CE   9D       RDOUT GHI D       ;Get the high 8 bits of RD
01CF   BF             PHI F       ;and put them in RF.1
01D0   D4 01E2        SUB 2HOUT   ;Sub to output two characters
01D3   8D             GLO D       ;Get the low 8 bits of RD
01D4   BF             PHI F       ;and  put them in RF.1.
01D5   D4 01E2        SUB HOUT    ;Sub to output two characters
01D8   D5             RETURN      ;Return to calling program.
01D9   00             IDL         ;

;This routine outputs a "CARRIAGE RETURN" & "LINEFEED".

01DA   F8 0D    CRLF  LDI 0D      ;Load ASCII for "CR".
01DC   DA             SEP A       ;Sub to output D.
01DD   F8 0A          LDI 0A      ;Load ASCII for "LF".
01DF   DA             SEP A       ;Sub to output D.
01E0   D5             RETURN      ;Return to calling program.
01E1   00             IDL         ;

;This routine outputs two ASCII hexadecimal characters from
;an 8 bit number in RF.1.

01E2   9F       2HOUT GHI F       ;Get data passed from program
01E3   B9             PHI 9       ;Hide it somewhere.
01E4   D8             SEP 8       ;Sub to shift right by four.
01E5   BF             PHI F       ;Put it in RF.1.
01E6   D4 01F2        SUB 1HOUT   ;Sub to output one character.
01E9   99             GHI 9       ;Get original data again and
01EA   FA 0F          ANI 0F      ;mask in lower four bits.
01EC   BF             PHI F       ;Put it in RF.1.
01ED   D4 01F2        SUB 1HOUT   ;Sub to output one character.
01F0   D5             RETURN      ;Return to calling program.
01F1   00             IDL         ;

;This routine outputs one ASCII hex character from the
;lower four bits of RF.1.

01F2   9F       1HOUT GHI F       ;Get data passed from program
01F3   FF 0A          SMI 0A      ;Subtract 0A, <0 if 0 to 9.
01F5   9F             GHI F       ;Get data again, but leave DF
01F6   C7             LSNF        ;Skp 2 if neg, was 0 to 9.
01F7   FC 07          ADI 07      ;Add partial offset if A to F
01F9   FC 30          ADI         ;Add offset for 0 to F.
01FB   DA             SEP A       ;Sub to output D.
01FC   D5             RETURN      ;Return to calling program.
```

MONITOR PROGRAM

;This is the subroutine to output the contents of D to
;serial port 1. It does not modify any registers, but
;does require RX to point to a free memory location.

```
Add.  Code    Label  Mnemonic         Comments
01FD  00             IDL         ;
01FE  00             IDL         ;Note that RX must be R7
01FF  00             IDL         ;on entry.
0200  D3      S1DON  SEP 3       ;Return to calling program.
0201  57      SOUT1  STR 7       ;Put D into memory.
0202  36 02   S1OK?  B3 S1OK?    ;Loop 'till transmitter free.
0204  61             OUT SER1    ;Output data.
0205  27             DEC 7       ;Restore RX value.
0206  30 00          BR S1DON    ;Done, so get out.
0208  00             IDL         ;IDL's (00) until 020F.
```

;This is the routine to output the contents of D to serial
;port 2. If RA is set to 0210 this routine will be used
;instead of SOUT1 whenever an SEP A (DA) is executed. This
;port is harder to use as the status must be read in with
;an input instruction and cannot be checked with the flags.
;It is more flexible as the control register is set with
;software rather than switches, but because of this the
;control register must be set before calling this program.
;This routine is not used by the monitor. RX must be R7 and
;point to 07F0 on entry. Location 07FF must be free.

```
020F  D3      S2DON  SEP 3       ;Return to calling program.
0210  57      SOUT2  STR 7       ;Save D in 07F0.
0211  F8 FF          LDI FF      ;Point to a free location.
0213  A7             PLO 7       ;
0214  6F      T2OK?  INP PIP1    ;Input parallel port 1.
0215  FE             SHL         ;Put transmitter status bit
0216  3B 14          BNF T2OK?   ;into DF, loop until ready.
0218  F8 F0          LDI F0      ;Point to stored D in normal
021A  A7             PLO 7       ;I/O location and
021B  66             OUT SER2    ;output data.
021C  27             DEC 7       ;Restore RX value and
021D  30 0F          BR S2DON    ;get out.
021F  00             IDL         ;
```

APPENDIX D

MORSE DECODING PROGRAM

MORSE PROGRAM

```
;This is the Morse code decoding program. Program segments
;are more in logical than numerical order.

Add.  Code      Label Mnemonic          Comments
03DE  F8 00     RSTRT LDI 00       ;Program restarts here, RC.0
03DF  AC              PLO C        ;is a monitor flag, reset it.
03E0  F8 03     MORSE LDI 03       ;Program starts here. R3 is
03E2  B3              PHI 3        ;set for use as RP.
03E3  F8 E6           LDI E6       ;
03E5  A3              PLO 3        ;
03E6  D3              SEP 3        ;Make R3 be RP.
03E7  E3              SEX 3        ;Make R3 be RX as well. This
03E8  67              OUT CON2     ;will output "F0" to serial
03E9  F0              (data)       ;port 2 to set the UART.
03EA  E7              SEX 7        ;Make R7 be RX.
03EB  7A              REQ          ;Reset the Q flag.
03EC  D4 0033         SUB SIN      ;Call the monitor as a sub-
;routine. Enter a 4 hex digit intial speed in the lefthand
;display (typ. 0000 to 0300), then type "R" to return.
03EF  F8 10     RSTR2 LDI 10       ;Preset RE.1 and RE.0, the
03F1  AE              PLO E        ;mark and space thresholds
03F2  BE              PHI E        ;respectively, to "10".
03F3  D4 03C0         SUB MASC     ;Sub to zero histograms, Q=0.
03F6  D4 01DA   RSTR1 SUB CRLF     ;Output a "CR" & "LF".
03F9  F8 00           LDI 00       ;Reset various registers:
03FB  AC              PLO C        ;RC.0 time of keyup
03FC  BC              PHI C        ;RC.1 # of keydowns in letter
03FD  A9              PLO 9        ;R9.0 time of keydown
03FE  B9              PHI 9        ;R9.1 Morse shift register
03FF  7A              REQ          ;Reset the Q flag.
0400  9D        TIME  GHI D        ;This is the start of the
0401  BF              PHI F        ;basic timing loop. The
0402  8D              GLO D        ;timing constant is moved
0403  AF              PLO F        ;from RD to RF.
0404  8F        TIM1  GLO F        ;Check RF.0,
0405  32 0A           BZ TIM2      ;jump ahead if zero,
0407  2F        TIM3  DEC F        ;else decrement RF,
0408  30 04           BR TIM1      ;and jump back for more.
040A  9F        TIM2  GHI 9        ;Check RF.1, and jump back
040B  3A 07           BNZ TIM3     ;if not also zero,
040D  6F              INP PIP1     ;else input Morse port, and
040E  64              OUT POP2     ;output it to a one bit (D0)
040F  27              DEC 7        ;port to check sampling.
```

MORSE PROGRAM

```
Add.  Code      Label Mnemonic        Comments
0410  FA 01           ANI 01     ;Mask in lsb, 0=up, 1=down.
0412  32 2C           BZ UP      ;Jump if key up.
0414  8C              GLO 8      ;If keyup timer zero, then
0415  32 21           BZ STDN    ;key was down before, so jmp.
0417  FF 40           SMI 40     ;If timer = 40, timer has
0419  32 21           BZ STDN    ;reached limit, so jump.
041B  D4 0540         SUB UBIN   ;Else sub to bin key up time
041E  C1 0568         L3Q AUT?   ;If Q was set by BIN, then
;a histogram bin was full, so jump to AUT? for action.
0421  F8 00     STDN  LDI 00     ;Reset key up timer.
0423  AC              PLO C      ;
0424  89              GLO 9      ;Get key down timer,
0425  FF 40           SMI 40     ;subtract 40, zero if full,
0428  32 00           BZ TIME    ;so jump back to timer.
042A  19              INC 9      ;Otherwise increment key
042A  30 00           BR TIME    ;down counter and go to timer
042C  8C        UP    GLO C      ;Here if key up, get timer.
042D  3A 58           BNZ STUP   ;Jump if 0, was up before.
042F  9E              GHI E      ;Else get mark threshold.
0430  57              STR 7      ;Put it in free location.
0431  89              GLO 9      ;Get key down timer, find
0432  F7              SM         ;Key down - mark threshold.
0433  33 3D           BGE DASH   ;Jump if >= 0, it was a dash.
0435  99              GHI 9      ;Get Morse shift register.
0436  FE              SHL        ;Shift left, 0=dot into lsb.
0437  B9              PHI 9      ;Put it back.
0438  89              GLO 9      ;Get timer AGAIN!!!
0439  FF 08           SMI 08     ;Subt. 08, 0 if perfect dot.
043B  30 45           BR SPD?    ;Jump ahead to check speed.
043D  99        DASH  GHI 9      ;Get shift register.
043E  FE              SHL        ;Shift left, 0 into lsb.
043F  FC 01           ADI 01     ;Add 01, 1=dash into lsb.
0441  B9              PHI 9      ;Put it back.
0442  89              GLO 9      ;Get timer again.
0443  FF 18           SMI 18     ;Subt. 18, 0 if perfect dash.
;At this point, the D register is zero if the dot or dash
;was of a perfect length. DF=1 if the speed is too  fast
;(dot > 08 or dash > 18), and DF=0 if too slow.
0445  32 4B     SPD?  BZ UP1     ;Jump if D=0, no change.
0447  1D              INC D      ;Inc.(slower) timing constant
0448  CF              LSDF       ;Skip 2 if DF=1, speed was
0449  2D              DEC D      ;fast, else dec. by 2, effec-
044A  2D              DEC D      ;tively by 1, raise speed.
044B  D4 0545   UP1   SUB DBIN   ;Sub to bin key down.
044E  39 54           BNQ UP2    ;Jmp if Q=0, no bins full,
0450  7A              REQ        ;else reset Q,
0451  C0 0568         LBR AUT?   ;and jump for action.
```

MORSE PROGRAM

```
Add.  Code   Label Mnemonic       Comments
0454  9C     UP2   GHI C       ;Get counter for number of
0455  FC 01        ADI 01      ;Key downs in a letter incre-
0457  BC           PHI C       ;ment and restore it.
0458  F8 00  STUP  LDI 00      ;Reset key down timer.
045A  A9           PLO 9       ;
045B  8C           GLO C       ;Get key up timer,
045C  FF 40        SMI 40      ;subt. 40, 0 if timer full,
045E  32 79        BZ UP3      ;jump if so.
0460  9C           GHI 9       ;If RC.0=0, there are no
0461  32 69        BZ UP4      ;elements in letter, so jump.
0463  8E           GLO E       ;Get space threshold,and put
0464  57           STR 7       ;it in a free location.
0465  8C           GLO C       ;Get key up timer and find
0466  F7           SM          ;key up - space threshold.
0467  32 80        BZ LOUT     ;If 0, now output letter.
0469  8E     UP4   GLO E       ;Find inter-character space:
046A  F6           SHR         ;
046B  57           STR 7       ;ICS = 4*RE.0 - RE.0/2
046C  8E           GLO E       ;     = 3.5*RE.0
046D  FE           SHL         ;For perfect morse it should
046E  FE           SHL         ;be 3.0, but hand sent is
046F  F7           SM          ;better with this ratio.
0470  57           STR 7       ;Put ICS in free location.
0471  8C           GLO C       ;Get key up timer and find
0472  F7           SM          ;key up - ICS
0473  3A 78        BNZ UP5     ;Jump if not zero, no space.
0475  F8 20        LDI 20      ;Load ASCII "space".
0477  DA           SEP A       ;Output to serial port 1
0478  1C     UP5   INC C       ;Increment key up timer.
0479  3F 00  UP3   BN4 TIME    ;Jump to timer if no key-
047B  30 B8        BR KCHK     ;board input, else go and
047D  00           IDL         ;see what it is.
047E  00           IDL         ;Following section outputs
047F  00           IDL         ;a letter.
0480  99     LOUT  GHI 9       ;Get Morse shift register,
0481  57           STR 7       ;put it in a free location,
0482  62           OUT POP1    ;output it to 8 leds for a
0483  27           DEC 7       ;visual check of dots/dashes.
0484  9C           GHI C       ;Check if 6 key downs in
0485  FF 06        SMI 06      ;letter, jump if not for
0487  3A 9A        BNZ N06     ;standard decoding.
```

MORSE PROGRAM

```
;The next segment of code is for Morse characters with
;exactly six elements as some require different decoding.

Add.  Code      Label  Mnemonic       Comments
0489  99               GHI 9       ;Get Morse shift register and
048A  FF 4C            SMI 4C      ;subtract Morse "?".
048C  3A 92            BNZ NO?     ;Jump if not that,
048E  F8 3F            LDI 3F      ;else load ASCII "?" and
0490  30 A8            BR MOUT     ;jump to output it.
0492  FF 2C     NO?    SMI 2C      ;Subtract further offset,
0494  3A A0            BNZ OKLET   ;jump if not 0,
0496  F8 3A            LDI 3A      ;else load ASCII ":" and
0498  30 A8            BR MOUT     ;jump to output it.

;The remaining 6 element characters are handled normally.
;At this point the D register holds # of key downs - six.

049A  3B A0     NO6    BL OKLET    ;If less than six, ok, but if
049C  F8 2A            LDI 2A      ;not load ASCII "*" for over-
049E  30 A8            BR MOUT     ;run and jump to output it.
04A0  99        OKLET  GHI 9       ;Get Morse shift register,
04A1  FA 3F            ANI 3F      ;mask in lower 6 bits, put
04A3  AF               PLO F       ;it in RF.0 as low address.
04A4  F8 05            LDI 05      ;Put high address in RF.1,
04A6  BF               PHI F       ;point to Morse/ASCII table.
04A7  0F               LDN F       ;Get ASCII
04A8  DA        MOUT   SEP A       ;and output it.
04A9  1C               INC C       ;Increment key up timer.
04AA  27               DEC 7       ;Point to free memory (07EF).
04AB  8C               GLO C       ;Get key up timer,
04AC  57               STR 7       ;save it in memory, and point
04AD  60               IRX         ;back to I/O location (07F0).
04AE  C0 0600          LBR THRM    ;Jump to find new thresholds.
04B1  00               IDL         ;
04B2  00               IDL         ;
04B3  00               IDL         ;
04B4  00               IDL         ;
04B5  00               IDL         ;
04B6  00               IDL         ;
04B7  00               IDL         ;
```

MORSE PROGRAM

```
;The next segment of code checks for input from the ASCII
;Keyboard and acts accordingly.

Add.  Code     Label  Mnemonic          Comments
04B8  3F B8    KCHK   BN4 KCHK    ;Loop until keyboard input.
04BA  6B              INP ASKEY   ;Input keyboard into D, M(RX)
04BB  FF 2F           SMI 2F      ;Subtract ASCII "/".
04BD  C2 03DD         LBZ RSTRT   ;Complete restart if "/"
04C0  FF 19           SMI 19      ;Subtract again, 0 if "H".
04C2  C2 0250         LBZ HIST    ;Output histogram if "H".
04C5  FF 03           SMI 03      ;Subtract again, 0 if "K".
04C7  3A D7           BNZ NOKO    ;Jump if not, else point
04C9  F8 10           LDI 10      ;register A to output
04CB  AA              PLO A       ;routine for serial port 2.
04CC  F8 1B           LDI 1B      ;Output "ESC" & "ETB" to
04CE  DA              SEP A       ;serial port 2. This sends
04CF  F8 17           LDI 17      ;a copy command to the
04D1  DA              SEP A       ;Tektronix Graphics Terminal.
04D2  F8 01           LDI 01      ;Point register A back to the
04D4  AA              PLO A       ;routine for serial port 1.
04D5  30 B8           BR KCHK     ;Back for more input.
04D7  FF 02    NOKO   SMI 02      ;Subtract again, 0 if "M".
04D9  C2 03F6         LBZ RSTR1   ;If so partial restart.
04DC  FF 01           SMI 01      ;Subtract again, 0 if "N".
04DE  32 F6           BZ SAUT     ;If so, jump to cancel auto.
04E0  FF 0A           SMI 0A      ;Subtract again, 0 if "X".
04E2  7A              REQ         ;Reset Q (will clear hist.)
04E3  32 EA           BZ GOMAS    ;If "X", go clear histograms.
04E5  FF 01           SMI 01      ;Subtract again, 0 if "Y".
04E7  3A F1           BNZ Z?      ;Jump if not, else
04E9  7B              SEQ         ;set Q (will histogram/2)
04EA  D4 03C0  GOMAS  SUB MASG    ;and divide hist. data by 2.
04ED  7A              REQ         ;Reset Q (X & Y return here)
04EE  C0 03F6         BR RSTR1    ;and do partial restart.
04F1  FF 01    Z?     SMI 01      ;Subtract AGAIN!, 0 if "Z".
04F3  CA 0000         LBNZ START;If not,, invalid, to monitor.
04F6  27       SAUT   DEC 7       ;Here if "Z" or "N", point to
04F7  27              DEC 7       ;location for auto command,
04F8  6B              INP ASKEY   ;(07EE) and put input there.
04F9  60              IRX         ;"Z" for auto, "N" nonauto.
04FA  60              IRX         ;Point back to I/O location.
04FB  30 00           BR TIME     ;Go back to timing loop.
04FD  00              IDL         ;
04FE  00              IDL         ;
04FF  00              IDL         ;
```

MORSE PROGRAM

```
;This is the lookup table to convert Morse to ASCII.
;A register is preset to  0000 0001  and the elements are
;shifted into the lsb, 0 for dot and 1 for dash. The
;leading 1 shows the beginning of the character. The letter
;"L" is ".-.."and would appear as  0001 0100. The upper two
;bits are set to 00. This forms the lower eight bits of the
;address, and 05 is the offset for the high eight bits.
;This scheme is memory efficient, but causes characters
;with six elements to show up anywhere in the table.
;This is only a problem for "?" and ":", so these are dealt
;with in software. "@" is used for an invalid character,
;and "<" for the Morse error "......".
```

Add.	Data	Character	Add.	Data	Character
0500	3C	<	0520	35	5 .....
0501	40	@	0521	34	4 ....-
0502	45	E .	0522	40	@
0503	54	T -	0523	33	3 ...--
0504	49	I ..	0524	40	@
0505	41	A .-	0525	40	@
0506	4E	N -.	0526	40	@
0507	4D	M --	0527	32	2 ..---
0508	53	S ...	0528	40	@
0509	55	U ..-	0529	40	@
050A	52	R .-.	052A	3B	; -.-.-.
050B	57	W .--	052B	40	@
050C	44	D -..	052C	40	@
050D	4B	K -.-	052D	5B	[-.--.-
050E	47	G --.	052E	40	@
050F	4F	O ---	052F	31	1 .----
0510	48	H	0530	36	6 -....
0511	56	V ...-	0531	5F	- -...-
0512	46	F ..-.	0532	2F	/ -..-.
0513	40	@	0533	2C	, --..--
0514	4C	L .-..	0534	40	@
0515	2E	. .-.-.-	0535	40	@
0516	50	P .--.	0536	40	@
0517	4A	J .---	0537	40	@
0518	42	B -...	0538	37	7 --...
0519	58	X -..-	0539	40	@
052A	43	C -.-.	053A	40	@
051B	59	Y -.--	053B	40	@
051C	5A	Z --..	053C	38	8 ---..
051D	51	Q --.-	053D	40	@
051E	40	@	053E	39	9 ----.
051F	40	@	053F	30	0 -----

MORSE PROGRAM

```
;This is the subroutine to enter the mark and space times
;into their respective histograms. The raw space data is
;stored from 0580 to 05BF, the raw mark data from 05C0
;to 05FF.

Add.  Code     Label Mnemonic        Comments
0540  8C       UBIN  GLO C      ;Enter here to bin space,get
0541  FC 80          ADI 80     ;key up timer, add offset,
0543  30 4D          BR SP      ;and jump ahead.
0545  89       DBIN  GLO 9      ;Enter here for mark, get
0546  FF 40          SMI 40     ;timer, subtract 40, jump if
0548  33 5F          BGE BIGM   ;>=0, therefore overflow.
054A  89             GLO 9      ;If ok, get timer again, add
054B  FC C0          ADI C0     ;offset.
054D  AF       SP    PLO F      ;For mark or space, put data
054E  F8 05          LDI 05     ;plus offset into RF.0, and
0550  BF             PHI F      ;high address into RF.1.

;The value from the timer, e.g. 2C, is used to point to the
;2Cth bin of the appropriate histogram. The number in the
;2Cth bin indicates the number of times the particular
;timer reached exactly 2C before the key changed.

0551  0F             LDN F      ;Get the data from that bin.
0552  FF 3F          SMI 3F     ;Subt. 3F (63), 0 if full.
0554  7B             SEQ        ;Set Q in case it's full.
0555  32 5C          BZ BIND    ;If full, jump out with Q=1.
0557  7A             REQ        ;Not full, so reset Q,
0558  0F             LDN F      ;and get data again from bin.
0559  FC 01          ADI 01     ;Increment the data
055B  5F             STR F      ;and restore it.
055C  C0 0760  BIND  BR SMTH    ;Done, jmp to smooth routine.

;Program ends up here if key down timer overflowed. This
;usually means the overall speed is far too high.

055F  8D       BIGM  GLO D      ;The timing constant is
0560  FE             SHL        ;stored in RD, and this is
0561  AD             PLO D      ;shifted left by one in a 16
0562  9D             GHI D      ;bit shift. This multiplies
0563  7E             SHLC       ;the constant by 2 and halves
0564  BD             PHI D      ;the speed of the timer.
0565  C0 03EF        LBR RSTR2 ;Do a partial restart with
;the new slower speed and zeroed histograms.
```

MORSE PROGRAM

```
;The binning subroutine sets the Q flag if a bin is full.
;On return to the key up or key down routines the flag
;is sampled. If set, control passes to this routine which
;checks the auto histogram flag stored in 07EE. This
;is not a subroutine as it can return to different places.
;If entered from the key down section, Q is still set, but
;the key up section resets Q first. Q tells this section
;where to return if no histogram was required.

Add.  Code     Label Mnemonic          Comments
0568  27       AUT?  DEC 7      ;Point to location which
0569  27             DEC 7      ;holds auto flag.
056A  72             LDXA       ;Get it and point back to
056B  60             IRX        ;free location (07F0).
056C  FF 5A          SMI 5A     ;Subtract 5A, 0 if "Z".
056E  C2 0250        LBZ HIST   ;If so, jump for histogram.
0571  C9 0454        LBNQ UP2   ;If Q=0, back to key up,
0574  7A             REQ        ;else reset Q
0575  C0 0421        LBR STDN   ;and back to key down.
0578  00             IDL        ;IDL's to 057F inc.

;0580 to 05BF contains raw histogram data for spaces.
;05C0 to 05FF contains raw histogram data for marks.

;This is the subroutine to clear (Q=0) or divide by two
;(Q=1) the raw histogram data.

Add.  Code     Label Mnemonic          Comments
03C0  F8 05    MASC  LDI 05     ;Put high address pointer
03C2  BF             PHI F      ;into RF.1.
03C3  F8 80          LDI 80     ;Low address is start of raw
03C5  AF             PLO F      ;space data, marks follow.
03C6  8F       MAS?  GLO F      ;If RF.0=00, then done.
03C7  C6             LSNZ       ;Skip two if not done,
03C8  D5             RETURN     ;else return.
03C9  00             IDL        ;
03CA  C5             LSNQ       ;Skip two if Q=0 (clear),
03CB  0F             LDN F      ;else get data from bin and
03CC  F6             SHR        ;shift right to halve it.
03CD  CD             LSQ        ;Skip two if Q=1 (halve),
03CE  F8 00          LDI 00     ;else load 00 to clear.
03D0  5F             STR F      ;Put new data into bin
03D1  1F             INC F      ;and increment pointer.
03D2  30 C6          BR MAS?    ;Loop back for more.
03D4  00             IDL        ;IDL's to 03DC inc.
```

MORSE PROGRAM

```
;This is the smoothing subroutine. It is a continuation of
;the binning subroutine which returns from here. The raw
;histogram data from 0580-05BF, 05C0-05FF is smoothed by
;a simple algorithm and stored from 0680-06BF, 06C0-06FF.
;The Q flag is still set if a bin was full.

Add.  Code   Label  Mnemonic        Comments
0760  F8 05  SMTH   LDI 05      ;Load 0580 into RF, point to
0762  BF            PHI F       ;raw data, 0680 into R0,
0763  F8 06         LDI 06      ;space for smoothed data.
0765  B0            PHI 0       ;
0766  F8 80         LDI 80      ;
0768  A0            PLO 0       ;
0769  AF            PLO F       ;
076A  E0            SEX 0       ;R0 used for math, make it RX
076B  0F     SMTH1  LDN F       ;Get raw data(N)
076C  FE            SHL         ;multiply by 2,
076D  50            STR 0       ;store it in smooth space.
076E  2F            DEC F       ;Point to raw data(N-1).
076F  8F            GLO F       ;If RF=057F, out of data
0770  FF 7F         SMI 7F      ;storage area,
0772  32 7B         BZ BDAT     ;so jump ahead.
0774  FF 40         SMI 40      ;If RF=05BF, marks intruding
0776  32 7B         BZ BDAT     ;into spaces, so jump ahead.
0778  0F            LDN F       ;Get raw data(N-1) and add it
0779  F4            ADD         ;to 2*raw data(N).
077A  50            STR 0       ;Put result in smoothed area.
077B  1F     BDAT   INC F       ;Point to raw data(N+1).
077C  1F            INC F       ;
077D  8F            GLO F       ;If RF=0600, out of data,
077E  32 87         BZ BDAT2    ;so jump ahead.
0780  FF 80         LDI 80      ;If RF=0580, spaces intruding
0782  32 87         BZ BDAT2    ;into marks, so jump ahead.
0784  0F            LDN F       ;Get raw data(N+1) and add it
0785  F4            ADD         ;to value from line 0779
0786  50            STR 0       ;Put result in smoothed area.
0787  F0     BDAT2  LDX         ;Get result so far,
0788  FC 02         ADI 02      ;add two for rounding.
078A  F6            SHR         ;Divide by four
078B  F6            SHR         ;
078C  50            STR 0       ;and store result.

;The data was smoothed with the following formula:
; smooth(n) = (2*raw(n) + raw(n-1) + raw(n+1) + 2)/4
```

MORSE PROGRAM

;This is the continuation of the smoothing routine.

```
Add.  Code      Label Mnemonic          Comments
078D  60              IRX        ;Point to next smooth space.
078E  80              GLO 0      ;If R0 not 0700, not done,
078F  3A 6B           BNZ SMTH1  ;so jump back for more,
0791  E7              SEX 7      ;else restore R7 as RX.
0792  CD              LSQ        ;If Q=0, no bins full,
0793  D5              RETURN     ;so return, otherwise skip 2.
0794  00              IDL        ;
0795  D4 03C0         SUB MASC   ;Sub to MASC, Q=1, so data/2.
0798  D4 01DA         SUB CRLF   ;Output "CR" & "LF"
079B  F8 3E           LDI 3E     ;Load ASCII ">" and output it
079D  DA              SEP A      ;to show histogram change.
079E  D5              RETURN     ;Return with Q still set for
079F  00              IDL        ;auto histogram check.
```

;The next set of routines computes the mark and space
;thresholds from the smoothed data. The main routine begins
;at 0600, but the subroutines are presented first.
;They all rely on the position of the data within memory.
;This subroutine returns the maximum value of a mark or
;space histogram in RC.1 and the bin number (00 to 3F),
;not the memory address, in RC.0. If Q is set the leftmost
;maximum is returned and if Q is reset, the rightmost.
;These values are different if two bins equal the maximum.
;R9 contains the starting address of data and RF.0 contains
;the bin number of that start data as not all scans start
;from bin 00 when looking for the leftmost maximum.

```
Add.  Code      Label Mnemonic          Comments
0740  F8 00     FMAX  LDI 00     ;Reset bin number
0742  AC              PLO C      ;and maximum value.
0743  BC              PHI C      ;
0744  CD              LSQ        ;If right maximum set RF.0
0745  AF              PLO F      ;to 00, else leave it alone.
0746  C4              NOP        ;
0747  E9        MAXM  SEX 9      ;Make R9 be RX for math use.
0748  9C              GHI C      ;Get current maximum and sub-
0749  F5              SD         ;tract data pointed to by R9.
074A  3B 53           BL NCHG    ;If less, no change, so jump.
074C  C5              LSNQ       ;Skip two if rightmost max.,
;that means change on >=. If leftmost max, change on > only
074D  32 53           BZ NCHG    ;Jump if leftmost and equal.
074F  09              LDN 9      ;If a change, get new maximum
0750  BC              PHI C      ;and put it in RC.1.
0751  8F              GLO F      ;Get new bin number and put
0752  AC              PLO C      ;it in RC.0.
```

MORSE PROGRAM

```
;This is a continuation of the "FMAX" subroutine.

Add.  Code    Label Mnemonic         Comments
0753  1F      NCHG  INC F      ;Increment bin number
0754  19            INC 9      ;and memory pointer.
0755  89            GLO 9      ;If R9=0600, mark scan done,
0756  32 5C         BZ MAXD    ;jump ahead.
0758  FF C0         SMI C0     ;If R9=05BF, space scan done,
075A  3A 47         BNZ MAXM   ;else jump back for more.
075C  E7      MAXD  SEX 7      ;Restore R7 as RX
075D  D5            RETURN     ;and get out.
075E  00            IDL        ;This next subroutine takes
075F  00            IDL        ;the data passed from FMAX
;and sets it up for later use. Enter with Q=1 for marks.
0700  F8 00   ADD50 LDI 00     ;Reset RF.0
0702  AF            PLO F      ;
0703  9C            GHI C      ;Get maximum peak height.
0704  F6            SHR        ;divide by two
0705  BC            PHI C      ;and put it back.
0706  F8 80         LDI 80     ;Get space offset
0708  C5            LSNQ       ;skip 2 if spaces
0709  F8 C0         LDI C0     ;else get mark offset.
070B  57            STR 7      ;Put offset in 07F0.
070C  8C            GLO C      ;Get maximum bin number
070D  F4            ADD        ;add offset for low address
070E  A9            PLO 9      ;and put it in R9.0.
070F  D5            RETURN     ;

;This subroutine finds the first occurence of a bin
; <= (maximum height)/2 while searching left to right.
;Enter with start address of search in R9, (maximum)/2
;in RC.1, and 00 in RF.0. On exit RF.0 contains the number
;of bins from the peak to <= (peak)/2.

Add.  Code    Label Mnemonic         Comments
0710  00            IDL        ;IDL's to 0713 inc.
0714  E9      LTOR  SEX 9      ;Make R9 be RX
0715  9C      MOLR  GHI C      ;Get 1/2 peak and
0716  F7            SM         ;subtract data.
0717  33 22         BGE LROK   ;Out if bin found,
0719  19            INC 9      ;else increment pointer
071A  1F            INC F      ;and bin offset.
071B  89            GLO 9      ;If R9=0700
071C  32 22         BZ         ;get out, end of marks.
071E  FF C0         SMI C0     ;If R9=06C0, end of spaces,
0720  3A 15         BNZ MOLR   ;else jump back for more.
0722  E7      LROK  SEX 7      ;Restore R7 as RX.
0723  D5            RETURN     ;
0724  00            IDL        ;IDL's to 0727 inc.
```

MORSE PROGRAM

```
;This subroutine is similar to the one above except that
;it searches from right to left. Q=0 for spaces and Q=1
;for marks. RE.0 and RE.1 contain leftmost stop addresses.

Add.   Code     Label  Mnemonic     Comments
0728   E9       RTOL   SEX 9        ;Make R9 be RX
0729   9C              GHI C        ;Get 1/2 peak and subtract
072A   F7              SM           ;data.
072B   33 3E           BGE RLOK     ;Out if bin found,
072D   E7              SEX 7        ;else restore R7 as RX.
072E   89              GLO 9        ;Get memory pointer and
072F   FF 80           SMI 80       ;subtract spaces offset.
0731   C5              LSNQ         ;Skip 2 if spaces, else
0732   FF 40           SMI 40       ;subtract extra mark offset.
0734   57              STR 7        ;Stuff it in 07F0
0735   8E              GLO E        ;Get overange for spaces,
0736   C5              LSNQ         ;skip 2 if spaces,
0737   9E              GHI E        ;else get it for marks.
0738   C4              NOP          ;
0739   F7              SM           ;Compare pointer and overange
073A   29              DEC 9        ;Decrement memory pointer,
073B   1F              INC F        ;increment bin coutner.
073C   3B 28           BM RTOL      ;If minus, back for more,
073E   E7       RLOK   SEX 7        ;else restore RX
073F   D5              RETURN       ;and get out.

;This is the routine to calculate the mark threshold.
;It calls the above three routines, FMAX, LTOR, & RTOL.
;In the first part "peak" or "maximum" refers to the
;histogram peak generated by dots.

Add.   Code     Label  Mnemonic     Comments
0600   F8 06    THRM   LDI 06       ;06C0 into R9, start address
0602   B9              PHI 9        ;of smoothed mark data.
0603   F8 C0           LDI C0       ;
0605   A9              PLO 9        ;
0606   7A              REQ          ;Reset Q, shows right maximum
0607   D4 0740         SUB FMAX     ;Sub to find right maximum.
060A   7B              SEQ          ;Set Q, shows marks.
060B   F8 06           LDI 06       ;Put partial address in R9.1.
060D   B9              PHI 9        ;
060E   D4 0700         SUB ADD50    ;Sub to fix data.
0611   D4 0714         SUB LTOR     ;Sub to find 50% peak bin.
```

MORSE PROGRAM

```
;This is the continuation of the mark threshold routine.
Add.  Code     Label Mnemonic        Comments
0614  8F             GLO F      ;Get distance from peak
0615  57             STR 7      ;and stuff into 07F0.
0616  8C             GLO C      ;Get peak bin # and
0617  F4             ADD        ;find bin twice as far from
0618  F4             ADD        ;peak as 50% bin.
0619  BE             PHI E      ;Put it in RE.1 as left stop
061A  AF             PLO F      ;and also into RF.0.
061B  FC C0          ADI C0     ;Add offsets to make this
061D  A9             PLO 9      ;a memory address in R9 in
061E  F8 06          LDI 06     ;the smoothed mark range.
0620  B9             PHI 9      ;
;From this point "peak" refers to the dash cluster.
0621  7B             SEQ        ;Set Q to find left maximum.
0622  D4 0740        SUB FMAX   ;Sub to find left maximum.
0625  F8 06          LDI 06     ;Put partial address in R9.1.
0627  B9             PHI 9      ;
0628  7B             SEQ        ;Set Q for marks.
0629  D4 0700        SUB ADD50  ;Sub to fix data.
062C  D4 0728        SUB RTOL   ;Sub to find 50% peak bin.
062F  8F             GLO F      ;Get distance from peak,
0630  FE             SHL        ;double it
0631  57             STR 7      ;and stuff it in 07F0.
0632  8C             GLO C      ;Get peak bin # and find
0633  F7             SM         ;bin twice as far from peak
0634  57             STR 7      ;as 50% and stuff it in 07F0.
0635  9E             GHI E      ;Add same result from dot
0636  F4             ADD        ;peak and divide by two to
0637  F6             SHR        ;find estimate for mark
0638  BE             PHI E      ;threshold, put it in RE.1.
0639  C0 0640        LBR THRS   ;Jump for space threshold.
063C  00             IDL        ;IDL's to 063F inc.

;This is the routine to calculate the space threshold.
;In the first part "peak" or "maximum", refers to the
;histogram peak generated by the intra-character spaces.
Add.  Code     Label Mnemonic        Comments
0640  F8 06    THRS  LDI 06     ;0680 into R9, start address
0642  B9             PHI 9      ;of smoothed space data.
0643  F8 80          LDI 80     ;
0645  A9             PLO 9      ;
0646  7A             REQ        ;Reset Q, shows right maximum
0647  D4 0740        SUB FMAX   ;Sub to find right maximum.
064A  7A             REQ        ;Reset Q, shows spaces.
064B  F8 06          LDI 06     ;Put partial address in R9.1.
064D  B9             PHI 9      ;
```

MORSE PROGRAM

;This is the continuation of the space threshold routine.

```
Add.  Code     Label Mnemonic          Comments
064E  D4 0700        SUB ADD50  ;Sub to fix data.
0651  D4 0714        SUB LTOR   ;Sub to find 50% peak bin.
0654  8F             GLO F      ;Get distance from peak
0655  57             STR 7      ;and stuff into 07F0.
0656  8C             GLO C      ;Get peak bin # and
0657  F4             ADD        ;find bin twice as far from
0658  F4             ADD        ;peak as 50% bin.
0659  AE             PLO E      ;Put it in RE.0 as left stop
065A  AF             PLO F      ;and also into RF.0.
065B  FC 80          ADI 80     ;Add offsets to make this
065D  A9             PLO 9      ;a memory address in R9 in
065E  F8 06          LDI 06     ;the smoothed space range.
0660  B9             PHI 9      ;

;Here "peak" refers to the inter-character space cluster.

0661  7B             SEQ        ;Set Q to find left maximum.
0662  D4 0740        SUB FMAX   ;Sub to find left maximum.
0665  F8 06          LDI 06     ;Put partial address in R9.1.
0667  B9             PHI 9      ;
0668  7A             REQ        ;Reset Q for spaces.
0669  D4 0700        SUB ADD50  ;Sub to fix data.
066C  D4 0728        SUB RTOL   ;Sub to find 50% peak bin.
066F  8F             GLO F      ;Get distance from peak,
0670  FE             SHL        ;double it
0671  57             STR 7      ;and stuff it in 07F0.
0672  8C             GLO C      ;Get peak bin # and find
0673  F7             SM         ;bin twice as far from peak
0674  57             STR 7      ;as 50% and stuff it in 07F0.
0675  8E             GLO E      ;Add result from intra-space
0676  F4             ADD        ;peak and divide by two for
0677  F6             SHR        ;first estimate of space
0678  57             STR 7      ;threshold, put it in 07F0.
;Due the poor "quality" of the inter-character space peak,
;it is averaged with the mark threshold which would be the
;same for perfect code.
0679  9E             GHI E      ;Get mark threshold and
067A  F4             ADD        ;simply average with estimate
067B  F6             SHR        ;just calculated. Store this
067C  AE             PLO E      ;final result in RE.0
067D  C0 07B0        LBR CLUP   ;Jump to cleanup loose ends.
```

MORSE PROGRAM

```
;This short routine follows the space threshold calculation
;to clear up various odds and ends.

Add.  Code     Label Mnemonic        Comments
07A0  00             IDL        ;IDL's to 07AF inc.
07B0  D4 01AB  CLUP  SUB DISPY  ;Display new thresholds.
07B3  F8 00          LDI 00     ;
07B5  A9             PLO 9      ;Reset key down timer and
07B6  BC             PHI C      ;counter for key downs in
07B7  F8 01          LDI 01     ;character. Set Morse shift
07B9  B9             PHI 9      ;register to 01.
07BA  27             DEC 7      ;Point to stored RC.0,
07BB  72             LDXA       ;get it, point to 07F0,
07BC  AC             PLO C      ;restore RC.0.
07BD  C0 0400        LBR TIME   ;Jump back to timer.

;These are the commands which the program recognizes from
;an ASCII keyboard. They are immediately recognized, so
;no carriage return is required.
```

/ Restart at 03DD, sub to monitor to enter new speed, press "R" to return.

H Output histogram immediately, wait for more commands.

K Send hardcopy command to graphics terminal, wait for more commands.

M Restart at 03F6, same histogram, speed and thresholds.

N Cancel autohistograms, continue at 0400.

X Zero raw histogram data, restart at 03F6.

Y Divide raw histogram data by two, restart at 03F6.

Z Set automatic mode for histograms. If any raw bin is full, the program stops and outputs a histogram. The raw data is divided by two, but until the program is restarted the smoothed data is untouched, so other copies can be made. This command continues at 0400.

"N" or "Z" can be pressed at any time during normal program execution. The other commands all disrupt decoding. Any command not in the above table will cause a restart of the entire system to the monitor.

MORSE PROGRAM

```
;The above routines were all essential to decoding Morse
;code. The remaining sections are only needed to output
;histograms to the Tektronix graphics terminal. Register A
;points to the routine for serial port 2, not 1 as usual.
;A lot of manipulation is necessary because the plotter
;works on a 1024X by 779Y absolute matrix, with each letter
;14X by 22Y, and the histogram requires 64X by 31Y
;numbered positons, each divisible by four!!! Calculations
;can only be done in integer math, so there are problems.
;This subroutine numbers an axis 0 to F repeatedly.

Add.  Code     Label  Mnemonic          Comments
0220  F8 00    LTTR   LDI  00      ;00 into RF.0 if not Q,
0222  C5              LSNQ         ;0F if Q, start number of
0223  F8 0F           LDI  0F      ;hex-numbered axis.
0225  AF              PLO  F       ;
0226  F8 1D    MLTR?  LDI  1D      ;Output "GS" to set
0228  DA              SEP  A       ;graphics mode.
0229  D4 02E0         SUB  XYOU1   ;Output XY coordinates.
022C  D4 02F8         SUB  XYOU2   ;
022F  F8 1F           LDI  1F      ;Output "US" to set
0231  DA              SEP  A       ;alphanumerics mode.
0232  8F              GLO  F       ;Get axis counter, mask in
0233  FA 0F           ANI  0F      ;lower 4 bits, put them
0235  BF              PHI  F       ;in RF.1 and output them as
0236  D4 01F2         SUB  1HOUT   ;a single hex digit.
;RC.0 contains the low address of the current coordinate
;being lettered: F1 for X or F2 for Y; high address 07.
;RC.1 contains the axis coordinate increment: 04 for
;horizontal, 05 for vertical. R9.0 holds stop count
;for axis, 3F horizontal, FF or 10 vertical.
0239  8C              GLO  C       ;Transfer low address to RX.
023A  A7              PLO  7       ;RX points to coordinate.
023B  9C              GHI  C       ;Get coordinate increment,
023C  F4              ADD          ;add it on,
023D  73              STXD         ;restore coordinate.
023E  F8 F0           LDI  F0      ;Point RX back to 07F0.
0240  A7              PLO  7       ;
0241  1F              INC  F       ;Increment axis counter, but
0242  C5              LSNQ         ;if Q was set, decrement
0243  2F              DEC  F       ;counter by 2, effectively
0244  2F              DEC  F       ;by 1.
0245  8F              GLO  F       ;Get axis counter and
0246  57              STR  7       ;stuff it in 07F0.
0247  89              GLO  9       ;Get stop count and
0248  F7              SM           ;subtract,
0249  3A 26           BNZ  MLTR?   ;jump back for more if not
024B  D5              RETURN       ;equal, else return.
024C  00              IDL          ;IDL's to 024F inc.
```

MORSE PROGRAM

```
;These two subroutines output an XY coordinate to the
;graphics terminal which requires 10 bit X and Y values,
;broken into four 5 bit ASCII characters. Enter with an
;8 bit X value in 07F1 and Y in 07F2. Normally call XYOU2
;immediately after XYOU1, but to obtain the full 10 bit
;precision available, the two lsb's in the Y value can be
;added at that point.

Add.  Code    Label  Mnemonic        Comments
02E0  60      XYOU1  IRX             ;Point to X value,
02E1  72             LDXA            ;get it and point to Y.
02E2  A0             PLO 0           ;X into R0.0,
02E3  F0             LDX             ;get Y,
02E4  B0             PHI 0           ;Y into R0.1.
02E5  27             DEC 7           ;Point to 07F0.
02E6  27             DEC 7           ;
02E7  F6             SHR             ;Get high 5 Y bits in lower
02E8  F6             SHR             ;3 positions.
02E9  F6             SHR             ;
02EA  F9 20          ORI 20          ;OR in identifier and
02EC  DA             SEP A           ;output high Y byte.
02ED  90             GHI 0           ;Get Y again,
02EE  FE             SHL             ;put 0's in two lsb's
02EF  FE             SHL             ;and stuff it in 07F0.
02F0  57             STR 7           ;
02F1  D5             RETURN          ;Return for possible mods to
;two lsb's to get 10 bit Y coordinate if required.
02F2  00             IDL             ;IDL'S to 02F7 inc.
02F8  F0      XYOU2  LDX             ;Get Y value back,
02F9  FA 1F          ANI 1F          ;mask in lower 5 bits only
02FB  F9 60          ORI 60          ;OR in identifier and
02FD  DA             SEP A           ;output low Y byte.
02FE  80             GLO 0           ;Get X value and
02FF  F6             SHR             ;put high 5 bits in lower
0300  F6             SHR             ;5 positions.
0301  F6             SHR             ;
0302  F9 20          ORI 20          ;OR in identifier and
0304  DA             SEP A           ;output high X byte.
0305  80             GLO 0           ;Get X again and
0306  FE             SHL             ;put 0's in two lsb's.
0307  FE             SHL             ;
0308  FA 1C          ANI 1C          ;Mask in required bits,
030A  F9 40          ORI 40          ;OR in identifier and
030C  DA             SEP A           ;output low X byte.
030D  D5             RETURN          ;All done, get out.
030E  00             IDL             ;
030F  00             IDL             ;
```

MORSE PROGRAM

```
;The above three subroutines are called by this section
;which handles the overhead for lettering the axes.
;This is the actual start of the histogram routine.

Add.  Code     Label Mnemonic      Comments
0250  D4 01DA HIST   SUB CRLF    ;Output a "CR" & "LF"
0253  F8 10          LDI 10      ;Point RA to output routine
0255  AA             PLO A       ;for serial port 2.
0256  F8 07          LDI 07      ;Output "BEL" to inform
0258  DA             SEP A       ;operator of histogram.
0259  F8 1B          LDI 1B      ;Output "ESC" & "FF" to
025B  DA             SEP A       ;erase screen.
025C  F8 0C          LDI 0C      ;
025E  DA             SEP A       ;
025F  F8 FF          LDI FF      ;This is a delay while
0261  BF             PHI F       ;the screen clears.
0262  AF             PLO F       ;FFFF into RF
0263  2F      SCR?   DEC F       ;
0264  9F             GHI F       ;Get RF.1, jump back if not
0265  3A 63          BNZ SCR?    ;zero, else delay over.
0267  F8 F1          LDI F1      ;Set up to label horizontal
0269  AC             PLO C       ;axis (see LTTR for expl.)
026A  F8 04          LDI 04      ;F1 in RC.0  = X axis
026C  BC             PHI C       ;04 in RC.1  = increment
026D  60             IRX         ;04 in 07F1  = X offset
026E  57             STR 7       ;61 in 07F2  = Y offset
026F  60             IRX         ;3F in R9.0  = last count+1
0270  F8 61          LDI 61      ; 0 in   Q   = pos. incr.
0272  73             STXD        ;
0273  27             DEC 7       ;Point to 07F0 again.
0274  F8 3F          LDI 3F      ;
0276  A9             PLO 9       ;
0277  7A             REQ         ;
0278  D4 0220        SUB LTTR    ;Go and letter X axis.
027B  F8 FF          LDI FF      ;F2 in RC.0 = Y axis
027D  A9             PLO 9       ;05 in RC.1 = increment
027E  F8 F2          LDI F2      ;00 in 07F1 = X offset
0280  AC             PLO C       ;11 in 07F2 = Y offset
0281  F8 05          LDI 05      ;FF in R9.0 = last count-1
0283  BC             PHI C       ; 1 in   Q  = neg. incr.
0284  F8 11          LDI 11      ;
0286  60             IRX         ;
0287  60             IRX         ;
0288  73             STXD        ;
0289  F8 00          LDI 00      ;
028B  73             STXD        ;Point to 07F0 again.
028C  7B             SEQ         ;
028D  D4 0220        SUB LTTR    ;Letter Y axis from  F to 0.
```

MORSE PROGRAM

```
Add.  Code      Label Mnemonic          Comments
0290  F8 10           LDI 10      ;Continue lettering axes.
0292  A9              PLO 9       ;F2 in RC.0 = Y axis
0293  60              IRX         ;05 in RC.1 = increment
0294  60              IRX         ;00 in 07F1 = X offset
0295  F8 66           LDI 66      ;66 in 07F2 = Y offset
0297  73              STXD        ;10 in R9.0 = last count+1
0298  F8 00           LDI 00      ; 0 in   Q  = pos. incr.
029A  73              STXD        ;Point to 07F0 again.
029B  C4              NOP         ;
029C  7A              REQ         ;
029D  D4 0220         SUB LTTR    ;Letter Y axis from 0 to F.
02A0  C0 0310         LBR TEKH    ;Continue with histogram.
02A3  00              IDL         ;IDL's to 02AF inc.
;This section outputs the data to the histogram.
;The smoothed data is used, but this is easily changed.
0310  F8 06     TEKH  LDI 06      ;Enter with Q=0, spaces.
0312  BC              PHI C       ;0680 in RC = spaces, if Q=0
0313  60              IRX         ;06C0 in RC = marks, if Q=1
0314  F8 05           LDI 05      ;05 in 07F1 = X offset
0316  57              STR 7       ;60 in 07F2 = Y offset, Q=0
0317  60              IRX         ;66 in 07F2 = Y offset, Q=1
0318  F8 80           LDI 80      ;
031A  C5              LSNQ        ;
031B  F8 C0           LDI C0      ;
031D  AC              PLO C       ;
031E  F8 60     TEK1  LDI 60      ;
0320  C5              LSNQ        ;
0321  F8 66           LDI 66      ;
0323  73              STXD        ;
0324  27              DEC 7       ;Point to 07F0 again.
0325  C4              NOP         ;
0326  F8 1D           LDI 1D      ;Output "GS", make this dark
0328  DA              SEP A       ;vector to new position.
0329  D4 02E0         SUB XYOU1   ;Output coordinates.
032C  D4 02F8         SUB XYOU2   ;
032F  60              IRX         ;
0330  60              IRX         ;Point R7 to Y coordinate.
0331  C4              NOP         ;
0332  0C              LDN C       ;Get smoothed data pointed
0333  F6              SHR         ;to by RC,
0334  F6              SHR         ;divide by 4,
0335  EC              SEX C       ;
0336  F4              ADD         ;add that to original,
0337  E7              SEX 7       ;in effect multiply by 1.25.
```

MORSE PROGRAM

```
;........continuation of histogram.........
;In order to fit histogram onto screen with even spacing,
;the very last (3Fth) bin is not output. This is not a
;problem as there should be no useful information there.

Add.  Code     Label Mnemonic      Comments
0338  CD             LSQ           ;Skip 2 if Q=1, marks.
0339  F5             SD            ;Spaces, vector down, so
033A  C4             NOP           ;subtract data from offset.
033B  C5             LSNQ          ;Skip 2 if Q=0, spaces.
033C  F4             ADD           ;Marks, vector up, so add
033D  C4             NOP           ;data to offset.
033E  73             STXD          ;Store Y coordinate in 07F2,
033F  27             DEC 7         ;point back to 07F0.
0340  C4             NOP           ;
0341  D4 02E0        SUB XYOU1     ;Sub to output high Y byte.
0344  0C             LDN C         ;Get data again,
0345  FA 03          ANI 03        ;mask in lower 3 bits only.
0347  CD             LSQ           ;Skip 2 if Q = 1 = marks,
0348  F5             SD            ;else spaces, subtract.
0349  C4             NOP           ;
034A  C5             LSNQ          ;Skip 2 if Q = 0 = spaces,
034B  F4             ADD           ;else marks, add. Store
034C  C4             NOP           ;this result back in 07F2.
034D  57             STR 7         ;This adds the two lsb's to
034E  D4 02F8        SUB XYOU2     ;Y for 10 bits, output it.
0351  1C             INC C         ;Increment RC, point to next
0352  60             IRX           ;data, increment RX,
0353  F0             LDX           ;get X coordinate.
0354  FC 04          ADI 04        ;Add 04, move along axis,
0356  57             STR 7         ;restore it.
0357  60             IRX           ;Point RX to Y coordinate.
0358  8C             GLO C         ;Get low data pointer,
0359  FF BF          SMI BF        ;subtract BF, neg if still
035B  CF             LSDF          ;spaces, skip 2 if positive.
035C  30 1E          BR TEK1       ;Neg, jump back, more spaces
035E  7B             SEQ           ;Set Q for marks,
035F  27             DEC 7         ;point to 07F0 again.
0360  27             DEC 7         ;If above subtraction was 0,
0361  32 10          BZ TEKH       ;back to START mark output.
0363  60             IRX           ;Else point to 07F2 = Y.
0364  60             IRX           ;
0365  FF 40          SMI 40        ;Subtract 40 from previous
0367  3A 1E          BNZ TEK1      ;result, if nonzero, back
0369  7A             REQ           ;for more spaces, else reset
036A  27             DEC 7         ;Q, point back to 07F0,
036B  27             DEC 7         ;
036C  C0 0370        LBR OUPAR     ;and jump ahead to output
036F  00             IDL           ;parameters.
```

MORSE PROGRAM

;This section outputs the speed and thresholds to the
;graphics terminal at the top of the histogram.

```
Add.  Code     Label  Mnemonic          Comments
0370  60       OUPAR  IRX          ;00 in 07F1 = X offset
0371  60              IRX          ;BA in 07F2 = Y offset
0372  F8 BA           LDI BA       ;
0374  73              STXD         ;
0375  F8 00           LDI 00       ;
0377  73              STXD         ;Point back to 07F0.
0378  C4              NOP          ;NOP's to 037B inc.
037C  F8 1D           LDI 1D       ;Output "GS" for dark vector
037E  DA              SEP A        ;
037F  D4 02E0         SUB XYOU1    ;Output coordinates for
0382  D4 02F8         SUB XYOU2    ;first lettering position.
0385  F8 1F           LDI 1F       ;Output "US" for alpha-
0387  DA              SEP A        ;numeric mode.
0388  F8 02           LDI 02       ;Point RC to the start of
038A  BC              PHI C        ;the first heading stack
03BB  F8 B0           LDI B0       ;at 02B0.
03BD  AC              PLO C        ;
038E  D4 03B6         SUB HEAD     ;Output first heading.
0391  D4 01CE         SUB RDOUT    ;Output speed from RD.
0394  F8 B7           LDI B7       ;Point RC to second heading.
0396  AC              PLO C        ;
0397  D4 03B6         SUB HEAD     ;Output second heading.
039A  9E              GHI E        ;Move mark threshold to RF.1
039B  BF              PHI F        ;
039C  D4 01E2         SUB 2HOUT    ;Output it as 2 hex digits.
039F  F8 CB           LDI CB       ;Point RC to third heading.
03A1  AC              PLO C        ;
03A2  D4 03B6         SUB HEAD     ;Output third heading.
03A5  8E              GLO E        ;Put space threshold in RF.1
03A6  BF              PHI F        ;
03A7  D4 01E2         SUB 2HOUT    ;Output it as 2 hex digits.
03AA  F8 01           LDI 01       ;Histogram finished, point
03AC  AA              PLO A        ;RA to routine for port 1,
03AD  C0 04B8         LBR KCHK     ;wait for keyboard input.
03B0  00              IDL          ;IDL's to 03B5 inc.
;This short subroutine outputs ASCII headings. Enter with
;start of data stack in RC, returns when first 00 found.
03B6  4C       HEAD   LDA C        ;Get heading element,
03B7  C6              LSNZ         ;skip 2 if nonzero,
03B8  D5              RETURN       ;else return.
03B9  00              IDL          ;
03BA  DA              SEP A        ;Output heading and
03BB  30 B6           BR HEAD      ;jump back for more.
03BC  00              IDL          ;IDL's to 03BF inc.
```

MORSE PROGRAM

;This is the heading data. Each stack ends with 00.

Add.	Code	Character	Add.	Code	Character
02B0	53	S	02C9	20	space
02B1	50	P	02CA	00	end
02B2	45	E	02CB	20	space
02B3	45	E	02CC	20	space
02B4	44	D	02CD	20	space
02B5	20	space	02CE	20	space
02B6	00	end	02CF	53	S
02B7	20	space	02D0	50	P
02B8	20	space	02D1	41	A
02B9	20	space	02D2	43	C
02BA	20	space	02D3	45	E
02BB	4D	M	02D4	20	space
02BC	41	A	02D5	54	T
02BD	52	R	02D6	48	H
02BE	4B	K	02D7	52	R
02BF	20	space	02D8	45	E
02C0	54	T	02D9	53	S
02C1	48	H	02DA	48	H
02C2	52	R	02DB	4F	O
02C3	45	E	02DC	4C	L
02C4	53	S	02DD	44	D
02C5	48	H	02DE	20	space
02C6	4F	O	02DF	00	end
02C7	4C	L			
02C8	44	D			

REFERENCES

1. Bedzyk, W.: "Machine Translation of Morse Code Using a Microprocessor", NTIS, AP-785-130, June 1974, pp. 1-113.

2. Bell, E.: "Processing of the Manual Morse Signal Using Optimal Linear Filtering, Smoothing and Decoding", NTIS, AD-A019-493, Sept. 1975, pp. 1-156.

3. Blair, C.: "On Computer Transcription of Manual Morse", Journal of the Association for Computing Machinery, July 1959, Vol. 6, No. 3, pp. 429-442.

4. Day, R.: "Communication Aids for Cerebral Palsied Children", M. Eng. Thesis, McMaster University, Hamilton, Ontario, Sept. 1976.

5. Freimer, M.; Gold, B.; Tritter, A.: "The Morse Distribution", IRE Transactions on Information Theory, March 1959, pp. 25-31.

6. Gold, B.: "Machine Recognition of Hand-Sent Morse Code", IRE Transactions on Information Theory, March 1959, pp. 17-24.

7. Gonzales, C.; Vogler, R.: "Automatic Radiotelegraph Translator and Transcriber", Ham Radio, Nov. 1971, pp. 8-23.

8. Grappel, R.; Hemmenway, J.: "Add the 6800 Morse Keyer to Your Amateur Radio Station", BYTE, Oct. 1976, pp. 30-35.

9. Guenther, J.: "Machine Recognition of Hand Sent Morse Code Using the PDP-12 Computer", NTIS, AD-786-492, Dec. 1973, pp. 1-153.

10. Hickey, W.: "The Computer Versus Hand Sent Morse Code", BYTE, October 1976, pp. 12-17, 106.

11. McElwain, C.; Evens, M.: "The Degarbler - A Program for Correcting Machine Read Morse Code", Information and Control, 1962, Vol. 5, pp. 368-384.

12. Reyer, S.; Steber, G.: "The Morse-A-Letter", Popular Electronics, Jan. 1977, pp. 37-43.

13. Signetics Analog Data Manual, "Phase Locked Loops", 1977, pp. 807-860.

14. Smith-Vaniz, W.: Barrett, E.: "Morse to Teleprinter Code Converter", Electronics, July 1, 1957, pp. 154-158.

15. Triggs, R.: "Morse Code Communication Aid", Chedoke Hospital Internal Report, Hamilton, Ontario, Sept. 1979.

16. User Manual for the CDP1802 COSMAC Microprocessor, RCA Corporation, 1976, pp. 1-115.