A DISPLAY SYSTEM

FOR

BLISS SYMBOLICS

A DISPLAY SYSTEM
FOR
BLISS SYMBOLICS

by

E. G. CALLWAY, B. ENG.

A **Project**

Submitted to the School of Graduate Studies

in Partial Fulfilment 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: A Display System for Bliss Symbolics

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

SUPERVISOR: Dr. Morris Milner

NO. OF PAGES: 124, 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. Milner, Ontario Crippled Children's Centre, for his advice and guidance. Appreciation is also expressed to Ted Bojanowski, Madeleine Harris, Ted Iler, Barbara Rush, and Tony Wallace for their continual advice and support.

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

COMMUNICATION WITH SYMBOLS

1.1 Brief History of Bliss Symbolics

Charles K. Bliss had an idea for a language which could be universally understood and accepted. It was to be pictographic (and occasionally arbitrary) and not based on previous languages and alphabets. Like Esperanto it was designed to increase understanding and lessen nationalistic tensions. Also like Esperanto, it was universally rejected for this purpose.

The Bliss system was rediscovered as a method for communicating with the disabled. The Ontario Crippled Children's Centre, in conjunction with Mr. Bliss, have developed more symbols and a teaching system. It is designed for use primarily with cerebral palsied children who lack normal language development.

1.2 Communication and the Handicapped

Many handicapped people have disabilities which are obvious to the rest of the world, such as those confined to a wheelchair. Often public buildings are modified to accomodate them, by providing ramps and larger washrooms. Their problems can be alleviated, and their presence is socially acceptable.

For those with a communication handicap, the situation is very different. Whether the problem arises from a physical or a mental disability is irrelevant. If a person is unable to talk to those he meets in everyday life, any transactions, from "hello" to business affairs become laborious and often impossible. When there is some speech, but it is slurred, few make the extra effort required to listen.

A vicious circle often ensues in which continual rejection leads to less effort to communicate. The person feels he has lost all control over his environment, and lapses into "learned helplessness". (1) Against this there are many stories about those who astonish the world with their brilliance (or more important their normality) when communication blocks are removed. Of course, Helen Keller is the classic example, but it is a common occurrence. (2)

1.3 Cerebral Palsy

Cerebral palsy 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. Day gives a pertinent summary. (3)

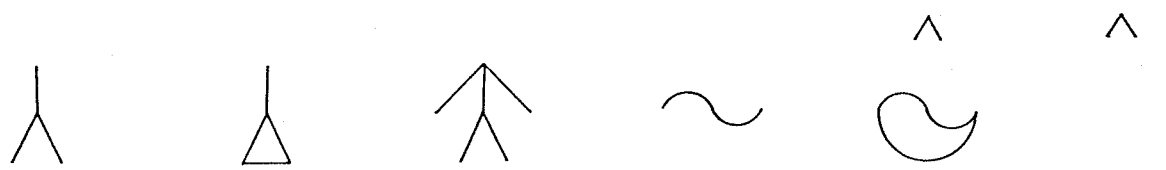
Many of those with cerebral palsy have other difficulties. Approximately 80% have speech impairment, of which

20% are unable to speak at all. There is a 60% incidence of retardation, against 3.5% of the normal population. (4) The combination of widespread motor, vocal and mental handicaps can cause great problems in communication. There is also the 40% group with normal or better intelligence who may be unable to express their thoughts due to purely motor difficulties. They are erroneously labelled as retarded since normal stimuli do not evoke a normal response. This group responds well when given a method of communication.

1.4 Bliss Symbolics as an Aid

There are several references giving detailed explanations of the system, ranging from dictionaries (5) to expositions of the underlying philosophy (6). A full explanation of the teaching system for symbols is beyond the scope of this work, and the various references should be consulted for more information. A dictionary of all the symbols used in this thesis is given in Appendix A.

Many of the symbols are pictographic, but there are some which are arbitrary. Some examples are given below:



Man Woman Father Water Wash (Verb)

These symbols will appear obvious after explanation, but they show some of the basic symbol constructions. The verb modifier can be placed over any symbol to convert it into a verb. It is an "action" indicator, derived from a volcano cone, "first primeval action on our young earth". This is arbitrary, but simple. The other explanations are as follows: (7)

Man - This is not a person with two legs, but a growth of the action symbol, since a man is someone "to whom Action, Endeavour, Success make Life worth living".

Woman - This is not a man with a skirt. The triangle is the symbol for God the creator, and since the woman is also a creator, she gets the triangle.

Father - This is a man with the protection symbol added. It invokes the idea of a roof for protection.

Water - It looks like a wave, hence it means water.

Wash - Like "father", this is also a symbol made from smaller units. There is water above a container, and the action symbol to make it a verb.

From these examples, it can be seen that very often the symbol is clear, despite its convoluted origins. They also show that once a basic unit is learned, such as "man" or "water", it can be used with a consistent but expanded meaning to build more symbols.

One important point is that the relative sizes and placement of the components of a symbol are important. The basic shapes of the action and protection sub-symbols are similar, but it is their size, placement and the angle subtended that differentiate them. For this reason the symbols should be accurately drawn. This is not usually a problem for two reasons. Instructions for drawing the symbols are complete and explicit, with all important edges falling on a rectangular grid. This allows the teacher to easily construct correct replicas of the originals. On the other side, the handicapped student never has to draw a symbol. Because of the motor disabilities, a large effort is required merely to indicate the symbol on a prepared board.

A problem may arise with computer generated symbols. If a display with poor resolution is used, sub-symbols such as "action" and "protection" may be indistinguishable. This is especially true when they are reduced to fit a large number of symbols onto a screen. For this reason any computer display should have adequate resolution, both in the storage and display sections.

CHAPTER II

A RATIONALE FOR THE USE OF BLISS SYMBOLICS

2.1 Success with Symbols

Bliss symbolics have been shown to be of great help as a communication system for the cerebral palsied. One question to be asked is why not spend the time teaching a normal language such as English? The answer is that symbols work and English does not. The children may eventually progress to English, but they often cannot acquire it without having first "learned to learn" using symbols. The question of why English cannot be learnt immediately remains.

A partial answer may be had by examining two widely separated studies. One may show why English is specifically unsuitable for the handicapped, and the other why Bliss symbolics are.

2.2 Japanese Children

In an interesting paper entitled "The Rarity of Reading Disorders in Japanese Children", Dr. Makita makes some startling observations. (8) Based on surveys and literature searches, it was noted that in English-speaking countries, the incidence of reading disabilities in school

children was approximately 10%, while in Japan it was hardly 1%. (9) Several discordant theories from Western literature ranging from birth defects to broken homes were mentioned, each trying to explain the cause of reading difficulties. The important point was made that in all other measures the Eastern and Western children were statistically identical. For this reason, it was concluded that for 90% of Western children with reading problems, the standard theories were grossly inaccurate. (10)

2.3 Japanese Scripts

The only real difference appeared to be in the actual way the two languages were written. Japanese has two basic kinds of script, Kanji and Kana. The former consists of ideographs modified from classical Chinese. Approximately 1850 are in daily use. The latter consists of two phonetic alphabets, each with 48 letters. It was not made clear why two such alphabets coexisted, as they contained identical sounds represented by different symbols. (10)

2.4 Kana Script

The phonetic scripts differ from the English alphabet in several ways: (11)

Alphabet	Kana
Each letter has several different sounds depending arbitrarily on where it is used. a = /ay/, /uh/, /ah/	Each symbol has only one pronunciation.
Some letters can sound like a consonant or a vowel. c = /see/ or /k/	With 5 vowels as FIXED exceptions each letter is a complete syllable with one consonant and one vowel.
Several consonants can be put together for inconsistent pronunciation. ph, ng, kn, gh, th	Consonants are only linked through vowels.
Each pronouncable sound does not have a known letter. /ai/ = i, y, igh, eye, aye	Every sound is one-to-one with a letter.

Although it takes several forms, the clear difference is that Kana letters are always identically pronounced as written, whereas the English alphabet is highly inconsistent. This will be shown to be important. Kana has another advantage in its repetitive structure of consonant-vowel, consonant-vowel. (12)

2.5 Kanji Script

As mentioned, this script developed from ancient Chinese pictographs, with almost 2000 symbols in regular use. While the two Kana scripts are fully introduced by the second grade, the process of learning the Kanji symbols is extended, with about 180 added each year until 1000 are known at the end of the sixth grade. Since a lot of memory

work is required, more children have trouble with this script than with the Kana phonetic alphabet.(12)

The Kanji symbols are closer to English in that their pronunciation varies depending on their use. The pictograph for "eye" is pronounced differently when combined in "eye drop", and differently again in "nearsightedness".(13)

There are some fundamental differences however between English words and Kanji symbols when reading new or forgotten material. The following pairs are Kanji symbols which appear similar, in that they have a common sub-symbol:(14)

釘　針　左　右　花　草

Nail.....Needle　Left.....Right　Flower.....Grass

If one of these symbols is partially forgotten or misread, it is very likely it will be transformed (mentally) into a symbol in the same class of meanings. This is because the sub-symbols have a meaning of their own which has been consistently used in the building of other words.

In contrast, "cat" gives a totally incorrect clue to the meaning of "catastrophic". Word pairs such as saw/was, good/could and left/felt are often confused due to misleading auditory and visual clues. In all these cases, a

misreading will produce a totally incorrect meaning. A thorough grounding in Latin, Greek and various old European languages may help, but these are beyond the grasp of a child with multiple handicaps.

Another pertinent observation was made concerning Kanji reading. When the reading skills of a normal were impaired by an acquired cerebral lesion, the ability to read Kana was greatly reduced, while Kanji abilities were almost intact. (14) This would seem to suggest that the visual nature of Kanji is more "natural" than the phonetic basis of Kana.

It would seem that although Kanji pictographs are more difficult to master, even for those with normal intelligence, the basic concept of consistent building blocks remains. The percentage of reading disabilities with Kanji is higher than with Kana, but it is still lower than with English.

2.6 Another Symbol System

In a book called "Language Without Speech", R. Deich and P. Hodges give a detailed account of a language intervention scheme for severely retarded children. (15) The first part is a comprehensive review of language development in children, normal and retarded, along with the intervention schemes of others. These range from finger signing to

symbols. The second part is an account of their experience with the Premack symbol system.

2.7 Of Men and Monkeys

Dr. Premack became famous as the man who taught a chimpanzee, Sarah, to "talk".(16) Instead of speech the method involves the use of coloured plastic pieces to represent words. For each new word a new piece was added. Beginning with nouns (apple), the chimp was taught many other types of words, such as verbs (give), colours (red), names of personnel (Mary), as well as less specific words (food, dirty, student). In each case, the method used was to unequivocally show that a certain piece of plastic represented one object, action or even idea (the "if-then" clause).

Others were able to duplicate the work, and extend it to cover the developmentally handicapped.(17) It should be pointed out that the connection is that the system was able to teach language where no language had been thought possible.

2.8 Teaching the Severely Retarded

Deich and Hodges felt that Dr. Premack's work would have relevance in helping the severely retarded to communicate. One such group of children had an average IQ of below 20.(18) At this level an IQ test involves questions such as "Has anyone reported the child ever was able to take a

drink?" (19) Using the Premack system, most of the children in even this very low group were able to learn some symbols and begin communicating. Clearly the system must contain some of the most basic elements of language.

2.9 A Basis for Language

Helen Keller noted that until she realized that "everything had a name", the idea of a method to talk to others was unknown. (20) The basis of any language is that some sound or written symbol (the "referent") is used to stand for the real object or idea. Dr. Premack reported that the chimps in his study occasionally confused the object and referent by trying to write with an apple or eat a symbol. Deich and Hodges found that not even the lowest functioning children made that mistake. (21) They might not learn that a symbol meant anything, but they would not presume it was the object. It was also found that a symbol did not have to resemble the object it represented. (22) From this it is possible to conclude that there is a basic human ability to learn a language based on abstract symbols.

2.10 Remembering and Reproducing Symbols

To use a language normally, one needs to perform tasks of recognition and reproduction. On hearing /ah/ - /pull/ or seeing "apple", it is necessary to remember what these symbols refer to a piece of round red food. To ask for

an apple it is further necessary to remember how to say /ah/-/pull/ or write "apple". Deich and Hodges discovered that learning the first task did not teach the second. This was theorized to be both a conceptual(23)and a memory problem.(24) In order to overcome this, two methods are used.

All of the symbols known to a student are always kept in view. A user must still remember enough to recognize a symbol, but full recall of all its details are unnecessary.

Unlike other communication methods such as speech, writing or finger signing, no complicated physical actions are required.(25) The user has only to move the symbol required. In this way one physical action suffices for all the different symbols.

Deich and Hodges reported on a study of memory conducted by Haber.(26) Large numbers of words and pictures were rapidly spoken and shown to the subjects, who were later asked to identify those words and pictures from a larger set. The visual recall for the pictures was over 90% despite the fact that over 2500 had been shown in a two day session. The investigator concluded that a separate and possibly better memory existed for visual stimuli.

2.11 Bliss - the Best of Both?

The above discussion presented a few observations on two very different communications methods, each of which

may, in some ways, be "better" than English. It can be shown that Bliss symbolics incorporate the good features of each method.

i) Consistent Meaning of Sub-units:

Kana - Each letter has only one pronunciation.

Kanji - A small pictograph has a consistent, reuseable meaning.

English - Every letter has several sounds and vice-versa.

Premack - Each symbol has one meaning.

Bliss - The smaller symbols are widely reused with the same meaning always. There is no pronunciation.

ii) Consistent Organization:

Kana - The pronunciation uses a repetitive structure.

English - This is possibly one of the most degenerate languages in terms of pronunciation and tense generation.

Premack - There is little grammar other than left to right progression.

Bliss - The rules for grammar are simple and fixed.

iii) Effects of Misreading:

Kana - Misreading is difficult due to the fixed pronunciation.

Kanji - The pictograph is often transformed into one of similar meaning if some of the sub-symbols are misunderstood.

English - Written English begs to be misread, and further offers no clues for correction.

Premack - A totally different meaning is obtained.

Bliss - It is essentially the same as Kanji in this respect.

iv) Use of Building Concepts:

Kanji - Smaller symbols are often used to generate new ones.

English - "Bake" expands to "bakery", but "infants" do not constitute an "infantry".

Premack - There is no growth other than adding new symbols.

Bliss - In both grammar (noun to verb) and the vocabulary (water to wash), the rules for expansion are simple and consistent.

v) Effects of Brain Injury:

Kana/Kanji - The phonetic Kana is impaired, but the pictographic Kanji remains.

English - While often phonetic, English is never pictographic, hence a loss should occur.

Premack - This is proven to work in this instance.

Bliss - This is purely a visual language, and also has been shown to work in difficult cases.

vi) Memory and Reproduction Requirements:

Kana - The sound reinforces the visual, but a complicated response is required.

Kanji - The visual appearance reinforces the meaning but a complicated response is required.

English - The sound, the meaning, the response and the appearance are always difficult and often conflicting.

Premack - The response is simple, and all the symbols are always in view.

Bliss - This is the same as for Premack.

2.12 Holophrastic Communication

In holophrastic speech, one word is used to do the job of many. This type of speech is found deliberately in telegrams, but is not encouraged in everyday use. Most languages pride themselves in the wealth of their descriptive terms, rather than their compactness. In young children however, holophrastic speech is common, simply due to a lack of vocabulary.(27) A child may call a furry winter jacket "doggie", but this is done as a logical extension of a word evoking a texture, not as a joke. (28)

During normal development, new words are always added to one's vocabulary, obviating the need for holophrastic speech. For the learning impaired however, each new word is a long struggle, and the total number acquired may remain quite low. Because of this, holophrastic communication remains essential.

Unless one is writing poetry, the average word in English evokes too wide a range of meanings to be useful. In contrast to this, a new language such as Bliss is ideal. Each symbol has a complete and definite meaning for use by itself, and the users are free of the intellectual baggage which may prevent them from making the logical extensions required.

2.13 Organization and Irrelevant Stimuli

Deich and Hodges reiterated the work of others in these areas. Although normal children remembered best with a self-imposed memory scheme which suited them, those with reduced intelligence performed better when the material was in an externally organized framework.(29) Bliss symbolics do well in this regard, as there is a rigid set of rules governing all aspects of their use. While this may be restrictive to an English major, it is helpful for the handicapped.

In a normal classroom, the sound of a police siren outside will certainly cause a momentary distraction. For the child with severe learning disabilities, it is important to reduce all distractions, as they are far less able to tune them out.(30) "Irrelevant stimuli" includes the teaching material used, as well as the environment. Due to its many inconsistencies, English contains a lot of distracting information. In Bliss symbolics, and other designed languages, each component presented contributes directly to the meaning.

2.14 Conclusions

Bliss does appear to be suited to the mentally and perceptually handicapped, even if this was not its original purpose. Its fixed rules, lack of misleading information and ability to grow in a controlled fashion all

contribute to its known success. Since it is used primarily with the non-vocal, its lack of a pronunciation method is an asset. The simple output method (pointing) removes the need for learning a complicated system in addition to that required for recognition. It appears to be similar in many ways to the Premack system, with the exception that Bliss symbols are designed to be combined, while a new Premack symbol is required for each word. Because of this, the Premack system probably is better for the very lowest functioning children, although the Bliss system can remain at a single symbol level for basic items.

CHAPTER III

STORAGE AND DISPLAY OF BLISS SYMBOLS

A person remembers what a Bliss symbol "looks like", but a computer needs systematic methods for storage and recovery of information.

3.1 Similar Problems

On a pocket calculator the numerals 0 through 9 must be displayed. It is common practice to use a seven segment display as shown in Figure 1. The seven segments and one decimal point allow 2^8 or 256 on/off combinations. Of these only 20 combinations are valid, ten numerals with a decimal and ten without. This storage requirement can be met by a memory 20 words long by 8 bits wide, which is quite small and therefore inexpensive.

For a computer terminal the character set may contain 128 characters, such as the numerals, upper and lower case alphabet, punctuation and special characters as required. A good representation of the above characters can by formed using a 9 by 7 dot matrix as shown in Figure 2. The data can be stored as 128 X 7 or 896 words of 9 bits each. Again this is a small amount of memory which is easily put on one integrated circuit.

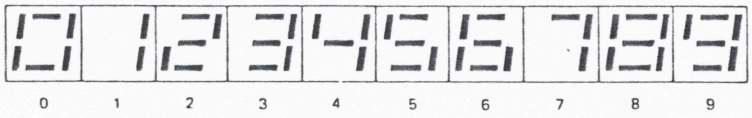


Figure 1. Seven Segment Display

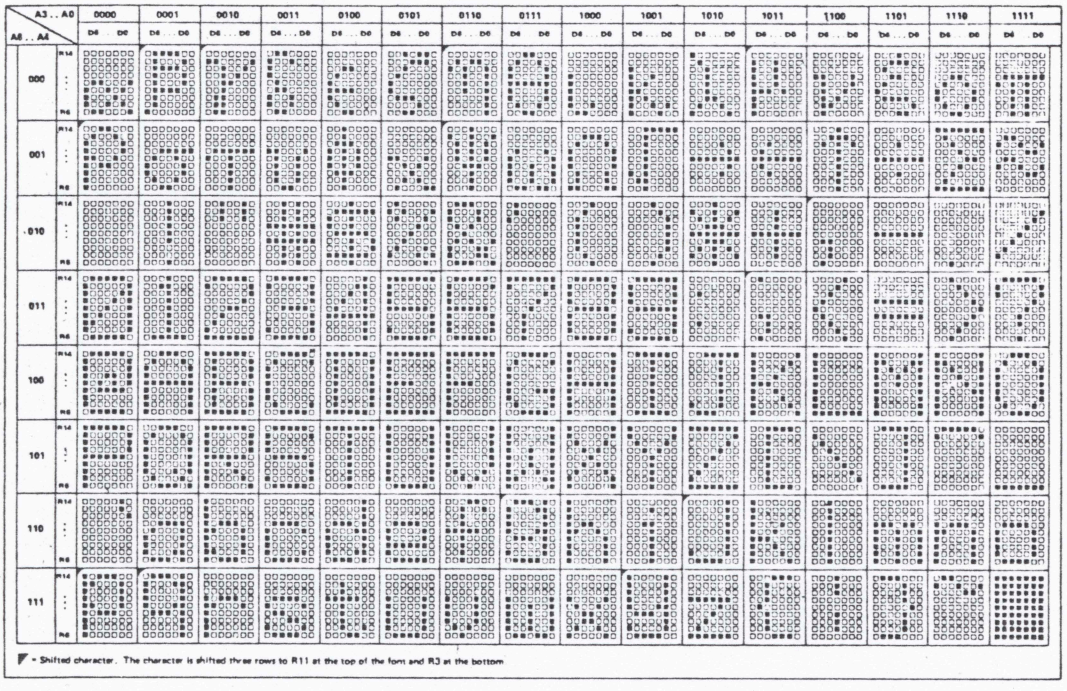


Figure 2. 9 X 7 Dot Matrix Letters

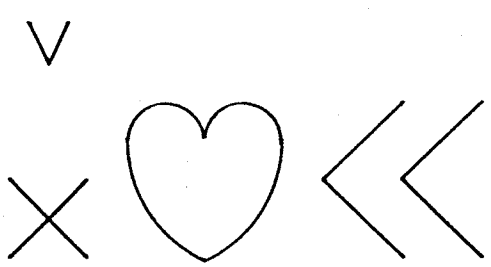
3.2 Bliss Symbols

Unfortunately Bliss symbols do not lend themselves as easily to computer storage for several reasons. These are variable complexity, variable vocabulary, and variable size of presentation.

All 128 characters in an alphanumeric set can be put into a 9 by 7 matrix because they are of similar width and complexity. The symbols for person and angry (Figure 3) are very different in width. The first might require a 5 by 12 matrix, while the second might require 50 by 20 for good representation. If 50 by 20 was adopted for all symbols the cost would be prohibitive, and a variable size for each symbol would need a method for telling where one symbol stops and another begins.

One reason that storing alphanumerics on an integrated circuit is inexpensive is volume. A single character set will suffice for hundreds of thousands of terminals. Bliss symbols are a language, not an alphabet, and therefore are variable. A typical Bliss vocabulary contains 400 symbols, but new ones are always being created as demanded. Although an alphabet is by nature unchanging, a language dies if it cannot grow. Additionally, each symbol can be modified to form adjectives, verbs and so on. In this way a 400 symbol set would require many thousand symbols if each variation was treated as a fixed item. A low cost Bliss

a) "Person"



b) "Angry"

Figure 3. Bliss Symbol Complexity

memory cannot be produced in the same fixed manner as an alphanumeric memory.

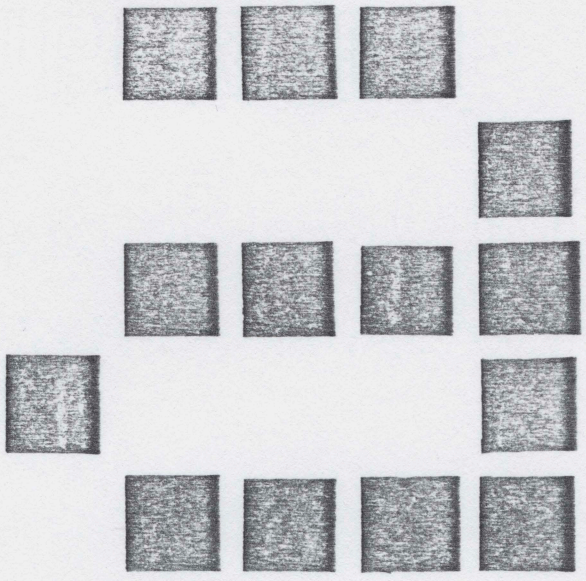
Figure 4 shows a letter produced using a dot matrix and also shows it enlarged. Notice that the inconspicuous dots become objectionable blocks when simply expanded. As Bliss symbols are used in a teaching environment, it would be helpful to be able to present them in different sizes. When a new symbol is introduced to the class it might fill the TV screen, but a student on his own might need 10 symbols visible at once to complete a sentence. Again, storing each symbol in several sizes would require too much memory.

3.3 Dot Graphics Requirements

The above discussion emphasized dot graphics because it is a popular and inexpensive method for presenting computer output on a television screen. The screen is divided into an array of dots (usually called pixels) and a single bit of memory is assigned to each dot.

The number of pixels on the screen is determined by two factors. The number of lines on the raster scan controls the vertical count in a one to one fashion, whereas the video bandwidth sets an analog limit to the number of horizontal pixels. Both factors directly affect the cost and complexity of the unit. As horizontal or vertical resolution increases, the memory requirements also grow. If very high resolution is required, a non-standard monitor is needed

a) Small



b) Enlarged Using Same Data

Figure 4. Dot Matrix Drawing

adding a further cost. It was decided to use a video format which was compatible with a standard television monitor.

3.4 Vertical Resolution

In a normal noninterlaced scan there are 262 scan lines, but some are lost during vertical retrace. 192 lines was chosen as the vertical display area. An interlaced scan would have allowed double the resolution, but did not seem necessary. This allows a maximum of 24 rows of alphanumerics, each 8 pixels high.

3.5 Horizontal Resolution

For the alphanumeric display there are 512 pixels per line to allow 64 characters per line, each 8 pixels wide. In the graphics mode this was considered excessive for two reasons. It would require a larger memory, and the pixels would be rectangular, rather than square. Square pixels allow a simple X-Y coordinate addressing scheme to be used, with equal resolution in both directions.

The maximum graphics resolution was set at 256 horizontal by 192 vertical, giving a rectangular display to fit a standard television screen. This requires a maximum of 6K bytes of memory, which is a reasonable amount.

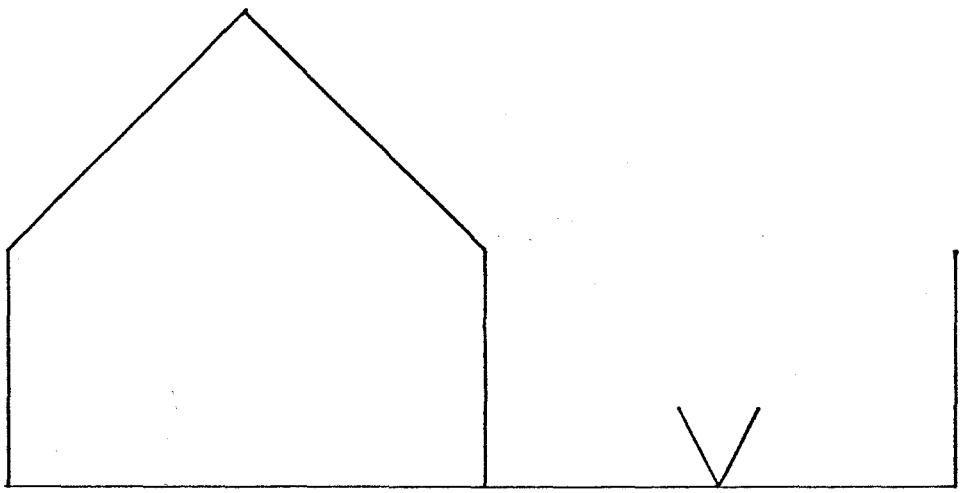
Although the memory is divided into 8 bit bytes for the microprocessor, to the software it is made to appear as an X-Y coordinate system, with each pixel having a unique

address. Each pixel can be set or reset, producing a white or black dot.

3.6 Reduction of Storage Requirements

To satisfy the particular requirements of Bliss symbol storage and still use a small memory, it was decided to use a coding scheme other than storing each dot of each symbol with all its variations. A cursory examination of a symbol table (see Appendix A) reveals that virtually all the symbols are composed of straight line segments and circular arcs. Since any line can be described by two points, and a circular segment by three, a large data reduction can be had by only storing these relevant points. In this way the interior of a circle does not have to be stored as a blank area, saving much memory.

To achieve this two important subroutines are required, one to draw a straight line given two points, and the other to draw a circular arc given three. With this method a square 20 by 20 pixels requires only 16 bytes to store the endpoints (two X, two Y) for each side. A simple dot storage method needs (20 X 20)/8 or 50 bytes. A further benefit is that now the symbols can be easily scaled and moved by multiplying the endpoints and adding an offset before drawing. As shown in Figure 5 , a symbol stored in this fashion does not appear as blocks when enlarged, and loses minimal resolution when reduced.



a) Enlarged Using Same Data

b) Small

Figure 5. Endpoint and Line Drawings

As well as straight and circular segments, there is a further level of data compaction available. Small symbols, such as "house" and "water" reappear many times as part of larger symbols, such as "backyard" and "lake". A single 8 bit number could be used to specify one of 256 of these small symbols, so that a large symbol such as "radio" could specify a square using one byte instead of 50. Although a reduction of only 3 to 1 was obtained for storing the square on its own, a reduction of 50 to 1 is obtained each time the symbol is required later.

3.7 Display Formatting

While the hardware sets the maximum resolution, the programmer is not limited to one type of display. Two sizes of graphics and six of alphanumerics are provided.

The basic Bliss method requires the symbol to be shown with the English equivalent underneath. For this reason the display is split into 12 regions of 16 scan lines each. The individual regions can be formatted in software with one of the eight possibilities above. Although this means that letters or numbers in a graphics region have to be drawn as graphics, this is not a problem. It ensures that alphanumerics which are an integral part of a symbol will be correctly scaled and placed, rather than jumping to fixed positions.

The following format options are available:

Mode #	Type	Whole Screen	----Resolution---	Per Region
0	Graphic	256H by 192V	pixels	256H by 16V
1	Graphic	128H by 96V	pixels	128H by 8V
2	Alpha	64H by 24V	characters	64H by 2V
3	Alpha	16H by 24V	characters	16H by 2V
4	Alpha	32H by 24V	characters	32H by 2V
5	Alpha	64H by 12V	characters	64H by 1V
6	Alpha	16H by 12V	characters	16H by 1V
7	Alpha	32H by 12V	characters	32H by 1V

Each mode has a particular use:

Mode 0) This is the normal graphics mode, producing the highest resolution. All of the graphics examples given use this mode.

Mode 1) This is a low resolution graphics mode which requires 1/4 the memory of mode 0. Using this, four complete pictures could be stored in memory and quickly switched to the display under user direction. This is useful for menu selection, "flashcards", and games.

Mode 2) This is the densest alphanumerics mode, for large amounts of text or closely packed symbols in a menu.

Mode 4) These alphanumerics fit a display with 6 to 10 symbols.

Mode 6) This produces very large letters, suitable for annotating a single symbol which fills the screen.

Mode 7) This looks best on a display with 4 to 6 symbols.

Modes 3 and 5 do not have visually pleasing aspect ratios and would not normally be used.

3.8 Software Components

The software falls into three main categories, utilities, math and graphics. In addition there are some stored Bliss symbols to demonstrate the system. All of the software written is contained in three EPROM's which reside in the system. An overview of the software is given here, while detailed explanations can be found in Appendix B, and the actual code in Appendix C.

All of the software is designed as subroutine modules, allowing their integration into a larger system. Consequently the routines do not respond to commands from a keyboard, but require parameters to be passed from a calling program. To run the system, the first instruction should be a jump to "START", followed by the user program. There is no monitor program, but data can be entered and programs run from the hexadecimal keyboard.

3.9 Utility Subroutines

The utilities package consists mainly of "house-keeping" routines which initialize the processor, manipulate the stack, allow subroutine calls, move data, and allow a user set delay.

The most important piece of software in this set is "EPROM". At the time this unit was developed, floppy disk storage was not inexpensive, so erasable-programmable read

only memory (EPROM) was chosen as the storage medium. These memories can be written and read electrically, but are only erased by ultraviolet light. A complicated sequence of high voltage pulses is required for the writing process, but this is provided by "EPROM". It is only necessary to specify the location of the data to be saved and run the program. In this way programs were reliably saved and quickly recalled without the use of cassette tapes.

An expanded utility set should include input-output routines, both for a terminal and the cassette driver.

3.10 Math Subroutines

Most of the math package was prompted by the requirements of drawing circles. The simpler routines allow multiplication, division, and double precision addition and subtraction.

To compact the memory required for storing circular arcs, it was decided to store only three points. Although the two endpoints and another point on the arc are sufficient, the reconstruction is complicated. Instead the two endpoints and the centre were chosen. To solve the equations a square root program was needed, which is implemented using an iterative method. Eight passes are required to give an 8 bit root for a 16 bit number.

3.11 Graphics Primitives

An applications programmer does not want to be concerned with details of the display hardware, so low level routines are needed to hide the hardware.

The basic graphics display has 256 pixels horizontally and 192 vertically for a total area of 49,152 pixels. For the convenience of the hardware these are grouped into 6,144 eight bit bytes with memory address 2000H at the top left hand corner. To address a particular dot on the screen, it is required to know both the byte and bit which contain that dot. The routine "AM256" allows the programmer to use a normal X-Y coordinate addressing scheme with the point (0,0) in the lower left corner. All of the graphics routines go through "AM256" just before going to the screen memory. Any Y coordinate greater than 192 is chopped to 192 and does not cause an error.

Other low level subroutines set, reset and test individual pixels, and format, clear and complement the entire screen.

"DLINE" takes two endpoints and draws a straight line between them in any orientation. The normal equation of the form $Y = mX + B$ could not be used as a vertical line and has infinite slope. Instead the program steps from one endpoint to the other with steps proportional to the total X or Y distance required.

"CIRCL" draws a counterclockwise arc from one endpoint to the other, given the centre. The only restriction is that all three points lie on a 256 by 256 grid. If the two endpoints are the same a complete circle is drawn. Errors can arise if the endpoints are not equidistant from the centre. Small errors in specifying these values are absorbed by the program, which draws a best fit arc joined to the endpoints by additional straight lines.

3.12 High Level Graphics

The above routines allow a programmer to do the basic functions necessary for drawing simple figures, but detailed calling instructions still have to be followed. To fill the screen with a meaningful display requires at least one higher level of language.

To achieve an effective data compaction it was noted above that a method was required for recalling commonly used symbols. A protocol was developed in which each graphics command occupied a 7 byte memory space. The first 6 bytes are for data such as endpoints, while the last byte is the actual instruction. Graphics figures can be drawn by using a list of endpoints and commands, rather than having to explicitly write a computer program. All of the low level programs are available, including circles, lines, data moves, delays and formatting. The move instruction is used to place an alphanumeric string (stored elsewhere) into the

screen memory. The last two permit teaching programs with dynamically changing displays to be used.

As a final addition, there is a set of commands which allow an instruction list to call another list as a subroutine. Using this feature, a complex symbol such as the heart (which requires four circular segments) is invoked by a single instruction when required by other symbols. This saves a lot of memory and programming time.

CHAPTER IV

HARDWARE DESIGN

4.1 Basic Elements

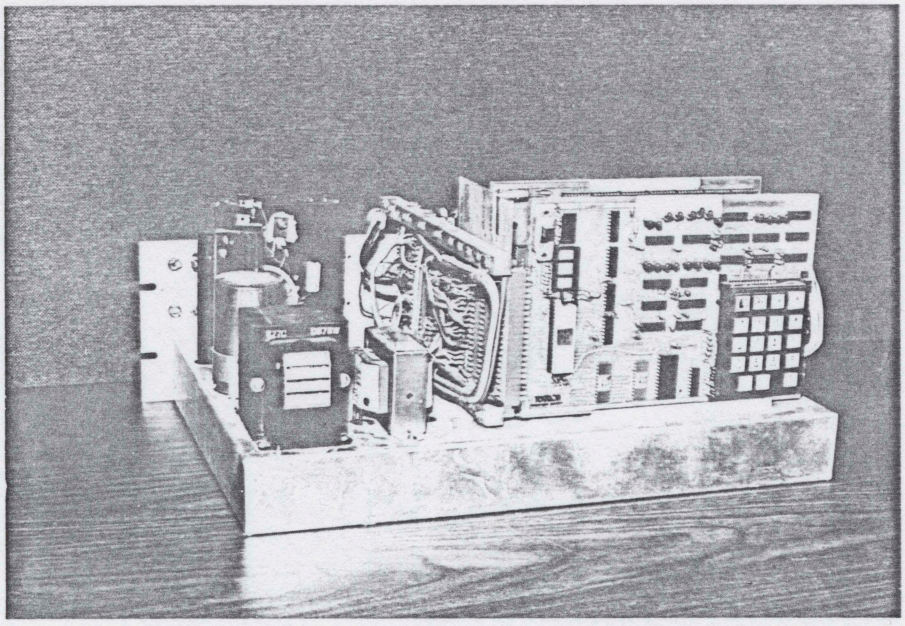
The hardware comprises a central processing unit (CPU), a serial input-output (I/O) board, an EPROM programmer, and a memory mapped video display generator spread over several boards. To support these, there is a power supply and a backplane and cardcage. The unit is shown in Figure 6.

All the boards have a standard 44 pin edge connector, but they are not interchangeable. Due to the complexity of the interconnections, it was necessary to wire each socket separately for each board.

The chassis mounts in a standard 19" rack cabinet, with the I/O and video connections at the rear. Presently, the front must remain off for access to the keypad on the CPU. If turn-key software is developed, the front could be closed with only a reset switch present.

4.2 CPU Board

This board section uses a commercially available board based on the RCA 1802 CMOS microprocessor. In addition to basic CPU support, it also contains a hexadecimal keypad and an LED display of address and data. These allowed



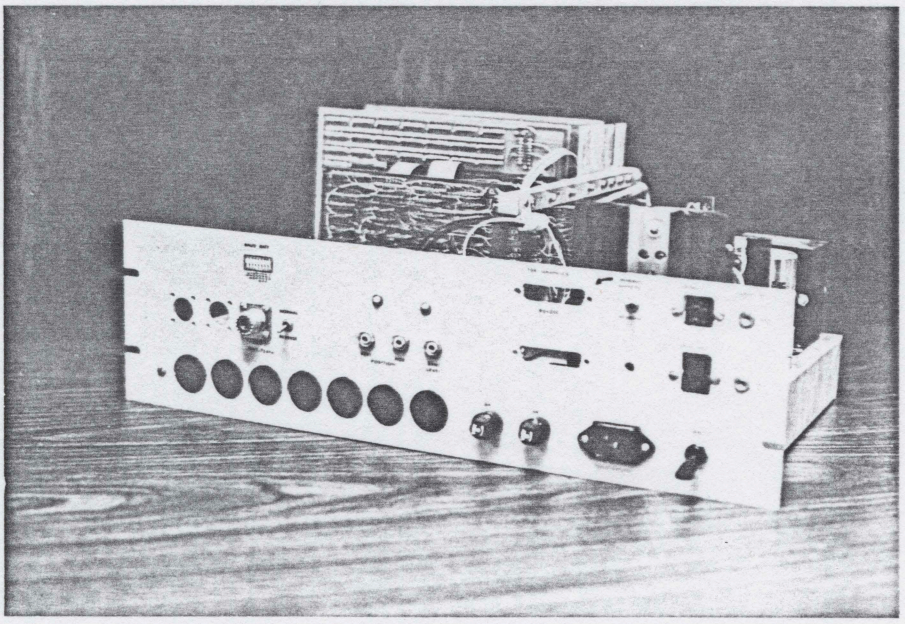


Figure 6. Bliss Display Unit

data to be entered into the system without the use of a monitor program. The schematic is shown in Figure 7 , and full details can be found in the operating manual. (31)

4.3 Power Supply

This is a conventional linear supply using fixed 3 terminal regulators. It has the following outputs:

Voltage (volts)	Current (amps)
+12	1
-12	1
+5	5
-5	1

The +5 volt supply is much larger as it provides the main power for the system. As a fault on the +5 would be disasterous, it has an overvoltage crowbar. This shorts the output if it tries to rise significantly over 5 volts.

4.4 Address Buffer and EPROM Board

This board performs several functions, all associated with memory control. The schematic is given in Figure 8.

The 1802 multiplexes the 16 bit address onto an 8 bit bus, with the high 8 bits available first, latched by TPA, and the low 8 bits present for the remainder of the cycle. IC1, 2 buffer the 8 address lines and MWR, MRD. IC3 latches the high address. IC4 decodes MA13' - MA15' so that

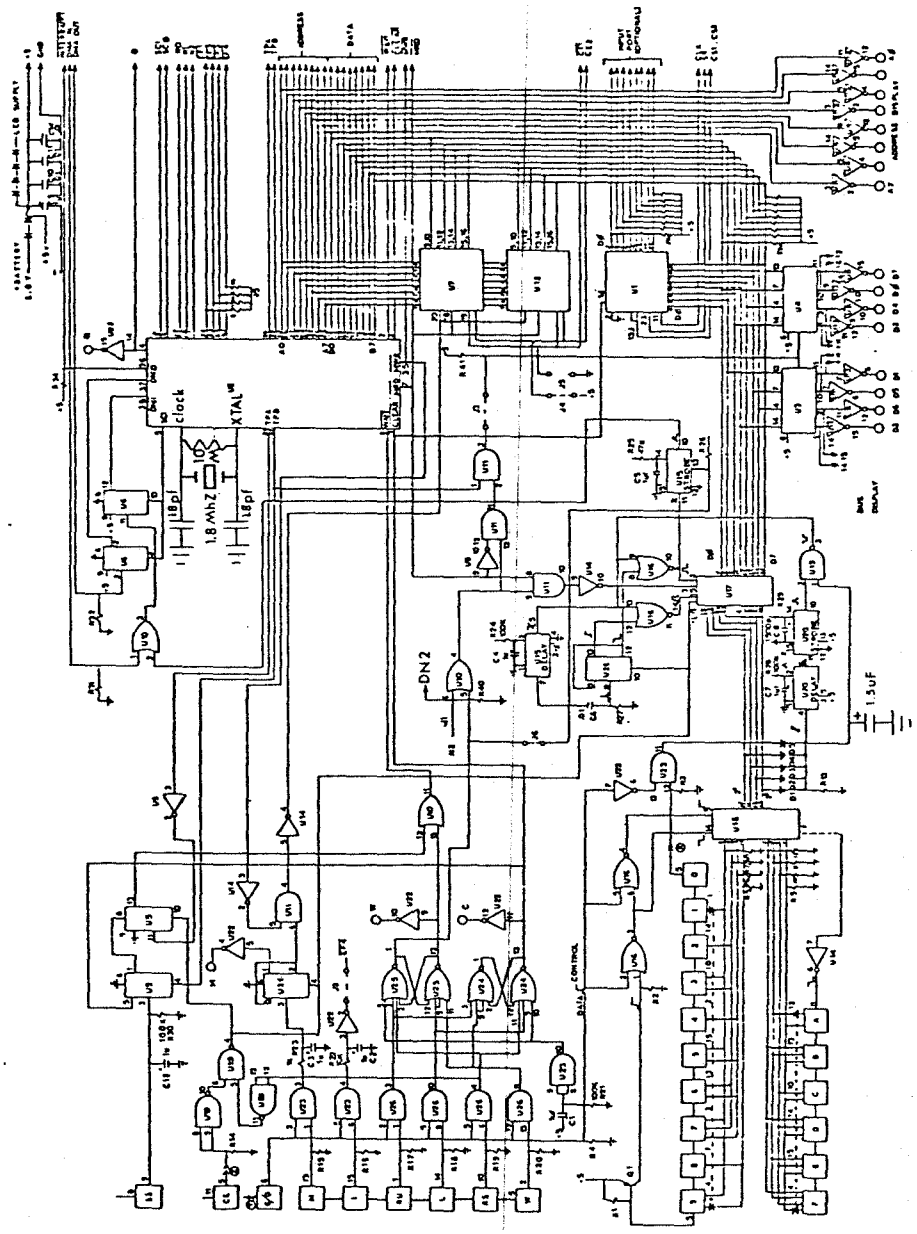


Figure 7. CPU Board

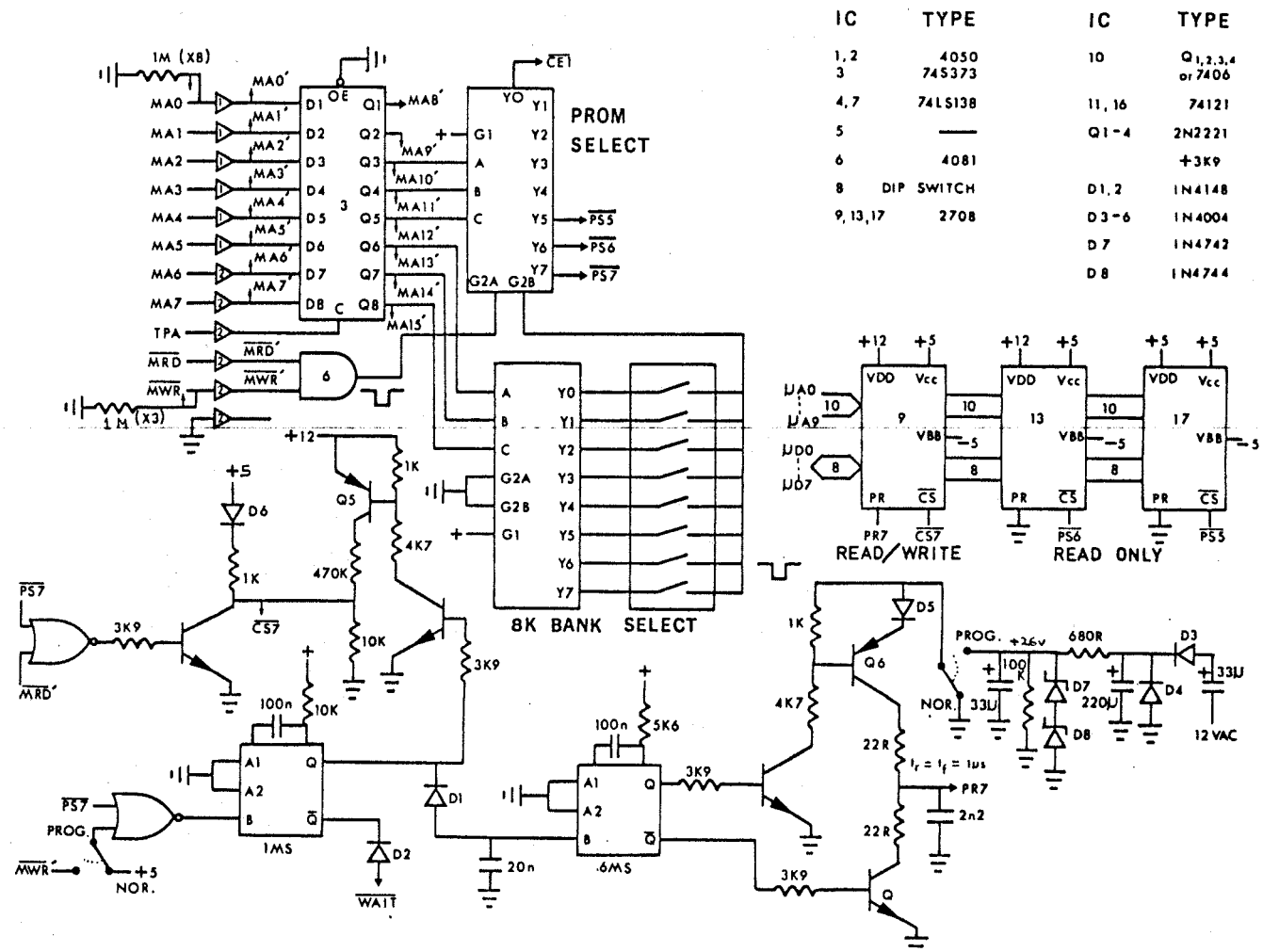


Figure 8. Address Buffer/EPROM Board

the DIP switch IC8 can be used to select to which 8K page of memory the EPROMs on this board and the 256 byte RAM on the CPU belong. Position 1 is usually selected so that the first 8K page, 0000 - 1FFF, is enabled. IC7 further decodes this to 8 1K pages for the individual EPROMs. The very first 1K segment, 0000 - 03FF, is sent back to the CPU board as CE1 so that its 256 byte RAM appears cyclically 4 times in a 1K space.

The board contains sockets for 3 2708 EPROMs, IC9, 13 and 17, which appear at addresses 1C00 - 1FFF, 1800 - 1BFF, and 1400 - 17FF respectively. There is space and decoding for more EPROM on this card.

The remaining circuitry forms a programmer for the 2708 EPROMs. Socket 9 is wired to allow programming when the PROGRAM/NORMAL switch is down in the program position. In the normal setting PS7 is an active low 5v chip select, the program pin PR7 is held at ground, and the monostables in IC11, 16 are disabled.

When the program mode is selected and a write to 1C00 - 1FFF is detected, IC16 starts the 1ms cycle by pulling the WAIT line low to stop the CPU and hold the address and data on the bus. Q5 pulls CS7 up to +12v. After a very short delay caused by D1 and the 20n capacitor, IC11 applies the actual 26v, .6ms programming pulse through Q4 and 6. D3, 4

7 and 8 comprise a voltage doubler to get 26v from a 12vac line. Q1-4 and the associated 3K9 resistors were originally a 7406 open collector IC, hence the use of a plug-in header for these components.

The programmer as built supplies the minimum hardware to burn a 2708. To fully program a memory the subroutine "EPROM" is required.

4.5 RAM Board

This is the main memory for the system, containing the CPU stack and the display memory. It is a commercial RAM board, but it has been modified for use as a display memory. It is shown in Figure 9.

The board's position in memory was initially fixed by IC6, 7 and 9, but this function has been replaced by the TV address board. It is configured as 7K of RAM from 2000 to 2BFF and 1K of EPROM from 2C00 to 2FFF. The RAM from 2800 - 2BFF is used for the stack, the rest for display.

The low address lines TA0 - TA7 enter through the edge connector, while the upper 5, TA8 - TA12, enter via a DIP cable into the old IC7 socket. The highest 3, TA13 - TA15, are decoded on the TV address board and are not needed on this board.

The read and write signals, TMRD and TMWR, are also controlled by the TV address board.

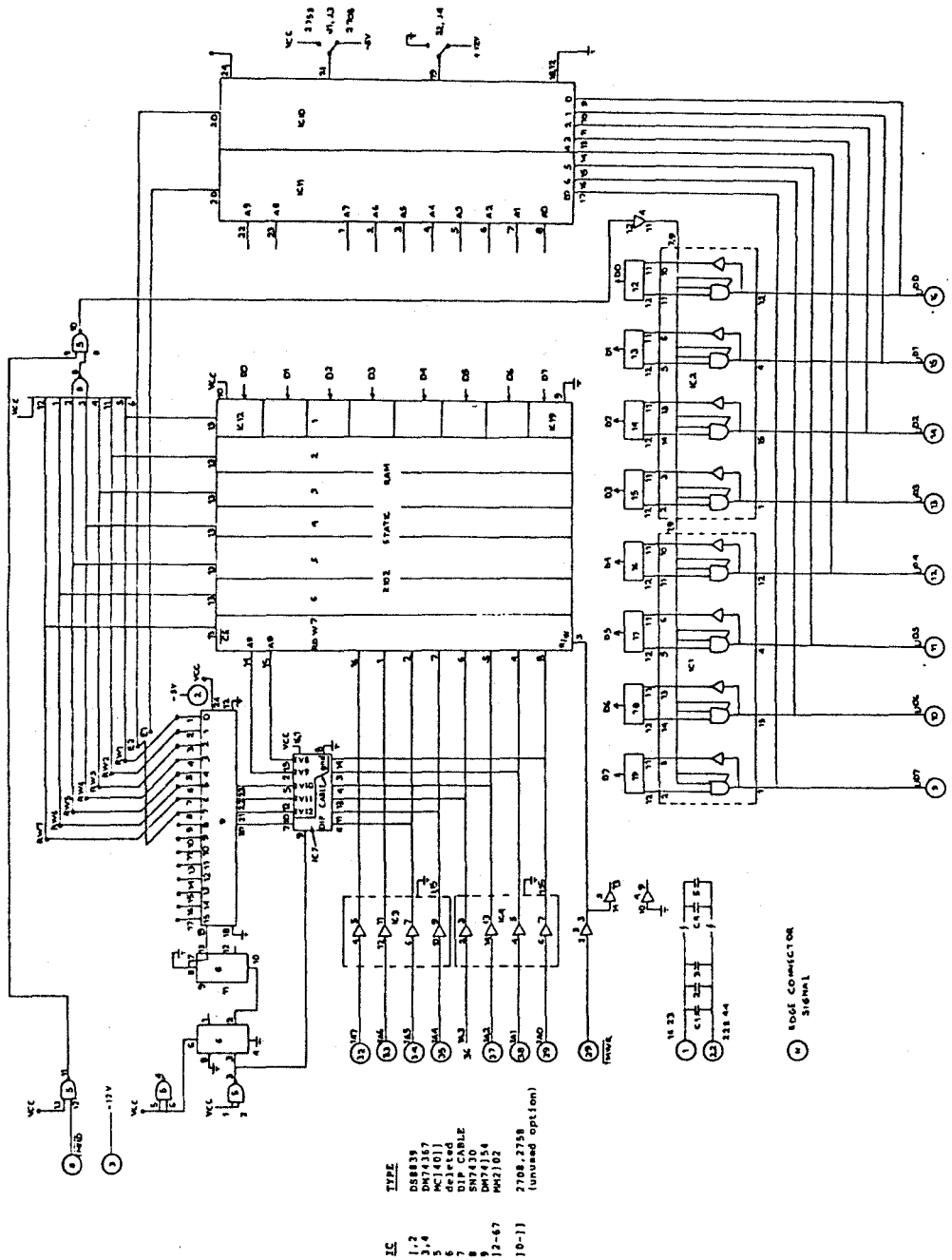


Figure 9. RAM Board

When the RAM is being accessed by the CPU, the normal bidirectional data bus on pins uD0 - 7 is used. If the RAM is being read by the display (never written), the data is picked off before the bus buffers and sent to the TV data board on TD0 - 7, and this data does not appear on uD0 - 7 because no TMRD signal is generated to enable IC1 and IC2. An an example TD0 is taken from IC2, pin 11. The bus buffer chips IC1 and 2 also undirectionally buffer the CMOS level data from the CPU to the display boards, uD0' - 7'. uD0' is taken from IC2 pin 10. This two port access requires just more wires, not more chips.

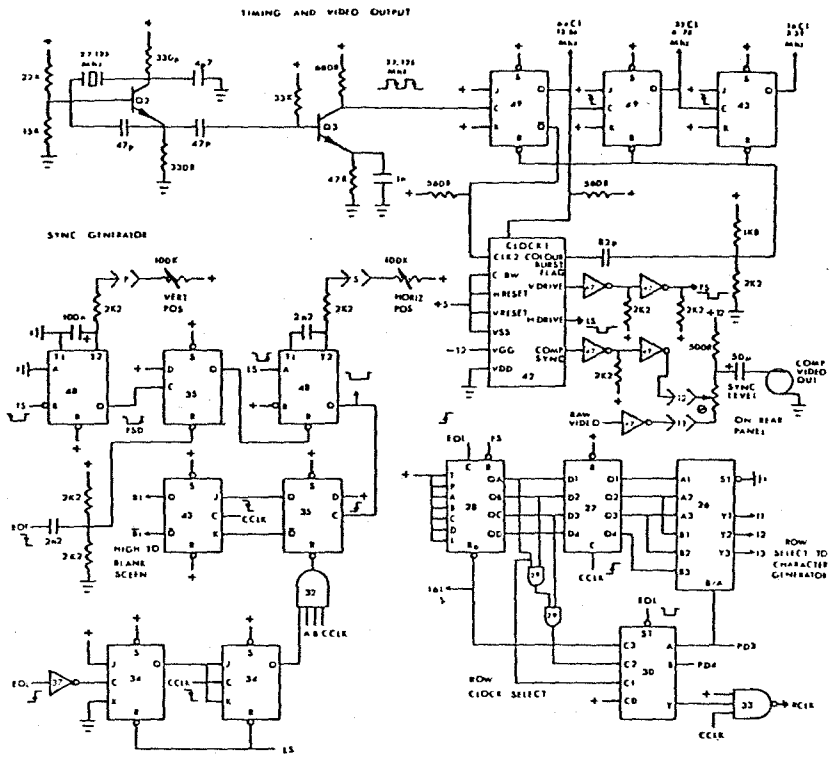
Socket IC10 can be used for another 2708 EPROM and is addressed as 2C00 - 2FFF. IC11 cannot be used as it would make the total board be 9K, and it only is allowed 8K. Note that the contents of IC10 cannot be directly displayed as it sits on the CPU side of the data bus.

4.6 T. V. Data Board

This is a complicated board comprising two main sections, the basic timing and sync generator (Figure10), and the data handling section (Figure11).

4.7 Timing and Sync Generation

Q2 is a 27.125 Mhz crystal oscillator which is the basic clock for the system. This is divided by IC49 and 43 to produce the various dot clocks 64CL, 32CL, 16CL and the



IC	TYPE
25	TMS2501
26	74LS157
27	74175
28	74LS161
29	74LS08
30	74153
31,39,40	74S374
32	74LS20
33	74LS10
34,43	74LS112
35	74LS74
36,44	DIP CABLE
37	74LS04
38	LM555
41	74161
42	3262ADC
45	74S251
46	74LS86
47	7406
48	4528
49	74S112
Q2,Q3	2N2369

Figure 10. TV Data Board - Timing and Sync

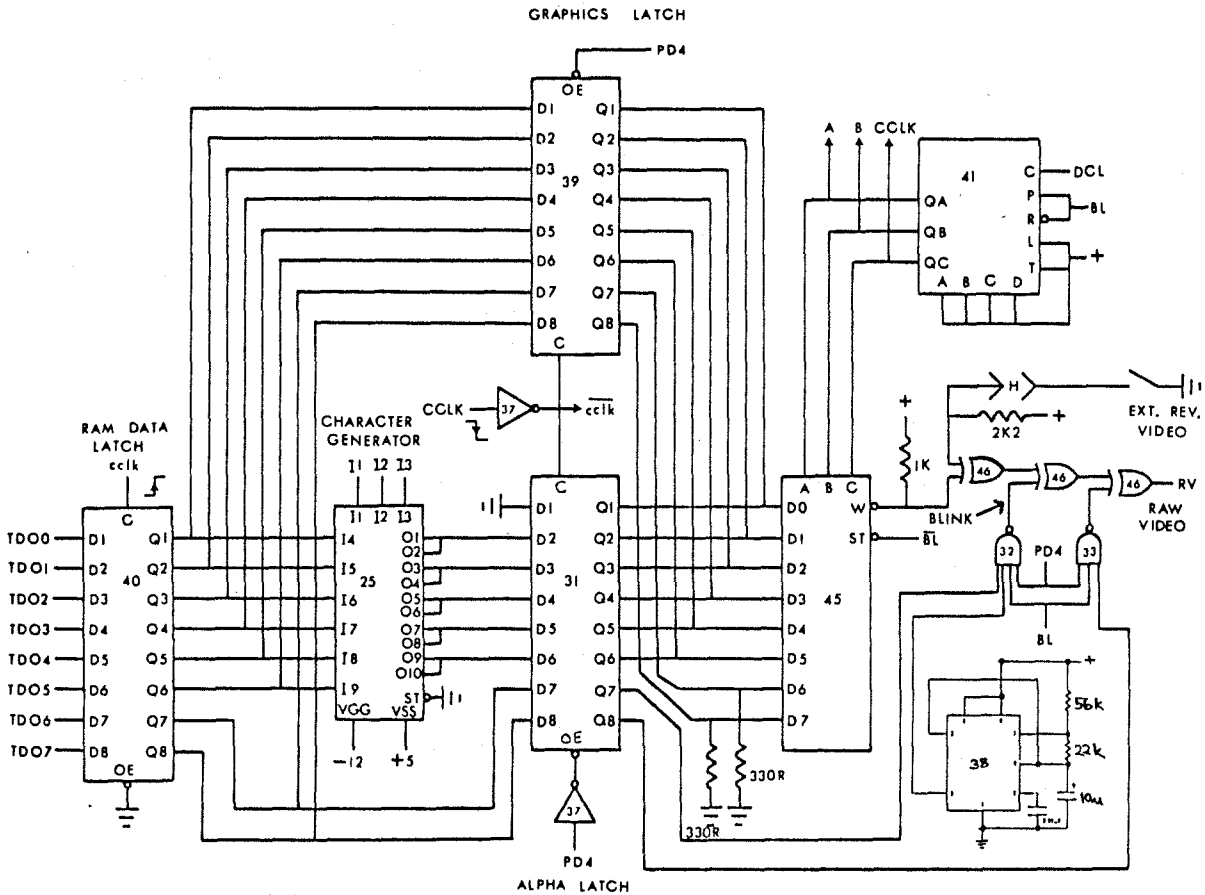


Figure 11. TV Data Board - Data Section

complementary drive for IC42.

IC42 is a composite sync generator which produces the signals required to drive a standard 525 line monitor. The colour burst flag occurs at the beginning of each line and applies a very short reset pulse to IC49 and 43. This ensures that no matter which dot clock is selected, its phase is fixed at the beginning of each line. The vertical drive output is buffered by IC47 and used as the frame sync signal FS. The composite sync is combined with the raw video in IC47, an open collector high voltage inverter chip and used to produce a standard video signal into a 75 ohm load. The sync level control should be adjusted so that the sync pulses are 0.5 volts more negative than the video.

IC34, 35, 43 and 48 control the blanking at the start and finish of lines and frames. FS is delayed by IC48 to produce the delayed frame sync, FSD. This delays the start of display relative to the vertical sync and hence on the screen. FSD sets IC35 so that the next LS pulse will begin displaying the first line of a new frame. The end of frame signal EOF clears IC35 so that the next line cannot start a display until the new FS signal appears.

The circuit is complicated by the fact that the system must display two more characters after EOL signals the end of a line. IC34 captures the short EOL pulse and

delays it by one CCLK. On the last dot of that character, A, B and CCLK go high together resetting IC35, so that when the 2nd character finishes, CCLK clears IC43, blanking the screen.

At the start of each line LS is delayed through IC43 to produce the delayed line sync LSD which positions the display horizontally on the screen. LSD sets IC35, which is clocked into IC43 to unblank the screen and start displaying a new line. By clocking the start of a line any jitter in the monostables is removed.

IC26, 27, 28, 29 and 30 generate the row select for the character generator, and the row clock RCLK for the TV address board. IC28 is reset once each frame by FS and clocked once each displayed line by EOL regardless of the display format selected. The outputs are delayed by one character in IC27 and passed to IC26. By selecting 3 out of the 4 lines to form the row selects I1-3 for the character generator, two different character heights can be used, 8 or 16 lines, selected by PD3, IC26 and 27 are ignored in graphics modes.

IC30 selects the signal used for RCLK, determining for how many lines the same data will be displayed. C0 changes the data every line for 256 graphics, controlled by the EOL signal to the strobe on IC30. C1 changes it every 2nd line for 128 graphics, C2 every 8 lines for single

height characters, and C3 every 16 for double height. The selected signal is combined with CCLK in IC33 to produce the actual RCLK.

4.8 Data Handling

This section captures the data from the 7K RAM and converts it to serial video. In the 64 character per line mode, a new character is displayed every 600 ns, but the access times for the RAM and character generator IC25 are 450 and 250 ns respectively, greater than 600 ns. To allow for this the data is buffered twice, once in IC40 and again in IC31 and IC39. PD4 selects whether alphanumerics from IC31 or graphic from IC39 are enabled.

IC45 is an 8 to 1 line data selector driven by counter IC41 to convert the ASCII or graphic data to serial form. IC41 also divides the dot clock by 8 to produce the character clock CCLK. It is reset and enabled by the blanking signal BL.

The exclusive OR gates in IC46 are used to allow inversion of the video signal, turning black to white. The external reverse video switch (located on the back panel) allows inversion of the whole image in graphics or alphanumeric modes. In alphanumeric mode, data bit 6 high will cause the character to blink for use as a cursor. Data bit 7 high causes the character to remain inverted. PD4 is applied to IC32 and IC33 so that these options are suppressed in graphics mode.

The video inversions are additive, and the blanking signal to IC32 and IC33 ensures that the border is never inverted but remains black.

4.9 T. V. Address Board

This board addresses the 7K RAM using either internal display counters or an address from the microprocessor. It also controls the format of the screen and is shown in Figure 12.

4.10 Display Modes and Control

The screen is divided into 16 segments of 16 lines. Only the first 12 of these are displayed for an active area of 192 lines. Each segment may be separately programmed under software control for a different display mode.

The display RAM occupies an 8K space, although only 7K of memory exists and only 6K can appear on the screen at one time. The RAM normally starts at 2000 (as far as the 1802 is concerned), but this is hardware selectable with DIP switch 24. The first address displayed can be varied in 128 byte increments, allowing for software scrolling.

All the programmable features can be set using an 8 bit control word with an OUT 3 instruction.

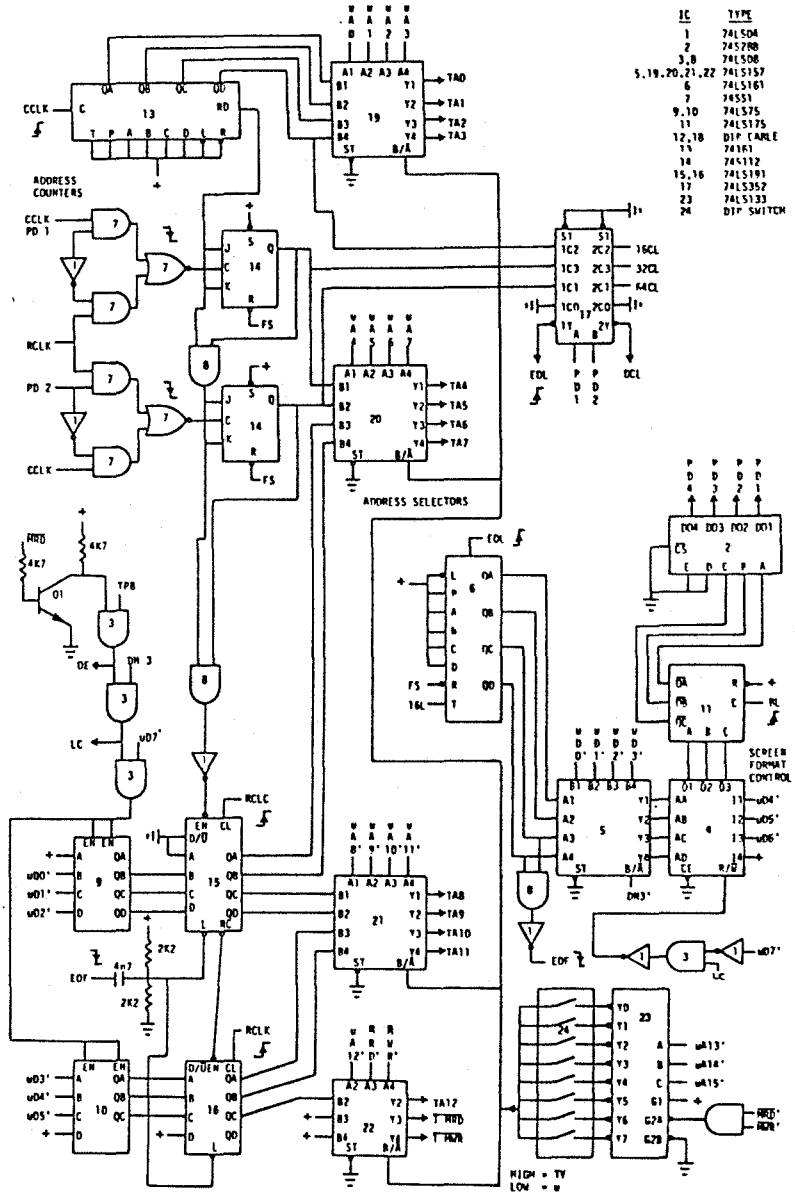


Figure 12. TV Address Board

CONTROL BYTE	FUNCTION
Ommm ssss	Set segment sss to mode mmm. Only segments 0000 to 1011 and modes 000 to 111 are displayed. ALL segments should be set a valid mode during initialization, requiring 16 control bytes.
lxaa aaa	The first address displayed is set to 0010 aaaa a000 0000. This byte can be changed for scrolling without affecting the memory contents.

MODE	BYTES LINE	LINES/RCLK	MEM	DISPLAY
000	32	1	6K	Graphics, 256H by 192V
001	16	2	3K	Graphics, 128H by 96V
010	64	8	1.5K	Alpha, 64H by 24V
011	16	8	3/8K	Alpha, 16H by 24V
100	32	8	3/4K	Alpha, 32H by 24V
101	64	16	3/4K	Alpha, 64H by 12V
110	16	16	3/16K	Alpha, 16H by 12V
111	32	16	3/8K	Alpha, 32H by 12V

Bytes/line is the number of bytes required from the memory per line on the screen. Lines/RCLK is the number of lines for which the same data is repeated. As an example, in mode 010 the same 64 bytes are fetched for 8 lines before the next 64 bytes are addressed. The memory sizes shown are the amount of memory on the screen if the whole screen is set to that mode.

Q1 and IC3 decode the OUT 3 signals. When bit 7 is high, the control word is loaded into latches IC9 and IC10 as the scrolling address. If bit 7 is low, the control word is interpreted as a mode control.

Counter IC6 is clocked every 16 lines using the over-flow from IC28, and its output shows which segment is being displayed. At the start of the 13th segment QC and QD

are decoded to produce the end of frame signal EOF.

IC4 is a 16 bit by 4 word memory which holds the mode for each segment. Normally its address comes from IC6, but if OUT 3 (or any instruction using line N3) is detected IC5 causes the address to come from bits 0 to 3 of the control word, specifying the segment. IC11 is a latch to delay mode changes until the end of a line.

The actual modes are controlled by IC2, and 8 bit by 32 word PROM. Using the 3 bits which specify the mode as an address, its contents have been programmed to cause output lines PD1 to PD4 (4 more are unused) to set various parameters throughout the display, such as the dot clock speed.

MODE	PD4321	DOT CLOCK	BYTES/LINE	LINES/RCLK	AL/GR
000	0011	6.78 Mhz	32	1	graphics
001	0110	3.39 Mhz	16	2	graphics
010	1001	13.56 Mhz	64	8	alpha
011	1010	3.39 Mhz	16	8	alpha
100	1011	6.78 Mhz	32	8	alpha
101	1101	13.56 Mhz	64	16	alpha
110	1110	3.39 Mhz	16	16	alpha
111	0111	6.78 Mhz	32	16	alpha

The truth table could have been implemented using gates, but it would have required more chips and would have resulted in an inflexible design. As a large part of the PROM is unused, with only reprogramming several other modes could be implemented, such as coarser or denser graphics.

4.11 Address Generation

Counters IC13, 14, 15 and 16 form a 13 bit presettable linear address generator, which can be broken into variable length segments with different clocks.

The various modes require 16, 32 or 64 characters per line. Thus the first part of the counter clocked once per character by CCLK must be 4, 5 or 6 bits long respectively. The remaining counters must be clocked once every 1, 2, 8 or 16 lines, by RCLK. This part of the counter varies from 9 to 7 bits, keeping the total length at 13 bits, or 8K.

IC14 contains bits 5 and 6, and the lookahead carry from IC13 is maintained using IC8, making the whole counter synchronous. IC7 is used to select the clock for bits 5 and 6, which can be CCLK or RCLK, controlled by PD1 and PD2. IC15 and IC16 are always clocked by RCLK, although still controlled by the lookahead carry based on CCLK.

At the end of each frame the EOF signal is used to preload IC15 and IC16 with the first address to be displayed from IC9 and IC10.

IC17 selects the EOL signal to signify the end of a line which may be 16, 32 or 64 bytes long. It also selects the correct dot clock DCL from the three signals available.

4.12 Microprocessor Access

When the processor wishes to address the memory it is necessary to change the address lines to the RAM. IC23 decodes the upper 3 address lines into 8 lines, one for each of the eight 8K spaces available for the RAM. Usually the Y1 output is selected via DIP switch 24, putting the RAM at 2000 to 3FFF. When an access to this RAM is detected IC19, 20, 21 and 22 change the address lines to the RAM so that they come from the processor. IC23 is gated by IC8 so that the addresses are only switched during the actual read or write operation. This is to minimize the flicker caused by the processor requests. IC22 gates the MWR' and MRD' lines so that during display the memory is not written to, and does not appear on the processor data bus.

4.13 Serial I/O Board

This board provides a 1200 baud phase encoded cassette interface and a serial I/O port with variable baud rate. Because the various timing circuits are sensitive to power supply noise, the +5 supply is derived from an on-board regulator. It is shown in Figures 13 and 14.

4.14 Baud Rate Generator and Serial Drivers

IC3 is a 307.2 KHz astable oscillator which provides the master clock. It is divided by IC4 to produce the 16x

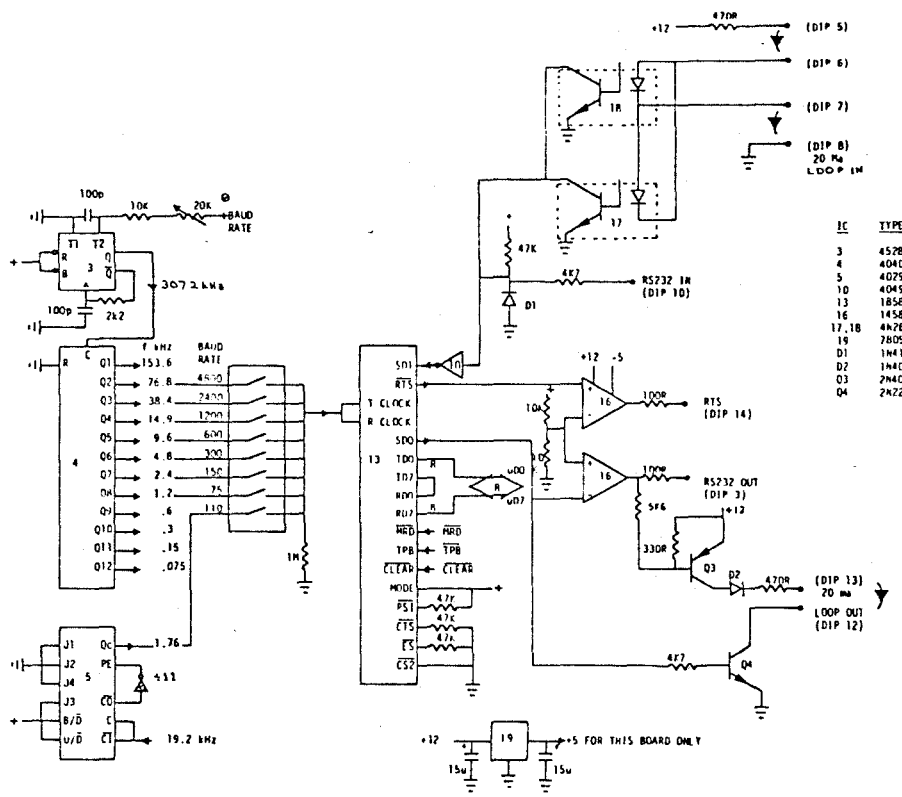


Figure 13. Serial Board - Main Port

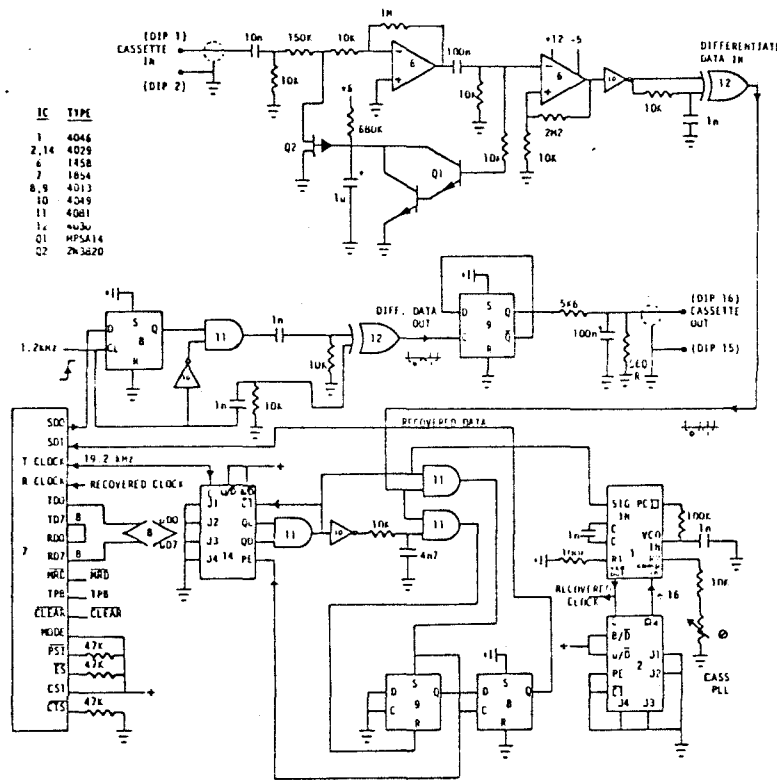


Figure 14. Serial Board - Cassette Port

frequencies required for the various baud rates. As the 1.76 KHz needed for 110 baud is not a simple division, IC5 counts from 4 to 14, dividing by 11. The baud rate desired is selected using a DIP switch which can be extended via a DIP cable to the back panel for convenience. IC13 is a CMOS UART designed for the 1802 microprocessor. In addition to the regular functions, it also controls the RTS (Request to Send), CTS (Clear to Send), PSI (Peripheral Status Interrupt) and ES (External Status) lines which normally require extra hardware. It can also generate automatic I/O based interrupt requests, but this function was not implemented.

IC17 and IC18 provide an isolated 20 ma loop input which is not sensitive to polarity. If the driving circuit is a switch (as in Teletype) the 470R resistor will provide the driving current. The 20 ma signal is OR'd with an RS232 (+/- 12v) input and passed through IC10 to the UART. The inputs are arranged so that if both are unused, the SDI line stays low, acting as a "break" signal to inform the processor.

IC16 converts the serial data out SDO and RTS lines to RS232 levels. Q3 and Q4 form the 20 ma loop output driver. By having the Q4 switch to ground and Q3 switch to power, most loops can be driven. As there is no standardization in this area, any 20 ma connection should be carefully investigated for polarity and active or passive switches.

4.15 Cassette Port

This is a 1200 baud phase encoded cassette port loosely based on the circuitry for the Processor Technology SOL-20 microcomputer. (32)

On output, the data at 1200 baud is latched in IC8 by a 1200 Hz clock from IC4. If the data is "0", the output of IC8 and IC11 remains low so that a short positive pulse appears at IC12 for every positive transition of the 1200 Hz clock counter. Thus each "0" bit appears as one half cycle at 600 Hz.

If the data is a "1", IC11 is enabled by IC8 so that IC12 produces a short pulse on both the positive and negative transitions of the 1200 Hz clock, producing a 2400 Hz signal. When this is divided by IC9, the result is one full cycle at 1200 Hz to represent a "1".

It should be noted that whether the data is "1" or "0" the data cell is 1/1200 second long, and that one transition per cell occurs for "0", with two transitions for a "1". The actual polarity of the signal is irrelevent, as the time between transitions carries the information.

On input the first part of IC6 amplifies the signal from the tape recorder, with Q1 and Q2 providing AGC action for constant level. The rest of IC6 is a high gain section

which together with IC10 produces a square wave logical level signal. IC12 and its RC network differentiate the signal to produce a short positive spike for each positive or negative transition of the signal.

IC14 is designed to count up from 0 and stop on the 12th count until reset from its PE input. It is driven from a 19.2 KHz clock (16x 1200), so that if the incoming signal was a "1", the second transition of the input should have occurred before the twelfth count (it should occur at the 8th pulse if there is no error). In this case the output of IC10 would be high (ignoring the RC deglitcher) during the second transition, causing IC9 to be set. When the third transition comes along at the start of the next cell, the IC14 has finally stopped at the twelfth count, so that IC10 is low and IC11 high, causing the data from IC9 to be clocked into IC8 and hence the UART. IC9 is also reset and IC14 cleared to count again.

If the data was a "0", IC14 has latched at 12 before the next transition arrives. In this case, IC9 stays reset, a zero is passed to IC8 and the UART, and IC14 is cleared to count again. This works well and IS simpler than it sounds.

If all is working well, the output of IC11 is a pulse with a repetition rate of 1200 Hz, locked to the input data rate. This is multiplied 16x to 19.2 KHz in the phase

lock loop IC1 and divider IC2. This recovered clock at 19.2 KHz is fed to the UART as the receive clock. This reduces the chance of read errors being made due to speed fluctuations in the cassette recorder. The fixed 19.2 KHz used by IC14 is hardly a limiting factor as there is roughly a +/- 50% leeway allowed because the transitions required for a "1" can occur up to 4 counts after the ideal 8 count position.

Although all the I/O hardware has been tested, no software was written for this board as it was not needed during development. The convenience of the EPROM programmer was found to eliminate the need for cassette storage. If the system is to be used for symbol display, the I/O functions would have to be implemented.

Appendix D gives chip and edge connector layouts for all the boards.

CHAPTER V

RESULTS AND RECOMMENDATIONS

5.1 Test Pictures

Some trial pictures were drawn with the system and these are shown in Figure 15 and Figure 16. These are actual photographs of the working display.

The symbol for telephone is shown in Figure 15 in three different sizes and was drawn using a stack of picture commands as outlined in Chapter III. In its largest size it nearly fills the screen, while the smallest size fits more than ten times in one row. The lettering underneath shows the largest size available. It can be seen that the pictures can be scaled with no loss of legibility.

Figure 16 shows a complete Bliss sentence as it might be used in a classroom, with the English word appearing under each symbol. On the last line the sentence is presented again with additional words such as "the" added, using the smallest lettering available.

5.2 Hardware Recommendations

The hardware performed well, and no major additions are required if the unit is used as a graphics peripheral. A mathematics chip could be used to speed up the circle

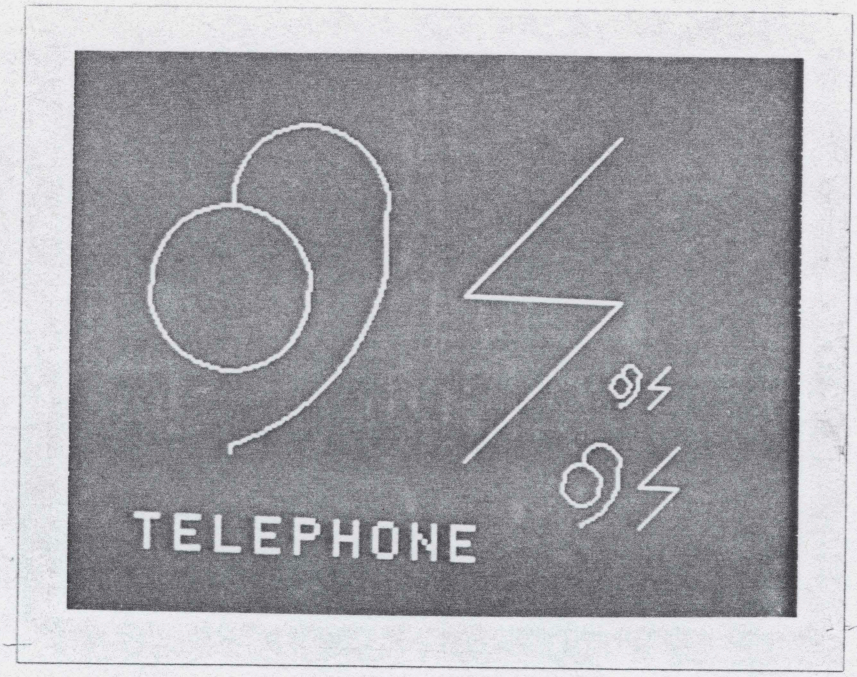


Figure 15. Scaled Symbol Display

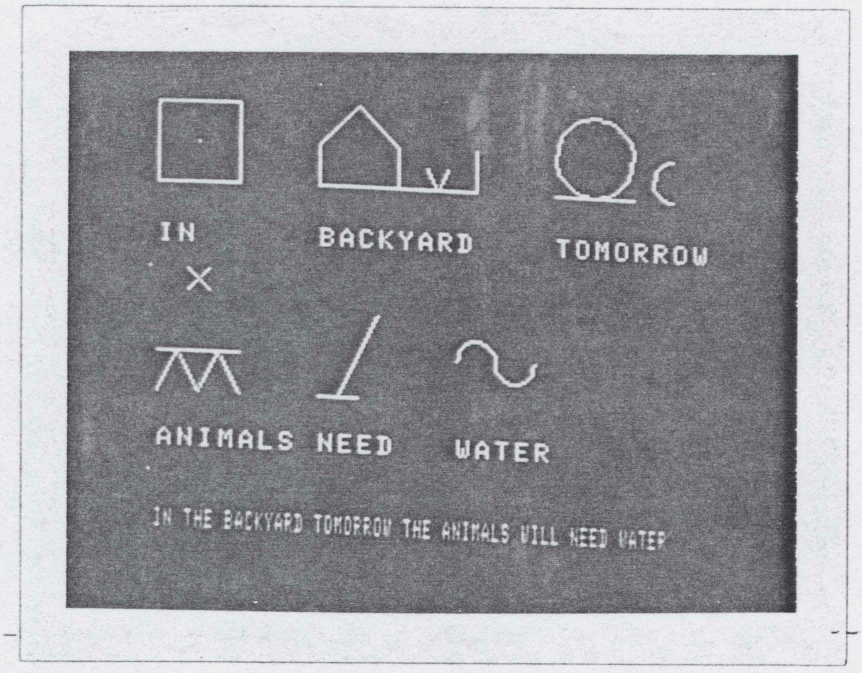


Figure 16. Complete Sentence Display

calculations, but this is not essential. If the unit is to stand alone more RAM memory, a floppy disk drive and a keyboard should be added.

5.3 Software Recommendations

To be a useful classroom tool, much more software needs to be written. The first program needed is a serial port driver and monitor program so that the machine language routines can be more easily debugged. The hexadecimal keypad on the unit is not adequate for larger programs.

The picture stack method was implemented using 7 bytes of memory for each command. This was done to allow maximum flexibility in the initial stages. It is obvious however that some commands need more space than others. A "clear screen" only needs one byte, while a "draw circle" needs all 7. A lot of memory could be saved by putting the command byte at the start of each stack, followed by only the exact amount of data required. A more complex program would be needed to unpack the command strings.

To encode the demonstration pictures they were laid out on graph paper and hand coded. A lot of repetitive work can be avoided if the picture stacks were partially computer generated. A cursor could be moved on the screen to a desired area, and then the common shapes such as boxes, semi-circles and hearts would be drawn and placed onto the

stack using a keyboard command. The finished stack would be saved on EPROM or tape under the name of the new symbol. An adjunct to this would be a program to enter alphanumeric characters without hand coding.

The scaling in Figure 15 was done by recoding the picture to the new size and placement. This does not exploit the possibilities given in Chapter III. Commands should be added for scaling a symbol and moving it on the screen.

Once the suggested cursor routines were in place, a further reduction might be made in storage requirements. When the symbols are drawn, the largest individual piece appears to be the box, as used by the symbol "in". Other sub-symbols such as the heart and the wave all fit within the box. If the corners of the box are confined to a 16 by 16 grid only four bits are required to specify them instead of eight, cutting the data space in half. It is not obvious, but all the circular segments also have their endpoints and centres on the same size and resolution grid. The advantage of the graphics method chosen is now clear. Rather than the blocks of a 16 by 16 grid being the limits of the display resolution, the points of the grid are used and high resolution lines are drawn in by the software. While the main cursor would place whole symbols on the screen, a secondary cursor would be used to move the 16 by 16 space while building a symbol.

5.4 Conclusions

The display unit was built and has adequate capabilities as shown by the sample pictures. The software however contains only the bare essentials for drawing symbols. Building the hardware and writing the code to simply drive the display took more time than was expected. To achieve the goal of a classroom instrument several thousand more lines of code are needed. The most efficient way of doing this would be to use this unit as a peripheral and do the programming in a high level language on a personal computer.

REFERENCES

1. Deich, R., P. Hodges: "Language Without Speech", Souvenir Press, London, 1977. pg. 124

2. Ibid., pg. 60.

3. Day, R. : "Communication Aids for Cerebral Palsied Children", M. Eng. Thesis, McMaster University, Hamilton, Ontario, Sept. 1976. pp. 3-18

4. Deich, pg. 105.

5. Silverman, H., S. McNaughton, B. Kates: "Provisional Dictionary", Revised edition July 1976, Blissymbolics Communication Foundation, Toronto, 1976.

6. Bliss, C.: "Semantography", Semantography Publications, Sydney, Australia, 1965.

7. Silverman, pg. 10.

8. Makita, K. : "The Rarity of Reading Disability in Japanese Children", American Journal of Orthopsychiatry, Vol. 38, 1968, p. 599.

9. Ibid., pg. 602.

10. Ibid., pg. 604.

11. Ibid., pg. 607.

12. Ibid., pg. 608.

13. Ibid., pg. 610.

14. Ibid., pg. 611.

15. Deich, pg. 599.

16. Ibid., pg. 246.

17. Ibid., pg. 137.

18. Ibid., pg. 189.

19. Deich, pg. 171.

20. Ibid., pg. 20.

21. Ibid., pg. 185.

22. Ibid., pg. 223.

23. Ibid., pg. 180.

24. Ibid., pg. 108.

25. Ibid., pg. 137.

26. Ibid., pg. 55.

27. Ibid., pg. 67.

28. Ibid., pg. 68.

29. Ibid., pg. 112.

30. Ibid., pg. 82.

31. Tekatch, E.: "Microprocessors for Industry", Training Course Manual, IEEE Hamilton Section, 1977.

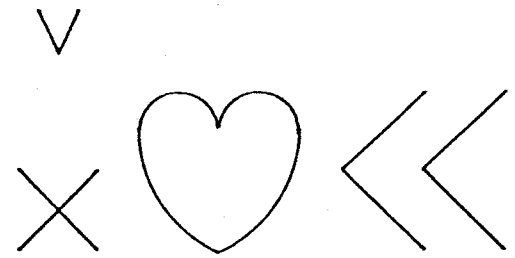
32. "SOL-20 Manual", Processor Technology Corporation, Emeryville, California, 1976.

APPENDIX A

DICTIONARY

OF

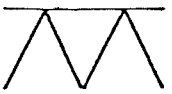
SYMBOLS USED



ANGRY

adj.

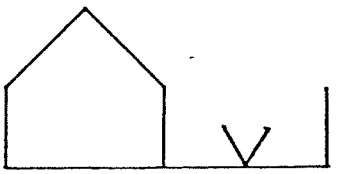
much feeling against



ANIMALS

noun

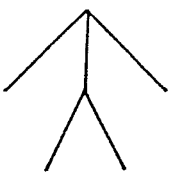
plural animal



BACKYARD

noun

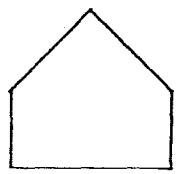
(house...)



FATHER

noun

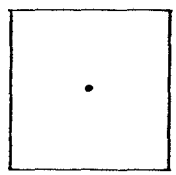
protection man



HOUSE

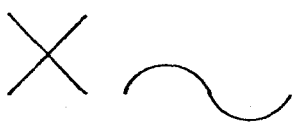
noun

protection box



IN

prep.



LAKE

noun

much water

MAN

noun

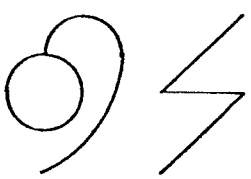
NEED

verb

(need, dependency)

PERSON

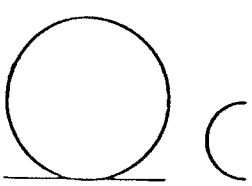
noun



TELEPHONE

noun

word electricity



TOMORROW

noun

day future

WASH

verb

water container

WATER

noun

WOMAN

noun

APPENDIX B

DETAILED SOFTWARE EXPLANATION

SUBROUTINES SUMMARIES & USE

With the exception of the first three utility programs all of the programs below are subroutines using the RCA standard call and return (SCRT) conventions found in *. When calling any program except START, R3 must be the program counter. RX can have any required value, but any CALLs or RETURNs will set it to R2.

START 1B00 - 1B30

This is an initialization routine which sets up the micro to follow the SCRT conventions, initializes the two stack pointers, formats the display for 256 graphics and clears it.

Entry conditions:
1. Jump in at 1B00.

Exit conditions:
1. 3BFF in R0 data stack pointer
2. 3AFF in R2 subroutine stack, math and I/O pointer
3. 0003 in R3 program counter on exit, required for SCRT
4. 1BE0 in R4 program counter for SCRT call program
5. 1BF2 in R5 program counter for SCRT return program
6. 2 in X sets up R2 for above uses
7. 3 in P sets up R3 as current program counter
8. The display is formatted to display 6K of memory from 2000 to 37FF.
9. The 6K screen memory is cleared to 00.
10. START jumps to 0003 when finished, expecting to find a program.

Example:

```
Add.  Code    Label Mnemonic          Comments
0000  C0 1B00 BEGIN LBR START ;Jump to initialize all
0003   ?      USER   ?        ;Start your program here
```

CALL 1BDF - 1BF0

This is an SEP subroutine used to call a subroutine under the SCRT conventions. A subroutine may call itself as long as there is a way out (no endless loop). A subroutine may have more than one entry or exit point.

Entry conditions:
1. Execute an "SEP 4" (D4) instruction to activate CALL.
2. The next two bytes must be the address of the subroutine being CALLed.

Exit conditions:
1. The current value in R6 is pushed onto the R2 stack.
2. R6 is loaded with the return address.
3. Execution continues at the CALLed address
4. R2 points to a new clear location.
5. All registers (even DF) are preserved except:
   D, X (set to 2), R2, R6. **** As these registers will be altered by every subroutine call they will not be mentioned each time. ****

RETURN 1BF1 - 1BFF

This is an SEP subroutine used to return from a program called by the SCRT method. To avoid stack crashes, there must be exactly as many RETURNs as CALLs executed by a program. The R2 stack grows two bytes with every CALL and shrinks by two with every RETURN.

Entry conditions:
1. Execute an "SEP 5" (D5)

Exit conditions:
1. R3 is loaded with the return address from R6.
2. The old R6 is pulled from the R2 stack.
3. Execution continues at the original calling program immediately after the calling instructions
4. Register use same as CALL.

Example:

```
Add.  Code    Label Mnemonic        Comments
xxxx  D4 1856       CALL S256 ;Call a graphics subroutine
xxxx+3 ?             ?        ;When done, return right here
```

SABC 1BA3 - 1BB1 (Save RA, RB & RC)

This is used to push RA, RB and RC onto the R0 data stack so they can be recalled at a later date.

Entry conditions:
1. R0 must point to a clear memory location with at least 6 more clear locations above it.
2. CALL SABC (D4 1BA3)

Exit conditions:
1. RA, RB and RC have been saved on the R0 stack.
2. All registers are unchanged except:
   R0 (decremented by 6)
3. R0 points to a (new) clear location.

RABC 1BB2 - 1BC1 (Restore RA, RB & RC)

Previously saved values for RA, RB and RC are pulled from the R0 data stack and are restored to the correct registers.

Entry conditions:
1. RA, RB and RC have been previously saved by SABC.
2. R0 is one less than the start of the stored data.
3. CALL RABC (D4 1BB2)

Exit conditions:
1. The stored values are put back into RA, RB and RC.
2. R0 is incremented by 6 to where it was before the last call of SABC.
3. All registers are unchanged except:
   R0, RA, RB and RC
4. R0 is left at a clear location.

RAS 18F4 - 18FF (Restore And Save RA, RB & RC)

This subroutine does an RABC to restore RA, RB and RC using the R0 stack, then it decrements R0 by 6 so that it is in place for another restore of the same data at a later time.

Entry conditions
1, 2 As for RABC
3. CALL RAS (D4 18F4)

Exit conditions:
1. The stored values are put back into RA, RB and RC.
2. R0 is left at its value before this subroutine.
3. All registers are unchanged except:
   RA, RB and RC

Example:

```
Add.  Code     Label  Mnemonic        Comments
xxxx   ?               ?          ;Put values in RA, RB & RC
xxxx  D4 1BA3          CALL SABC  ;Save those values in memory
xxxx   ?               ?          ;Mess up RA, RB and RC
xxxx  D4 18F4          CALL RAS   ;Get values back, fix R0 so
                                  ;you can get them later
xxxx   ?               ?          ;Mess up RA, RB and RC
xxxx  D4 1BB2          CALL RABC  ;Get values, put R0 where it
                                  ;was before this example
```

DELAY 19E4 - 19FF

This routine can be called with an inline parameter to give an uncalibrated delay, such as for switch debouncing. It will be as stable as the clock of the 1802.

Entry conditions:
1. CALL DELAY hhhh (D4 19E4 hhhh)
2. hhhh is an inline, two byte timing parameter from 0000 to FFFF. The bigger it is, the longer the delay.

Exit conditions:

1. There will of course be a delay before returning. do not adjust your micro.
2. All registers are unchanged.

MOVE 1B75 - 1BA2

This can be used to move a block of data from anywhere in memory to anywhere in memory, even in overlapping areas. All the data from M(RA) to M(RB) inclusive is moved to the block of memory beginning at M(RC). No large buffer store is required.

Entry conditions:
1. RA start address of memory to be moved
2. RB stop address of memory to be moved
3. RC start address of new location for above data
4. CALL MOVE (D4 1B75)

Exit conditions:
1. The memory move described above will be done.
2. All registers are unchanged except:
3. RA, RB, RC, RF and DF

PROM 1B31 - 1B74

This routine is called and superseded by EPROM below.

EPROM 1987 - 19CA

This is a subroutine to program 2708 EPROMs using the EPROM socket at 1C00 to 1FFF. This socket can be used for normal read only operation, as the others, but is the only one which can be used for writing. The data in M(RA) to M(RB) inclusive is put into the EPROM. RC must contain the REAL, FINAL address that the new data will occupy in memory It is not necessary to find the address from 1C00 to 1FFF.

The PROM board, when in the PROGRAM mode, will delay the entire micro with 1 ms. wait states. It takes several minutes, so be patient. If the program is run with the PROM board switch in the NORMAL position, the operation of the program can be checked quickly (no waits) with no change of data in the PROM. A PROM can be partially programmed by specifying a small data block in RA and RB. Any bit still in the "1" state can be set to a zero without UV erasing.

Entry conditions:
1. RA start address of new data (ANYWHERE in memory)
2. RB end address of new data
3. RC final address in memory where the new data will be used - ignore the location of the PROM and socket

Operation:
A - Ensure programming switch is on NORMAL.
B - Enter new data into RAM (or specify another EPROM).

C - Enter a program from 0003 which sets up RA, RB, RC and then calls EPROM.
D - Run program and the program will:
 1. Clear screen, set up for 256 graphics.
 2. Display present EPROM data in third 1/6th of screen.
 3. Display new data ONLY correctly placed inside a 1K size block of clear memory in fourth 1/6 of screen.
 4. Call PROM which:
  a) Moves present EPROM to sixth 1/6 of display RAM.
  b) Moves new data into its correct space in that RAM. At this point the bottom of the screen contains what the EPROM should look like when finished.
  c) Writes that RAM 256 times into the EPROM, programming it. In the NORMAL position this takes less than one minute with no programming taking place. In the PROGRAM position this operation takes several minutes. Go have a coffee.
 5. Display the finished EPROM in the fifth 1/6th of the screen.
 6. Return to calling program.
E - Look at the screen and check that:
 a) The old EPROM looks as you left it - is it the right one? - is the important section erased?
 b) The new data block is correct - is it the right size?
  - is it in the correct space inside the 1K block?
  - is it the right data? (some bytes CAN be recognised)
 c) This will look like a) if no programming, like d) if programming successful.
 d) The new and old data merge - does it sit where you expected?
F - When you are fully convinced step E looks O.K., set the switch to PROGRAM and run the program again.
G - As soon as the screen stops flickering turn the switch back to NORMAL, as any further writes to the EPROM socket could cause trouble. Repeat the checks in step E.

Exit conditions:
1. The EPROM will be programmed.
2. All registers are unchanged except:
   RA, RB, RC, RF and DF
3. It is not expected that EPROM will be part of any program other than the short one necessary to run it.

Example:

```
Add.  Code     Label  Mnemonic        Comments
0000  C0 1B00  BEGIN  LBR START  ;Initialize everything
0003   ?       USER    ?        ;Take as much space as needed
                                ;to set up RA, RB and RC
0020  D4 1987          CALL EPROM;Program the EPROM
0023  7B               SEQ       ;Set Q to signal the end
0024  30 24    DONE   BR DONE    ;This loop keeps the 1802 off
```

;the display for visual check

SR0+9 1A77 - 1A87 (Save RA, RB and RC, Add 9)
     This is an extension of SABC to store RA, RB and RC at a different spot on the R0 data stack and return R0 to its value before this call.

Entry conditions:
1. R0-3 to R0-9 must be clear locations.
2. CALL SR0+9 (D4 1A77)

Exit conditions:
1. RA, RB and RC have been saved on the R0 stack from R0-3 to R0-9.
2. All registers are unchanged.
3. R0 is left where it was before the call (unlike SABC).

R0-9R 1A67 - 1A76 (Subtract 9, Restore RA, RB and RC)
     This is an extension of RABC to restore RA, RB and RC from a different spot on the R0 data stack and return R0 to its value before the call.

Entry conditions:
1. RA, RB and RC have been previously saved by SR0+9 at R0-3 to R0-9.
2. CALL R0-9R (D4 1A67)

Exit conditions:
1. The stored values are put back into RA, RB and RC.
2. All registers are unchanged except:
   RA, RB and RC.
3. R0 is left where it was before the call (like RAS)

MATH SUBROUTINES

DADD 1BC2 - 1BD0 (Double precision ADDition)
     This performs a 16 bit addition, RC + RF, putting the 16 bit answer in RC and the carry in DF.

Entry conditions:
1. RC and RF contain the two 16 bit numbers to be added.
2. CALL DADD (D4 1BC2)

Exit conditions:
1. RC + RF is in RC.
2. DF contains the carry, 1 = carry, 0 = no carry.
3. All the registers are unchanged except:
   RC & DF

DSUB 1BD1 - 1BDE (Double precision SUBtraction)

This performs a 16 bit subtraction, RB - RA, putting the 16 bit answer in RC and the sign in DF.

Entry conditions:
1. RA contains the subtrahend and RB the minuend.
2. CALL DSUB (D4 1BD1)

Exit conditions:
1. RB - RA is in RF.
2. R6 is loaded with the return address.
3. DF = 0 if negative (RA > RB), DF = 1 if positive or zero
5. All the registers are preserved except:
   RF & DF

MULT 1900 - 191C (MULTiply)
This subroutine is used to multiply two 8 bit unsigned numbers, RA.0 & RB.0, returning a 16 bit unsigned answer in RA.

Entry conditions:
1. RA.0 and RB.0 contain the two numbers to be multiplied.
2. CALL MULT (D4 1900)

Exit conditions:
1. RA contains RA.0 x RB.0.
2. There can never be a carry.
3. The registers are unchanged except for:
   R8.1, RA & DF

DIV 191D - 1949 (DIVide)
This is an unsigned routine to divide RA.0 into RB, returning the answer in RB and twice the remainder in RA.1.

Entry conditions:
1. RA.0 contains the 8 bit divisor.
2. RB contains the 16 bit dividend.
3. CALL DIV (D4 191D)

Exit conditions:
1. RB holds the 16 bit quotient. Division by zero will return 0000 or 0001, but no error flag.
2. RA.1 contains twice the remainder.
3. The registers are unchanged save:
   R8, RA.1, RB & DF

SQRT 194A - 1969 (SQuare RooT)
An iterative technique is used to find the square root of RC. If RC >= FE00, the answer FF is automatically returned in RA.0 with no calculation. Only 8 iterations are required for an 8 bit answer accurate to +/- 01 for RC<FE00.

Entry conditions:
1. RC has the 16 bit number to be SQRTed.
2. CALL SQRT (D4 194A)

Exit conditions:
1. The 8 bit square root of RC is in RA.0.
2. All registers are unchanged except:
   R7.0, R8, RA, RB & DF

ABSQ1 196A - 1986 (ABsolute SQuare)
ABSQ2 196E - 1986
MABAF 1977 - 1986 (Multiply RA.0 x RB.0, Add into RF)
ADDAF 197A - 1986 (ADD RA + RF into RF)

This is one long routine with several useful entry points. During execution ABSQ1 leads into ABSQ2 and so on.

Entry conditions for ADDAF:
1. RA and RF contain two 16 bit numbers to be added.
2. CALL ADDAF (D4 197A)

Exit conditions:
1. RA + RF is in RF, the carry is in DF.
2. All registers are unchanged except:
   RF & DF

Entry conditions for MABAF:
1. RA.0 and RB.0 contain two 8 bit numbers to be multiplied into a 16 bit answer in RA, which is then added to RF.
2. CALL MABAF (D4 1977)

Exit conditions:
1. (RA.0 x RB.0) is in RA.
2. (RA.0 x RB.0) + RF is in RF, carry in DF.
3. All registers are unchanged except:
   R8.1, RA, RF & DF

Entry conditions for ABSQ2:
1. RA.0 and RB.0 contain two 8 bit numbers.
2. CALL ABSQ2 (D4 196E)

Exit conditions:
1. [RA.0 - RB.0]^2 is in RA
2. [RA.0 - RB.0]^2 + RF is in RF, carry in DF.
3. All registers are unchanged but:
   R8.1, RA, RB.0, RF & DF

Entry conditions for ABSQ1:
1. RA.0 and RB.0 contain two 8 bit numbers.
2. CALL ABSQ1 (D4 196A)

Exit condtions:

1. [RA.0 - RB.0]^2 is in RA
2. [RA.0 - RB.0]^2 is in RF
3. All registers are unchanged except:
   R8.1, RA, RB.0, RF & DF

RSQ 1A5B - 1A66
     This is a small slave routine for RSQD.

Entry conditions:
1. Two 8 bit numbers in RA.0 and RB.0, Q reset.
2. RF may or may not contain a 16 bit number.
2. CALL RSQ (D4 1A5B)

Exit conditions:
1. [RA.0 - RB.0]^2 into RA.
2. [RA.0 - RB.0]^2 + RF into RF, carry into DF.
3. If there is a carry, Q will be set. It will not be reset
   if there is no carry.
4. An RAS is performed to get stacked values and put them
   back into RA, RB & RC.
5. All registers are unchanged except:
   R8.1, RA, RB, RC, RF, DF & Q

RSQD 1A00 - 1A5A (Radius SQuareD)
     This is really a graphics program used to calculate a radius for a circle given two points on the circumference and the centre. Although the centre and one other point should be sufficient, both are used to produce an average value to overcome the limited precision integer math used. R^2 and R are put on the R0 data stack.

Entry conditions:
1. RA, RB and RC should have been saved with SABC. They
   contain the following information:
   P1   RA.1 = X1   RA.0 = Y1
   P2   RB.1 = X2   RB.0 = Y2
   PC   RC.1 = XC   RC.0 = YC
2. R0 should be at a free location 1 less than the saved
   registers. R0 to R0-2 should be free.
3. CALL RSQD (D4 1A00)

Exit conditions:
1. R^2 is calculated from:
R^2 = ([X1-XC]^2 +[X2-XC]^2 +[Y1-YC]^2 +[Y2-YC]^2 +01)/2
2. The high 8 bits (HiR^2) are in M(R0).
   The low 8 bits (LoR^2) are in M(R0-1).
   R is in M(R0-2).
   R0 is left where it was, but M(R0) is no longer free.
3. All the registers are unchanged except:
   R7.0, R8, RA, RB, RC, RF DF and Q

YWOC 19CB - 19E3 (Y value WithOut Centre)
This calculates the Y value of a circle asumming the centre is at 0,0.

Entry conditions:
1. RSQD has been called and R^2 and R are on the R0 stack.
2. [X - XC]^2 is in RA & RF. This can be found with ABSQ2.
3. CALL YWOC (D4 19CB)

Exit conditions:
1. Y^2 = (R^2 - [X - XC]^2) is in RC.
2. Y = SQRT(RC) is in RA.0.
3. If (R^2 - [X - XC]^2) is negative, RC is set to 0000 and RA.0 is set to 00.
2. All registers are unchanged except:
R7.0, R8, RA, RB, RC, RF and DF

GRAPHICS SUBROUTINES

ZSCR 1856 - 1868 (Zero SCReen memory)
This subroutine writes 00 into the memory from 2000 to 37FF. If the screen is set for 256 graphics this will blank the entire screen. Any alphanumeric portion will show "@".

Entry conditions:
1. CALL ZSCR (D4 1856)

Exit conditions:
1. M(2000) to M(37FF) contains 00.
2. All the registers are unchanged except:
RF

CSCR 1869 - 187C (Clear SCReen memory)
Each byte from M(2000) to M(37FF) is complemented. If the screen is set for 256 graphics a black image will become white and vice versa. Any alphanumeric data will be completely garbled.

Entry conditions:
1. CALL CSCR (D4 1869)

Exit conditions:
1. M(2000) to M(37FF) contain the complement of their previous values.....F6 would become 09.
2. All the registers are preserved except:
RF

S256 187D - 1891 (Set 256 graphics)
This subroutine formats all twelve portions of the

screen to display 256 by 196 density graphics. The display is set to begin at M(2000) and covers 6K to 37FF in this mode. No memory is modified, just the display format.

Most of the following routines require the screen to be formatted with this routine. With care, regions of the screen can be reformatted for alphanumerics.

Entry conditions:
1. CALL S256 (D4 187D)

Exit conditions:
1. The screen is formatted as required.
2. The registers are unchanged except for:
   RF

AM256 1800 - 1833 (Address & Mask for 256 graphics)

In the 256 graphics mode, each bit in 6K of memory is displayed on the screen as a light or dark dot. To read or write any dot in that memory it is neccessary to know both the memory address of the byte and the particular bit in the byte. The top left of the screen is M(2000) and the addresses increase from left to right and top to bottom This subroutine allows you to enter with an X coordinate in RD.0 and a Y coordinate in RE.0. (each <= FF and >= 0), a more convenient format. The bottom left of the screen is the origin. The maximum Y value displayed is BF, and any values larger than this are chopped to BF causing no error. X values to FF are correctly displayed.

The program returns the true memory address of the required byte in RC and a bit mask in RF.1. RF.1 will be all 0's except for the required bit which will be 1.

Entry conditions:
1. RD.0 contains an X coordinate. 00 <= X <= FF
2. RE.0 contains a Y coordinate. 00 <= Y <= FF
   Y values > BF are displayed as BF.
3. CALL AM256 (D4 1800)

Exit conditions:
1. RC holds the memory address of the desired word.
2. RF.1 is all 0's except for the desired bit.
3. The registers are unchanged save:
   RC, RF & DF

GBIT 1834 - 183F (Get BIT)

This subroutine will retrieve a particular bit from the screen memory once AM256 is used to find the location.

Entry conditions:
1. AM256 has JUST been called to find the address and mask.
2. CALL GBIT (D4 1834)

Exit conditions:
1. Q is set if the bit was 1, reset if it was 0.
2. All registers are unchanged except:
   Q

SBIT 1840 - 184D (Set BIT)
This is used to set or reset a bit on the screen.

Entry conditions :
1. AM256 has JUST been called to find the address and mask.
2. Q = 1 to set the bit, Q = 0 to reset it.
3. CALL SBIT (D4 1840)

Exit conditions:
1. The desired bit has been set or reset.
2. All registers are unchanged.

CBIT 184E - 1855 (Complement Bit)
This is used to complement a bit on the screen.

Entry conditions:
1. AM256 has JUST been called to find the address and mask.
2. CALL CBIT (D4 184E)

Exit conditions:
1. The desired bit has been complemented.
2. All registers are unchanged.

DLINE 1892 - 18F2 (Draw LINE)
This is the main graphics program and is used to draw a straight line on the screen given the two endpoints. To avoid solving "y = mx + b", which causes problems for vertical lines, X and Y are slowly incremented in a ratio proportional to the desired slope.

This routine uses AM256, so the origin and coordinate limits are the same. R9.0 is checked and can be used as a flag to use a routine other than AM256, although no others have been written. This code has been bypassed for the present but can easily be reactivated.

Any line with a Y coordinate >BF will appear at the top of the screen with the correct X coordinates.

Entry conditions:
1. RA and RB contain the following information:
   P1 RA.1 = X1 RA.0 = Y1
   P2 RB.1 = X2 RB.0 = Y2
   There is no difference between the endpoints - swapping RA and RB draws the same line.
2. R9.0 could be used as a flag to a different address and mask routine, but this code is temporarily bypassed.

3. CALL DLINE (D4 1892)

Exit conditions:
1. The desired line has been drawn on the screen.
2. All the registers are unchanged except:
   R7, R8, RC, RD.0, RE.0, RF, DF and Q

DRAW 1A88 - 1AC7
     This is a slave routine for CIRCL.

CIRCL 1500 - 158F (CIRCLe)
     This is the ultimate routine to draw full circles or arcs on the screen. The circle is drawn from P1 to P2 in a counterclockwise direction. If P1 = P2 a full circle will be drawn. If they are different an arc will be drawn. Any design using this routine should be laid out on graph paper so that P1 and P2 are the same distance from PC. Small errors are absorbed by the program, as it calculates the average radius before drawing, and explicitly draws lines from P1 and P2 to the actual arc drawn.
     If the program does not stop drawing, P1, P2 or PC should be changed by +/- 01 in either the X or Y values.

Entry conditions:
1. RA, RB and RC contain the following information:
   P1 RA.1 = X1 RA.0 = Y1
   P2 RB.1 = X2 RB.0 = Y2
   PC RC.1 = XC RC.0 = YC
   There is a difference between the endpoints - swapping them will draw the part of the circle which the other configuration did not, with the same centre and radius.
2. M(R0) to M(R0-0F) must be free locations.
3. CALL CIRCL (D4 1500)

Exit conditions:
1. The required arc or circle has been drawn.
2. R0 is returned to its original value.
3. All registers are unchanged except:
   R7, R8, RA, RB, RC, RD, RE, RF, Q and DF

PICT 1480 - 1497 (PICTure)
     This powerful section of code is used to pull graphics data and graphics instructions from a "picture stack" to draw pictures without writing explicit programs. The format of a section of the picture stack is:

byte 1	RA.0	data as required	All the data bytes
byte 2	RA.1	" " "	must be present as
byte 3	RB.0	" " "	space holders even
byte 4	RB.1	" " "	if not required by
byte 5	RC.0	" " "	the instruction.
byte 6	RC.1	" " "	

byte 7 R3.0 low address byte of graphic instruction in this page (1400 - 14FF)

The first 6 bytes are data stacked in the same way as SABC. Each time PICT is run the next 6 bytes are loaded in RA, RB and RC. The 7th byte is a low address which is put into the program counter causing a short branch. If "2F" was in the 7th position, execution would continue at 142F. When a graphic instruction is finished there is a short branch back to PICT to fetch the next 7 byte unit.

On entry R1 should contain an address 1 less than the start of the picture stack. Each time a 7 byte unit is finished, R1 has been incremented by 7.

Entry conditions:
1. M(R1+1) should be the start of a picture stack.
2. R0 should be in a clear area.
1. CALL PICT (D4 1480)

Exit conditions:
1. The graphic instructions in the picture stack have been carried out.
2. Since this program can conceivably call any other subroutine, no guarantees can be made about any of the registers. The programs should be written however so R0 and R2 are in their original state when PICT is done.

The remaining sections are the graphic instructions which have been written. There is room for several more, but they must reside in 1400 to 14FF. They are NOT subroutines, but are branched to, from PICT, and branch back when done, with the exception of PIRE.

PILI 1400 - 1407 (PIcture LIne)

This calls DLINE to draw a line from RA to RB. Any data in RC is ignored.

Entry conditions:
1. RA and RB contain two X-Y coordinates as required by DLINE.
2. Byte 7 of a picture stack unit is 00.

Exit conditions:
1. The required line is drawn.
2. Execution continues at PICT.

PICI 1408 - 140F (PIcture CIrcle)

This graphic instruction calls CIRCL to draw a counclockwise circle from P1 (RA) to P2 (RB) with centre PC (RC) as specified in the CIRCL instructions.

Entry conditions:

1. RA, RB & RC contain the data required by CIRCL.
2. Byte 7 of a picture stack unit is 08.

Exit conditions:
1. The required curve is drawn.
2. Execution continues at PICT.

PIZE 1410 - 1417 (PIcture ZEro)
This calls ZSCR to clear the screen.

Entry conditions :
1. Byte 7 of a picture stack unit is 10.

Exit conditions:
1. The screen is erased.
2. Execution continues at PICT.

PICO 1418 - 141F (PIcture COmplement)
This calls CSCR to complement the entire screen.

Entry conditions:
1. Byte 7 of a picture stack unit is 18.

Exit conditions:
1. The entire picture is complemented, black to white, etc.

PIDE 1420 - 142A (PIcture DElay)
This calls DELAY to give a delay in execution which is proportional to RA>

Entry conditions:
1. 0000 <= RA <= FFFF
2. Byte 7 of a picture stack unit is 20.

Exit conditions:
1. There is a delay before another picture element is used.
2. Exectuion continues at PICT.

PIRE 142F (PIcture REturn)
This is a single byte (D5) to get out of PICT and return to the program that called it. If this is not the last instruction of a picture the program will run off into memory and do strange things.

Entry conditions:
1. All the desired drawing for this run of PICT has been accomplished.
2. Byte 7 of a picture stack unit is 2F.

Exit conditions:
1. A RETURN is executed, and execution continues from where

PICT was called.
2. It is the user's responsibility that all stacks are in usable condition.

PICS 1430 - 1439 (PIcture Call Stack)
This graphic insruction allows whole picture stacks to be called by other picture stacks. This is useful if a cer- group of instructions (say a house drawing) is required by many other pictures. To be used, the picture stack doing the calling must have the address minus 1 of the new stack in RA. The current picture stack in R1 is pushed onto the R2 stack and RA is put into R1 as the new picture stack.

Entry conditions:
1. M(RA+1) is the start of a picture subroutine stack.
2. The picture subroutine stack must end with a PIRS, not a PIRE.
3. Byte 7 of a picture stack unit is 30.

Exit conditions:
1. The current picture stack has been saved on the R2 stack
2. RA is entered as the new picture stack.
3. A PIRE must not be executed until a PIRS has been done for every PICS done.
4. Execution contiunes at PICT, but using the new picture.

PIRS 143A - 1442 (PIcture Return Stack)
This must be the last instruction in a picture subroutine stack. It pulls the old picture stack pointer from the R2 stack and puts it back into R1.

Entry conditions:
1. The picture currently being drawn must be a picture subroutine called by PICS.
2. Byte 7 of the subroutine picture stack unit is 3A.

Exit conditions:
1. The last value of the picture stack pointer is pulled from the R2 stack and restored to R1.
2. Execution continues at PICT.

PIFO 1448 - 144F (PIcture FOrmat)
This is used to format the screen for other than 256 graphics while drawing a picture. This can be used to add alphanumeric areas to a picture, or to "hide" what is being drawn until it is ready.
The 12 regions of the screen can all be formatted separately and the starting address can be varied. If the screen is set for 256 graphics, M(2000) is really the only choice.
The various format codes are explained in the hardware

chapter.

Entry conditions:
1. RA.0 contains the format word to be sent to the display.
2. Byte 7 of the picture stack unit is 48.

Exit conditions:
1. The screen is formatted as required.
2. Execution continues at PICT.

PIMO 1450 - 1457 (PIcture MOve)
This calls the MOVE subroutine to move M(RA) to M(RB) to the block starting at M(RC). This could be used to move alphanumeric data directly to the screen, or even to swap picture stacks.

Entry conditions:
1. RA contains the start address of the data to be moved.
2. RB contains the last address of the data.
3. R3 contains the start address of the new location.
4. Byte 7 of a picture stack unit is 50.

Exit conditions:
1. The desired move has been accomplished.
2. Execution continues at PICT.

APPENDIX C

SOFTWARE CODE

UTILITY PROGRAMS

```
Add.  Code     Label  Mnemonic          Comments
1B00  F8 1B    START  LDI 1B      ;Load 1B06 in R3 for use as
1B02  B3              PHI 3       ;the program counter (RP)
1B03  F8 06           LDI 06      ;
1B05  A3              PLO 3       ;
1B06  D3              SEP 3       ;Make R3 be RP
1B07  F8 FF           LDI FF      ;
1B09  A0              PLO 0       ;
1B0A  A2              PLO 2       ;
1B0B  F8 3B           PHI 1       ;Initialize the registers
1B0D  B0              PHI 8       ;as required.
1B0E  F8 3A           PLO C       ;
1B10  B2              PHI B       ;3BFF  R1  data stack
1B11  F8 1B           LDI 01      ;3AFF  R2  subroutine stack
1B13  B4              PLO 4       ;1BE0  R4  CALL program
1B14  B5              PHI 4       ;1BF2  R5  RETURN program
1B15  F8 E0           PHI 5       ;
1B17  A4              PLO A       ;
1B18  F8 F2           LDI 02      ;
1B1A  A5              PHI A       ;
1B1B  E2              SEX 2       ;Make R2 be RX
1B1C  D4 187D         CALL S256   ;Format screen for graphics
1B1F  D4 1856         CALL ZSCR   ;Fill screen memory with 00
1B22  C0 0003         LBR USER    ;Begin user's program
1B25  00              IDL         ;1B25 to 1B30 reserved for
1B30  00              IDL         ;expansion of START
**************************************************************
Add.  Code     Label  Mnemonic          Comments
1BDF  D3       OUTC   SEP 3       ;Go to called program
1BE0  E2       CALL   SEX 2       ;Ensure R2 stack
1BE1  96              GHI 6       ;Push R6 to stack
1BE2  73              STXD        ;
1BE3  86              GLO 6       ;
1BE4  73              STXD        ;Leave stack at free location
1BE5  93              GHI 3       ;Put R3 into R6
1BE6  B6              PHI 6       ;
1BE7  83              GLO 3       ;
1BE8  A6              PLO 6       ;
1BE9  46              LDA 6       ;Get called address from
1BEA  B3              PHI 3       ;M(R6), M(R6+1)
1BEB  46              LDA 6       ;
1BEC  A3              PLO 3       ;
1BED  30 DF           BR OUTC     ;
1BEF  00              IDL         ;
1BF0  00              IDL         ;
**************************************************************
Add.  Code     Label  Mnemonic          Comments
```

```
1BF1  D3       OUTR  SEP 3      ;Return to calling program
1BF2  E2       RET   SEX 2      ;Ensure R2 stack
1BF3  96             GHI 6      ;Put R6 into R3
1BF4  B3             PHI 3      ;
1BF5  86             GLO 6      ;
1BF6  A3             PLO 3      ;
1BF7  60             IRX        ;Point to data
1BF8  72             LDXA       ;Pull R6 from stack
1BF9  A6             PLO 6      ;
1BFA  F0             LDX        ;Leave stack at free location
1BFB  B6             PHI 6      ;
1BFC  30 F1          BR OUTR    ;
1BFE  00             IDL        ;
1BFF  00             IDL        ;
*****************************************************************
Add.  Code     Label Mnemonic          Comments
1BA3  E0       SABC  SEX 0      ;Set up for R0 stack
1BA4  9C             GHI C      ;Push RC onto stack
1BA5  73             STXD       ;
1BA6  8C             GLO C      ;
1BA7  73             STXD       ;
1BA8  9B             GHI B      ;Push RB onto stack
1BA9  73             STXD       ;
1BAA  8B             GLO B      ;
1BAB  73             STXD       ;
1BAC  9A             GHI A      ;Push RA onto stack
1BAD  73             STXD       ;
1BAE  8A             GLO A      ;
1BAF  73             STXD       ;Leave stack at free location
1BB0  D5             RETURN     ;Return restores R2 stack
1BB1  00             IDL        ;
*****************************************************************
Add.  Code     Label Mnemonic          Comments
1BB2  E0       RABC  SEX 0      ;Set up for R0 stack
1BB3  60             IRX        ;Point to data
1BB4  72             LDXA       ;Pull RA from stack
1BB5  AA             PLO A      ;
1BB6  72             LDXA       ;
1BB7  BA             PHI A      ;
1BB8  72             LDXA       ;Pull RB from stack
1BB9  AB             PLO B      ;
1BBA  72             LDXA       ;
1BBB  BB             PHI B      ;
1BBC  72             LDXA       ;Pull RC from stack
1BBD  AC             PLO C      ;
1BBE  F0             LDX        ;Leave at free location
1BBF  BC             PHI C      ;
1BC0  D5             RETURN     ;Return restores R2 stack
1BC1  00             IDL        ;
*****************************************************************
Add.  Code     Label Mnemonic          Comments
```

```
18F4  D4 1BB2 RAS    CALL RABC ;Pull RA, RB and RC from R0
18F7  20             DEC 0     ;stack, but then decrement R0
18F8  20             DEC 0     ;back to data for another
18F9  20             DEC 0     ;pull of the same data
18FA  20             DEC 0     ;
18FB  20             DEC 0     ;
18FC  20             DEC 0     ;
18FD  D5             RETURN    ;
18FE  00             IDL       ;
18FF  00             IDL       ;
*************************************************************
Add.  Code     Label  Mnemonic         Comments
19E4  8F       DELAY  GLO F      ;Push RF onto R2 stack so it
19E5  73              STXD       ;isn't changed
19E6  9F              GHI F      ;
19E7  52              STR 2      ;
19E8  46              LDA 6      ;Get 2 byte in-line timing
19E9  BF              PHI F      ;parameter from calling
19EA  46              LDA 6      ;program into RF
19EB  AF              PLO F      ;
19EC  8F       DEL1   GLO F      ;
19ED  C6              LSNZ       ;Skip if RF.0 not 00
19EE  9F              GHI F      ;Else check RF.1 as well
19EF  E2              SEX 2      ;(fast NOP)
19F0  32 F5           BZ DEL2    ;Done if both were 00
19F2  2F              DEC F      ;If not: decrement RF
19F3  30 EC           BR DEL1    ;and wait some more
19F5  72       DEL2   LDXA       ;Pull RF from stack
19F6  BF              PHI F      ;
19F7  F0              LDX        ;
19F8  AF              PLO F      ;
19F9  D5              RETURN     ;
19FA  00              IDL        ;19FA to 19FF blank
19FF  00              IDL        ;
*************************************************************
Add.  Code     Label  Mnemonic         Comments
1B75  D4 1BD1  MOVE   CALL DSUB  ;Put RB-RA into RF, the num-
1B78  8A              GLO A      ;ber of bytes to move
1B79  73              STXD       ;Find the sign of RC-RA
1B7A  60              IRX        ;
1B7B  8C              GLO C      ;
1B7C  F7              SM         ;
1B7D  9A              GHI A      ;
1B7E  73              STXD       ;
1B7F  60              IRX        ;
1B80  9C              GHI C      ;
1B81  77              SMB        ;
1B82  33 90           BPZ MOV2   ;If RC >= RA, use MOV2 method
1B84  1F              INC F      ;Correct initial byte count
1B85  2F       MOV1   DEC F      ;Decrement byte count
1B86  4A              LDA A      ;Get data, advance pointer
```

```
1B87  5C             STR C      ;Store it in new place
1B88  1C             INC C      ;Advance that pointer
1B89  9F             GHI F      ;Check if done
1B8A  3A 85          BNZ MOV1   ;
1B8C  8F             GLO F      ;
1B8D  3A 85          BNZ MOV1   ;
1B8F  D5             RETURN     ;Method 1 finished
1B90  D4 1BC2 MOV2   CALL DADD  ;Put RF+RC into RC=stop point
1B93  EC             SEX C      ;Use RC as RX for move
1B94  1F             INC F      ;Correct initial byte count
1B95  2F       MOV3  DEC F      ;Decrement byte count
1B96  0B             LDN B      ;Get data
1B97  2B             DEC B      ;Decrement old pointer
1B98  73             STXD       ;Store, decrement new pointer
1B99  9F             GHI F      ;Check if done
1B9A  3A 95          BNZ MOV3   ;
1B9C  8F             GLO F      ;
1B9D  3A 95          BNZ MOV3   ;
1B9F  E2             SEX 2      ;
1BA0  D5             RETURN     ;Method 2 finished
1BA1  00             IDL        ;
1BA2  00             IDL        ;
*************************************************************
Add.  Code     Label  Mnemonic          Comments
1B31  D4 1BA3  PROM   CALL SABC ;Save RA, RB & RC
1B34  F8 00           LDI 00    ;
1B36  AA              PLO A     ; 1C00 into RA  PROM start
1B37  AC              PLO C     ; 1FFF into RB  PROM end
1B38  F8 1C           LDI 1C    ; 3400 into RC  RAM start
1B3A  BA              PHI A     ;
1B3B  F8 1F           LDI 1F    ;
1B3D  BB              PHI B     ;
1B3E  F8 FF           LDI FF    ;
1B40  AB              PLO B     ;
1B41  F8 34           LDI 34    ;
1B43  BC              PHI C     ;
1B44  D4 1B75         CALL MOVE ;Move PROM to RAM to modify
1B47  D4 1BB2         CALL RABC ;Restore RA, RB & RC
1B4A  9C              GHI C     ;Subtract 04 from RC.1 until
1B4B  BC       PRO1   PHI C     ;it is an address in 1K
1B4C  FF 04           SMI 04    ;
1B4E  C7              LSNF      ;
1B4F  30 4B           BR PRO1   ;
1B51  9C              GHI C     ;Get new RC.1 and add 34 to
1B52  FC 34           ADI 34    ;get address in 1K from 3400
1B54  BC              PHI C     ;
1B55  D4 1B75         CALL MOVE ;Move new data to RAM at 3400
1B58  F8 00           LDI 00    ;Zero RD.0 to make 256 passes
1B5A  AD              PLO D     ;
1B5B  F8 00    PRO2   LDI 00    ; 3400 into RA  RAM start
1B5D  AA              PLO A     ; 37FF into RB  RAM end
```

```
1B5E  AC             PLO C      ; 1C00 into RC  PROM start
1B5F  F8 34          LDI 34     ;
1B61  BA             PHI A      ;The RAM contains the old
1B62  F8 37          LDI 37     ;PROM data modified with the
1B64  BB             PHI B      ;desired changes
1B65  F8 FF          LDI FF     ;
1B67  AB             PLO B      ;
1B68  F8 1C          LDI 1C     ;
1B6A  BC             PHI C      ;
1B6B  D4 1B75        CALL MOVE  ;Move RAM to PROM 256 times
1B6E  2D             DEC D      ;Decrement pass counter
1B6F  8D             GLO D      ;
1B70  3A 5B          BNZ PRO2   ;Check if done
1B72  D5             RETURN     ;
1B73  00             IDL        ;
1B74  00             IDL        ;
*****************************************************************
Add.  Code    Label  Mnemonic          Comments
1987  D4 1BA3 EPROM  CALL SABC  ;Save RA, RB & RC
198A  D4 187D        CALL S256  ;Set up screen for graphics
198D  D4 1856        CALL ZSCR  ;Clear screen
1990  F8 00          LDI 00     ;
1992  AA             PLO A      ; 1C00 into RA  PROM start
1993  AC             PLO C      ; 1FFF into RB  PROM end
1994  F8 1C          LDI 00     ; 2800 into RC  RAM start
1996  BA             PHI A      ;
1997  F8 1F          LDI 1F     ;
1999  BB             PHI B      ;
199A  F8 FF          LDI FF     ;
199C  AB             PLO B      ;
199D  F8 28          LDI 28     ;
199F  BC             PHI C      ;
19A0  D4 1B75        CALL MOVE  ;Move PROM to RAM for display
19A3  D4 18F4        CALL RAS   ;Restore and save RA, RB & RC
19A6  9C             GHI C      ;Convert real address in RC
19A7  FA 03          ANI 03     ;to 1K relative address from
19A9  FC 2C          ADI 2C     ;2C00
19AB  BC             PHI C      ;
19AC  D4 1B75        CALL MOVE  ;Move new data for display
19AF  D4 1BB2        CALL RABC  ;Restore RA, RB & RC
19B2  D4 1B31        CALL PROM  ;Program the PROM
19B5  F8 00          LDI 00     ;
19B7  AA             PLO A      ; 1C00 into RA  PROM start
19B8  AC             PLO C      ; 1FFF into RB  PROM end
19B9  F8 1C          LDI 1C     ; 3000 into RC  RAM start
19BB  BA             PHI A      ;
19BC  F8 1F          LDI 1F     ;
19BE  BB             PHI B      ;
19BF  F8 FF          LDI FF     ;
19C1  AB             PLO B      ;
19C2  F8 30          LDI 30     ;
```

```
19C4  BC             PHI C      ;
19C5  D4 1B75        CALL MOVE  ;Move finished PROM to RAM
19C8  D5             RETURN     ;for display and comparision
19C9  00             IDL        ;
19CA  00             IDL        ;
*************************************************************
Add.  Code     Label  Mnemonic         Comments
1A67  80       R0-9R  GLO 0       ;R0 - 09 into R0
1A68  FF 09           SMI 09      ;Move R0 pointer up by 9 to
1A6A  A0              PLO A       ;get to different data
1A6B  90              GHI 0       ;
1A6C  7F 00           SMBI 00     ;
1A6E  B0              PHI 0       ;
1A6F  D4 1BB2         CALL RABC   ;Get that data, incrementing
1A72  10              INC 0       ;R0 by 6
1A73  10              INC 0       ;Add 3 more to restore R0 to
1A74  10              INC 0       ;original position
1A75  D5              RETURN      ;
1A76  00              IDL         ;
*************************************************************
Add.  Code     Label  Mnemonic         Comments
1A77  20       SR0+9  DEC 0       ;Move R0 up by 3 to get to
1A78  20              DEC 0       ;clear space
1A79  20              DEC 0       ;
1A7A  D4 1BA3         CALL SABC   ;Save RA, RB & RC, decremen-
1A7D  80              GLO 0       ;ting R0 by 6 more
1A7E  FC 09           ADI 09      ;
1A80  A0              PLO 0       ;R0 + 09 into R0 to restore
1A81  90              GHI 0       ;R0 to original position
1A82  7C 00           ADCI 00     ;
1A84  B0              PHI 0       ;
1A85  D5              RETURN      ;
1A86  00              IDL         ;
1A87  00              IDL         ;
*************************************************************
```

MATH PROGRAMS

```
Add.  Code     Label  Mnemonic         Comments
1BC2  8C       DADD   GLO C       ;Put RC + RF into RC, carry
1BC3  73              STXD        ;bit into DF, both preserved
1BC4  60              IRX         ;through return
1BC5  8F              GLO F       ;
1BC6  F4              ADD         ;
1BC7  AC              PLO C       ;
1BC8  9C              GHI C       ;
1BC9  73              STXD        ;
1BCA  60              IRX         ;
1BCB  9F              GHI F       ;
1BCC  74              ADC         ;
```

```
1BCD  BC            PHI C       ;
1BCE  D5            RETURN      ;
1BCF  00            IDL         ;
1BD0  00            IDL         ;
*************************************************************
Add.  Code     Label Mnemonic         Comments
1BD1  8A       DSUB  GLO A       ;Put RB - RA into RF, sign
1BD2  73             STXD        ;into DF (0 = borrow), both
1BD3  60             IRX         ;preserved through return
1BD4  8B             GLO B       ;
1BD5  F7             SM          ;
1BD6  AF             PLO F       ;
1BD7  9A             GHI A       ;
1BD8  73             STXD        ;
1BD9  60             IRX         ;
1BDA  9B             GHI B       ;
1BDB  77             SMB         ;
1BDC  BF             PHI F       ;
1BDD  D5             RETURN      ;
1BDE  00             IDL         ;
*************************************************************
Add.  Code     Label Mnemonic         Comments
1900  F8 00    MULT  LDI 00      ; 0 into DF  carry bit
1902  FE             SHL         ; 00 into RA.1  partial
1903  BA             PHI A       ; product register
1904  F8 09          LDI 09      ; 09 into R8.1  pass counter
1906  A8             PLO 8       ;
1907  9A       MUL1  GHI A       ;Shift RA right to:
1908  76             RSHR        ; 1. Put RA.0 lsb into DF
1909  BA             PHI A       ; 2. Shift partial products
190A  8A             GLO A       ;    from RA.1 to RA.0
190B  76             RSHR        ;note: partial products push
190C  AA             PLO A       ;out original RA.0 value
190D  28             DEC 8       ;Decrement pass counter,
190E  88             GLO 8       ;check it,
190F  C6             LSNZ        ;skip for more,
1910  D5             RETURN      ;or get out if done
1911  00             IDL         ;
1912  3B 07          BNF MUL1    ;If DF was 0, nothing to do
1914  9A             GHI A       ;If 1, add RB.0 to partial
1915  52             STR 2       ;product sum in RA.1
1916  8B             GLO B       ;
1917  F4             ADD         ;
1918  BA             PHI A       ;
1919  30 07          BR MUL1     ;Go back for more
191B  00             IDL         ;
191C  00             IDL         ;
*************************************************************
Add.  Code     Label Mnemonic         Comments
191D  F8 11    DIV   LDI 11      ; 11 into R8.0  pass counter
191F  A8             PLO 8       ;               (17 passes)
```

```
1920  F8 00          LDI 00     ; 00 into RA.1  for subtract
1922  BA             PHI A      ; 00 into R8.1  for carry
1923  B8             PHI 8      ;               (only lsb)
1924  8A       DIV1  GLO A      ;Find RA.1 - RA.0
1925  52             STR 2      ;If minus, can't divide yet
1926  9A             GHI A      ;
1927  F7             SM         ;
1928  52             STR 2      ;Value into M(RX), sign in DF
1929  98             GHI 8      ;R8.1 into D just in case
192A  CF             LSDF       ;Ahead if can't divide,
192B  76             RSHR       ;else put R8.1 lsb into DF
192C  C7             LSNF       ;Skip if no borrow available
192D  F0             LDX        ;Get abcve subtraction back
192E  BA             PHI A      ;to replace value in RA.1
192F  8B             GLO B      ;Shift DF into lsb of RB to
1930  7E             RSHL       ;build up quotient
1931  AB             PLO B      ;Shift msb of RB into lsb of
1932  9B             GHI B      ;RA.1 for use as dividend
1933  7E             RSHL       ;
1934  BB             PHI B      ;
1935  9A             GHI A      ;
1936  7E             RSHL       ;
1937  BA             PHI A      ;Shift msb of RA.1 into lsb
1938  98             GHI 8      ;of R8.1 for use as carry
1939  7E             RSHL       ;
193A  B8             PHI 8      ;
193B  28             DEC 8      ;Decrement pass counter
193C  88             GLO 8      ;Check pass counter,
193D  3A 24          BNZ DIV1   ;back if needed
193F  8A             GLO A      ;Get divisor
1940  52             STR 2      ;
1941  9A             GHI A      ;Get remainder x 2
1942  F7             SM         ;Subtract, if + or 0 round up
1943  98             GHI 8      ;
1944  CF             LSDF       ;Skip if positive or zero,
1945  76             RSHR       ;else shift R8.1 lsb into DF
1946  C7             LSNF       ;and skip if no carry
1947  1B             INC B      ;Round up if required
1948  D5             RETURN     ;Quotient in RB and
1949  D5             RETURN     ;twice remainder in RA.1
*****************************************************************
Add.  Code     Label  Mnemonic          Comments
194A  9C       SQRT   GHI C      ;
194B  FF FE           SMI FE     ;Find sign of RC.1 - FE
194D  F8 FF           LDI FF     ;Put FF in RA.0 (needed any-
194F  AA              PLO A      ;way, could even be answer!)
1950  C7              LSNF       ;Skip if negative, else we've
1951  D5              RETURN     ;got answer, get out
1952  00              IDL        ;
1953  F8 08           LDI 08     ; 08 into R7.0  pass counter
1955  A7              PLO 7      ;
```

```
1956  9C       SQR1  GHI C      ;Put RC into RB
1957  BB             PHI B      ;
1958  8C             GLO C      ;
1959  AB             PLO B      ;
195A  D4 191D        CALL DIV   ;Put RB/RA.0 into RB
195D  8B             GLO B      ;Put (RB.0 + RA.0)/2 in RA.0
195E  52             STR 2      ;
195F  8A             GLO A      ;
1960  F4             ADD        ;
1961  76             RSHR       ;
1962  AA             PLO A      ;
1963  27             DEC 7      ;Decrement pass counter
1964  87             GLO 7      ;
1965  3A 56          BNZ SQR1   ;Go back if not done
1967  D5             RETURN     ;note: This is an iterative
1968  00             IDL        ;technique, there is no magic
1969  00             IDL        ;about 8 passes
*****************************************************************
Add.  Code     Label Mnemonic          Comments
196A  F8 00    ABSQ1 LDI 00     ; 00 into RF  clear RF if
196C  BF             PHI F      ; entered here
196D  AF             PLO F      ;
196E  8A       ABSQ2 GLO A      ;Find RB.0 - RA.0
196F  52             STR 2      ;
1970  8B             GLO B      ;
1971  F7             SM         ;
1972  CF             LSDF       ;Skip if positive or zero,
1973  8B             GLO B      ;else find RA.0 - RB.0 which
1974  F5             SD         ;will then be positive
1975  AA             PLO A      ;Answer into RA.0 & RB.0
1976  AB             PLO B      ;
1977  D4 1900 MABAF  CALL MULT  ;Put [RA.0 - RB.0]^2 into RA
197A  8A       ADDAF GLO A      ;Put RA + RF into RF
197B  52             STR 2      ;
197C  8F             GLO F      ;
197D  F4             ADD        ;
197E  AF             PLO F      ;
197F  9A             GHI A      ;
1980  52             STR 2      ;
1981  9F             GHI F      ;
1982  74             ADC        ;
1983  BF             PHI F      ;
1984  D5             RETURN     ;Answer in RF, carry in DF
1985  00             IDL        ;
1986  00             IDL        ;
*****************************************************************
Add.  Code     Label Mnemonic          Comments
1A5B  D4 196E  RSQ   CALL ABSQ2;Put [RA.0 - RB.0]^2 into RA
1A5E  C7             LSNF       ;Skip if no carry,
1A5F  7B             SEQ        ;else set Q to show it
1A60  C4             NOP        ;
```

```
1A61  D4 18F4        CALL RAS   ;Get and resave RA, RB & RC
1A64  D5             RETURN     ;
1A65  00             IDL        ;note: This is really a slave
1A66  00             IDL        ;routine for RSQD
*****************************************************************
Add.  Code     Label Mnemonic          Comments
1A00  D4 18F4 RSQD   CALL RAS   ;Get and resave RA, RB & RC
1A03  7A             REQ        ;Reset Q from wherever
1A04  9A             GHI A      ;
1A05  AA             PLO A      ;
1A06  9C             GHI C      ;
1A07  AB             PLO B      ;
1A08  D4 196A        CALL ABSQ1 ;Put [X1 - XC]^2 into RF
1A0B  9C             GHI C      ;
1A0C  AA             PLO A      ;
1A0D  9B             GHI B      ;
1A0E  AB             PLO B      ;
1A0F  D4 1A5B        CALL RSQ   ;Put [XC - X2]^2 + RF into RF
1A12  8C             GLO C      ;
1A13  AB             PLO B      ;
1A14  D4 1A5B        CALL RSQ   ;Put [Y1 - YC]^2 + RF into RF
1A17  8C             GLO C      ;
1A18  AA             PLO A      ;
1A19  D4 1A5B        CALL RSQ   ;Put [YC - Y2]^2 + RF into RF
1A1C  8F             GLO F      ;Put RF + 0001 into RF to get
1A1D  FC 01          ADI 01     ;rounding when shifting
1A1F  AF             PLO F      ;
1A20  9F             GHI F      ;
1A21  7C 00          ADCI       ;
1A23  C7             LSNF       ;Skip if no overflow,
1A24  7B             SEQ        ;else set Q to show it
1A25  C4             NOP        ;
1A26  F6             SHR        ;Shift high byte to divide
1A27  C5             LSNQ       ;Skip if there was no carry
1A28  FC 80          ADI 80     ;Else add 80 to set msb high
1A2A  BC             PHI C      ;Put it in RC.1 for SQRT
1A2B  50             STR 0      ;Store it in "HR^2" location
1A2C  20             DEC 0      ;Point to "LR^2" location
1A2D  8F             GLO F      ;Shift low byte to divide it,
1A2E  76             SHRC       ;DF still has info in it
1A2F  AC             PLO C      ;Put it in RC.0 for SQRT
1A30  50             STR 0      ;Store it in "LR^2" location
1A31  10             INC 0      ;Point to "HR^2"
1A32  D4 194A        CALL SQRT  ;Put SQRT(RC) into RA.0
1A35  8A             GLO A      ;Hide answer in RF.0,
1A36  AF             PLO F      ;uncorrected radius "R"
;note: The rest of this program checks for the possibility
;that the computed radius is less than the X distance from
;X1 or X2 to XC. This only arises due to rounding errors
;in SQRT and is corrected by incrementing R. Since the Y
;values are computed they need not be checked in this way.
```

```
1A37  D4 18F4        CALL RAS   ;Restore and save RA, RB & RC
1A3A  7A             REQ        ;Could be set from carry
1A3B  2F       RSQ3  DEC F      ;Correct for initial pass
1A3C  1F       RSQ1  INC F      ;Increase "R" value
1A3D  9A             GHI A      ;Get X1
1A3E  C5             LSNQ       ;But if Q set,
1A3F  9B             GHI B      ;get X2
1A40  C4             NOP        ;
1A41  52             STR 2      ;
1A42  9C             GHI C      ;Get XC
1A43  F7             SM         ;Find XC - X1 or X2
1A44  CF             LSDF       ;Skip if positive,
1A45  9C             GHI C      ;
1A46  F5             SD         ;else find X1 or X2 - XC
1A47  52             STR 2      ;Put (positive) result away
1A48  8F             GLO F      ;Check if R >= [X? - XC]
1A49  F7             SM         ;
1A4A  3B 3C          BM RSQ1    ;If not, inc RF, try again
1A4C  31 51          BQ RSQ2    ;If Q set, go ahead,
1A4E  7B             SEQ        ;if not, set Q for other X's,
1A4F  30 3B          BR RSQ3    ;go back and check
1A51  8F       RSQ2  GLO F      ;Get fully checked R value
1A52  20             DEC 0      ;
1A53  20             DEC 0      ;
1A54  50             STR 0      ;Store it in "R" location
1A55  10             INC 0      ;
1A56  10             INC 0      ;Point to "HR^2"
1A57  7A             REQ        ;Clear Q
1A58  D5             RETURN     ;and get out
1A59  00             IDL        ;
1A5A  00             IDL        ;
*****************************************************************
Add.  Code     Label  Mnemonic          Comments
19CB  20       YWOC   DEC 0       ;Enter: [X - XC]^2 in RA & RF
19CC  40              LDA 0       ;Put R^2 - [X - XC]^2 in RC
19CD  52              STR 2       ;
19CE  8A              GLO A       ;
19CF  F5              SD          ;
19D0  AC              PLO C       ;
19D1  40              LDA 0       ;
19D2  20              DEC 0       ;
19D3  52              STR 2       ;
19D4  9A              GHI A       ;
19D5  75              SDB         ;
19D6  33 DE           BPZ YWO1    ;Ahead if positive, O.K.
19D8  F8 00           LDI 00      ;If not,
19DA  AC              PLO C       ; 0000 into RC  squared value
19DB  BC              PHI C       ; 00 into RA.0  final answer
19DC  AA              PLO A       ;
19DD  D5              RETURN      ;
19DE  BC       YWO1   PHI C       ;If positive,
```

```
19DF  D4 194A       CALL SQRT ;put root into RA.0
19E2  D5            RETURN    ;and return
19E3  00            IDL       ;
************************************************************
```

GRAPHICS PROGRAMS

```
Add.  Code     Label  Mnemonic        Comments
1856  EF       ZSCR   SEX F      ;Make RF be RX
1857  F8 37             LDI 37    ; 37FF into RF  end of screen
1859  BF                PHI F     ;
185A  F8 FF             LDI FF    ;
185C  AF                PLO F     ;
185D  F8 00    ZSC1     LDI 00    ;Put 00 into M(RF) to blank
185F  73                STXD      ;screen, decrement RF
1860  9F                GHI F     ;Check if RF.1 is 1F, so RF
1861  FF 1F             SMI 1F    ;would be 1FFF, meaning done
1863  3A 5D             BNZ ZSC1  ;Back if not done,
1865  E2                SEX 2     ;else make R2 be RX again
1866  D5                RETURN    ;and get out
1867  00                IDL       ;
1868  00                IDL       ;
************************************************************
Add.  Code     Label  Mnemonic        Comments
1869  EF       CSCR   SEX F      ;Make RF be RX
186A  F8 37             LDI 37    ; 37FF into RF  end of screen
186C  BF                PHI F     ;
186D  F8 FF             LDI FF    ;
186F  AF                PLO F     ;
1870  F8 FF    CSC1     LDI FF    ;Exclusive or FF with each
1872  F3                XOR       ;byte to complement screen
1873  73                STXD      ;
1874  9F                GHI F     ;
1875  FF 1F             SMI 1F    ;
1877  3A 70             BNZ CSC1  ;Back if RF not 1FFF
1879  E2                SEX 2     ;Make R2 be RX
187A  D5                RETURN    ;
187B  00                IDL       ;
187C  00                IDL       ;
************************************************************
Add.  Code     Label  Mnemonic        Comments
187D  F8 00    S256     LDI 00    ;Set high 4 bits to 0000 to
187F  AF                PLO F     ;show 256 graphics wanted
1880  8F       S251     GLO F     ;Set low 4 bits to 0000 to
1881  52                STR 2     ;format top of screen
1882  63                OUT 3     ;Output RF.0 to TV address
1883  22                DEC 2     ;board as a format word
1884  1F                INC F     ;Point to next screen segment
1885  8F                GLO F     ;
1886  FA 10             ANI 10    ;And 10 to check if 16 passes
```

```
1888  32 60          BZ S251    ;Go back if not
188A  F8 80          LDI 80     ;
188C  52             STR 2      ;
188D  63             OUT 2      ;Output 80 to TV address
188E  22             DEC 2      ;board to start display at
188F  D5             RETURN     ;M(2000)
1890  00             IDL        ;
1891  00             IDL        ;
*****************************************************************
Add.  Code     Label  Mnemonic          Comments
1800  8E       AM256  GLO E       ;Get Y value
1801  FF BF           SMI BF      ;Subtract BF, invert Y coord.
1803  C7               LSNF       ;Skip if neg, on screen, O.K.
1804  F8 00           LDI 00      ;Else set to 00,top of screen
1806  FB FF           XRI FF      ;Complement, add 01 so Y=00
1808  FC 01           ADI 01      ;will appear at bottom, Y>=BF
180A  BF               PHI F      ;appears at top
180B  8D               GLO D      ;Get X value
180C  F6               SHR        ;Shift out lower 3 bits which
180D  F6               SHR        ;contain bit address, 1 of 8
180E  F6               SHR        ;columns
180F  73               STXD       ;Store byte value (000XXXXX)
1810  60               IRX        ;
1811  9F               GHI F      ;Get corrected Y value
1812  FE               SHL        ;Shift out byte values,
1813  FE               SHL        ;leave bit values correspon-
1814  FE               SHL        ;ding to 1 of 8 rows
1815  FE               SHL        ;(YYY00000)
1816  FE               SHL        ;
1817  F1               OR         ;OR X and Y values, put this
1818  AC               PLO C      ;low address into RC.0
1819  9F               GHI F      ;Get corrected Y value again,
181A  F6               SHR        ;shift out bit values, leave
181B  F6               SHR        ;bytes (000YYYYY)
181C  F6               SHR        ;
181D  FC 20           ADI 20      ;Add screen address offset,
181F  BC               PHI C      ;put high address into RC.1
1820  F8 01           LDI 01      ;Put 01 into RF.1 as an
1822  BF               PHI F      ;initial bit mask value
1823  8D               GLO D      ;Get X value
1824  FA 07           ANI 07      ;Mask in bit value
1826  AF               PLO F      ;Save it as a counter
1827  C6       AM1     LSNZ       ;Skip ahead if counter not 00
1828  D5               RETURN     ;Else mask now finished, out
1829  00               IDL        ;
182A  9F               GHI F      ;Get mask,
182B  FE               SHL        ;shift it left,
182C  BF               PHI F      ;
182D  2F               DEC F      ;decrement counter,
182E  8F               GLO F      ;get ready to check it,
182F  30 27            BR AM1     ;and go back for more
```

```
1831  00            IDL       ;
1832  00            IDL       ;
1833  00            IDL       ;
*****************************************************************
Add.  Code    Label Mnemonic        Comments
1834  9F      GBIT  GHI F     ;Get bit mask
1835  EC            SEX C     ;Make RC be RX
1836  F2            AND       ;And mask and data, get bit
1837  E2            SEX 2     ;Make R2 be RX
1838  CE            LSZ       ;Skip 2 if bit 0
1839  7B            SEQ       ;Else bit 1, set Q
183A  D5            RETURN    ;and get out
183B  7A            REQ       ;Bit 0, reset Q
183C  D5            RETURN    ;and get out
183D  00            IDL       ;
183E  00            IDL       ;
183F  00            IDL       ;
*****************************************************************
Add.  Code    Label Mnemonic        Comments
1840  9F      SBIT  GHI F     ;Get mask
1841  EC            SEX C     ;Make RC be RX
1842  CD            LSQ       ;Skip if bit to be set
1843  FB FF         XRI FF    ;Else put compl. of mask in D
1845  CD            LSQ       ;Skip if bit to be set
1846  F2            AND       ;Compl. mask AND byte to zero
1847  38            SKP       ;bit, skip next byte
1848  F1            OR        ;Mask OR byte to set bit
1849  5C            STR C     ;Restore byte w/ changed bit
184A  E2            SEX 2     ;Make R2 be RX
184B  D5            RETURN    ;
184C  00            IDL       ;
184D  00            IDL       ;
*****************************************************************
Add.  Code    Label Mnemonic        Comments
184E  9F      CBIT  GHI F     ;Get mask
184F  EC            SEX C     ;Make RC be RX
1850  F3            XOR       ;Exclusive or mask and data
1851  5C            STR C     ;to complement bit, restore
1852  E2            SEX 2     ;Make R2 be RX
1853  D5            RETURN    ;
1854  00            IDL       ;
1855  00            IDL       ;
*****************************************************************
Add.  Code    Label Mnemonic        Comments
1892  F8 00   DLINE LDI 00     ;
1894  A7            PLO 7     ;Set sgn(dX) to 00 = positive
1895  A8            PLO 8     ;Set sgn(dY) to 00 = positive
1896  9A            GHI A     ;Put X1
1897  AD            PLO D     ;into X
1898  52            STR 2     ;and M(R2)
1899  9B            GHI B     ;Get X2
```

```
189A  F7          SM        ;Put X2 - X1 into D
189B  33 A2       BPZ DL1   ;Jump if + or 0, O.K.
189D  FF 01       SMI 01    ;Else find two's complement
189F  FB FF       XOR FF    ;(make it positive) and set
18A1  17          INC 7     ;sgn(dX) to 01 = negative
18A2  73    DL1   STXD      ;Push [dX] to stack (working
18A3  B7          PHI 7     ;value) and into R7.1
18A4  8A          GLO A     ;Put Y1
18A5  AE          PLO E     ;into Y
18A6  52          STR 2     ;and M(R2)
18A7  8B          GLO B     ;Get Y2
18A8  F7          SM        ;Put Y2 - Y1 into D
18A9  33 B0       BPZ DL2   ;Jump if + or 0, O.K.
18AB  FF 01       SMI 01    ;Else find two's complement
18AD  FB FF       XOR FF    ;(make it positive) and set
18AF  18          INC 8     ;sgn(dY) to 01 = negative
18B0  73    DL2   STXD      ;Push [dY] to stack (working
18B1  B8          PHI 8     ;value) and into R8.1
18B2  7B          SEQ       ;Set Q so SBIT will set bits
18B3  89    DL6   GLO 9     ;This checks R9 as a flag if
18B4  32 BB       BZ DL3    ;(unwritten) routines other
18B6  D4 1800     SUB AM256 ;than AM256 required - for
18B9  30 BE       BR DL4    ;now both choices are AM256
18BB  D4 1800 DL3 SUB AM256 ;Get mem. address & bit mask
18BE  D4 1840 DL4 SUB SBIT  ;Set required bit
18C1  7A          REQ       ;Reset Q 'cause it's needed
18C2  8D          GLO D     ;Get X
18C3  52          STR 2     ;
18C4  9B          GHI B     ;Get X2
18C5  F3          XOR       ;Exclusive or them, 00 if =
18C6  C6          LSNZ      ;Skip if not equal, not done
18C7  B7          PHI 7     ;Else set [dX] to 00,
18C8  7B          SEQ       ;and set Q to show X's done
18C9  8E          GLO E     ;Get Y
18CA  52          STR 2     ;
18CB  8B          GLO B     ;Get Y2
18CC  F3          XOR       ;Exclusive or them, 00 if =
18CD  C6          LSNZ      ;Skip if not equal, not done
18CE  B8          PHI 8     ;Else set [dY] to 00
18CF  CD          LSQ       ;Skip if Q set, X's also done
18D0  7A          REQ       ;Reset Q 'cause only X's done
18D1  C4          NOP       ;
18D2  60    DL5   IRX       ;(Q set if X's and Y's done)
18D3  60          IRX       ;Point to working X
18D4  C5          LSNQ      ;Skip if not done, Q = 0
18D5  7A          REQ       ;Else reset Q
18D6  D5          RETURN    ;and leave
18D7  97          GHI 7     ;Get [dX] from R7.1
18D8  F4          ADD       ;Put [dX] + working X into D
18D9  73          STXD      ;Restore, point to working Y
18DA  3B E2       BNF DL7   ;Jump ahead for Y if no carry
```

```
18DC  7B             SEQ       ;Else set Q to show change
18DD  1D             INC D     ;and increment X
18DE  87             GLO 7     ;Get sgn(dX),
18DF  CE             LSZ       ;skip if positive, O.K.
18E0  2D             DEC D     ;Else decrement X by 2,
18E1  2D             DEC D     ;effectively by 1
18E2  98       DL7   GHI 8     ;Get [dY] from R8.1
18E3  F4             ADD       ;Put [dY] + working Y into D
18E4  73             STXD      ;Restore, point to free loc.
18E5  3B ED          BNF DL8   ;Jump ahead if no carry
18E7  7B             SEQ       ;Else set Q to show change
18E8  1E             INC E     ;and increment Y
18E9  88             GLO 8     ;Get sgn(dY),
18EA  CE             LSZ       ;skip if positive, O.K.
18EB  2E             DEC E     ;Else decrement Y by 2,
18EC  2E             DEC E     ;effectively by 1
18ED  31 B3    DL8   BQ DL6    ;If change, draw new point,
18EF  30 D2          BR DL5    ;if not, go back until there
18F1  00             IDL       ;is a new point ready
18F2  00             IDL       ;
*****************************************************************
Add.  Code     Label Mnemonic          Comments
1A88  D4 18F4 DRAW   SUB RAS   ;Get and resave P1, P2 & PC
1A8B  99             GHI 9     ;Get X from R9.1
1A8C  AB             PLO B     ;Put it into RB.0
1A8D  9C             GHI C     ;
1A8E  AA             PLO A     ;Put XC into RA.0
1A8F  D4 196E        SUB ABSQ2 ;Put [XC - X]^2 into RA
1A92  D4 19CB        SUB YWOC  ;Find Y value, centre (00,00)
1A95  8A             GLO A     ;Hide it in RF.0
1A96  AF             PLO F     ;
1A97  D4 18F4        SUB RAS   ;Get and resave P1, P2 & PC
1A9A  8F             GLO F     ;Put Y into M(R2)
1A9B  52             STR 2     ;
1A9C  46             LDA 6     ;Load immediate byte, 00 = up
1A9D  3A A6          BNZ DR1   ;If not 00, draw below centre
1A9F  8C             GLO C     ;Get YC,
1AA0  F4             ADD       ;add it to Y value, = final Y
1AA1  C7             LSNF      ;Skip if no carry,
1AA2  F8 FF          LDI FF    ;else set maximum value
1AA4  30 AB          BR DR2    ;Go ahead
1AA6  8C       DR1   GLO C     ;Get YC,
1AA7  F7             SM        ;find YC - Y, = final Y
1AA8  CF             LSDF      ;Skip if no borrow,
1AA9  F8 00          LDI 00    ;else set minimum value
1AAB  AF       DR2   PLO F     ;Hide final Y value
1AAC  D4 1A67        SUB R0-9R ;Get last line coordinates
1AAF  8F             GLO F     ;Put final Y
1AB0  AB             PLO B     ;into RB.0
1AB1  99             GHI 9     ;Put current X
1AB2  BB             PHI B     ;into RB.1
```

```
1AB3  D4 1A77       SUB SR0+9 ;Save coords. while drawing
1AB6  D4 1892       SUB DLINE ;Draw from last point to new
1AB9  7B            SEQ       ;Set Q to show circle started
1ABA  D4 1A67       SUB R0-9R ;Get coordinates just drawn
1ABD  8B            GLO B     ;Make point just calculated
1ABE  AA            PLO A     ;into last point, RB into RA
1ABF  9B            GHI B     ;
1AC0  BA            PHI A     ;
1AC1  D4 1A77       SUB SR0+9 ;Save those coordinates
1AC4  D5            RETURN    ;
1AC5  00            IDL       ;1AC5 to 1AFF blank
1AFF  00            IDL       ;
*****************************************************************
Add.  Code     Label Mnemonic          Comments
1500  D4 1BA3 CIRCL CALL SABC ;Save P1, P2 & PC
1503  D4 1A77       CALL SR0+9;Save them higher up too
1506  D4 1A00       CALL RSQD ;Find R^2 and R
1509  D4 18F4       CALL RAS  ;Get P1, P2 & P3
150C  7A            REQ       ;Reset Q just in case
150D  20            DEC 0     ;
150E  20            DEC 0     ;
150F  40            LDA 0     ;Get R
1510  10            INC 0     ;
1511  52            STR 2     ;Put it in M(RX),
1512  AF            PLO F     ;and hide it RF.0
1513  9C            GHI C     ;Get XC
1514  F4            ADD       ;Find XC + R = right limit
1515  C7            LSNF      ;Skip if no carry, O.K.
1516  F8 FF         LDI FF    ;Else replace with max. value
1518  73            STXD      ;Push XC + R onto R2 stack
1519  C4            NOP       ;
151A  8F            GLO F     ;Get R again
151B  52            STR 2     ;
151C  9C            GHI C     ;Get XC again
151D  F7            SM        ;Find XC - R = left limit
151E  CF            LSDF      ;Skip if no borrow, O.K.
151F  F8 00         LDI 00    ;Else replace with min. value
1521  73            STXD      ;Push XC - R onto R2 stack
1522  9A            GHI A     ;Put X1, starting X, into
1523  B9            PHI 9     ;R9.1, the X value register
1524  8C            GLO C     ;Get Y1
1525  52            STR 2     ;
1526  8A            GLO A     ;Get Y1
1527  F7            SM        ;Find Y1 - YC, if negative,
1528  3B 4F         BM CR1    ;jump to start lower routine
152A  D4 1A88 CR3   CALL DRAW ;Find and draw this part. 00
152D  00            DATA      ;is data byte for upper rout.
152E  D4 18F4       CALL RAS  ;Get P1, P2 & PC
1531  99            GHI 9     ;Get X
1532  FF 01         SMI 01    ;Find X - 01, next point
1534  CF            LSDF      ;Skip if no borrow, O.K.
```

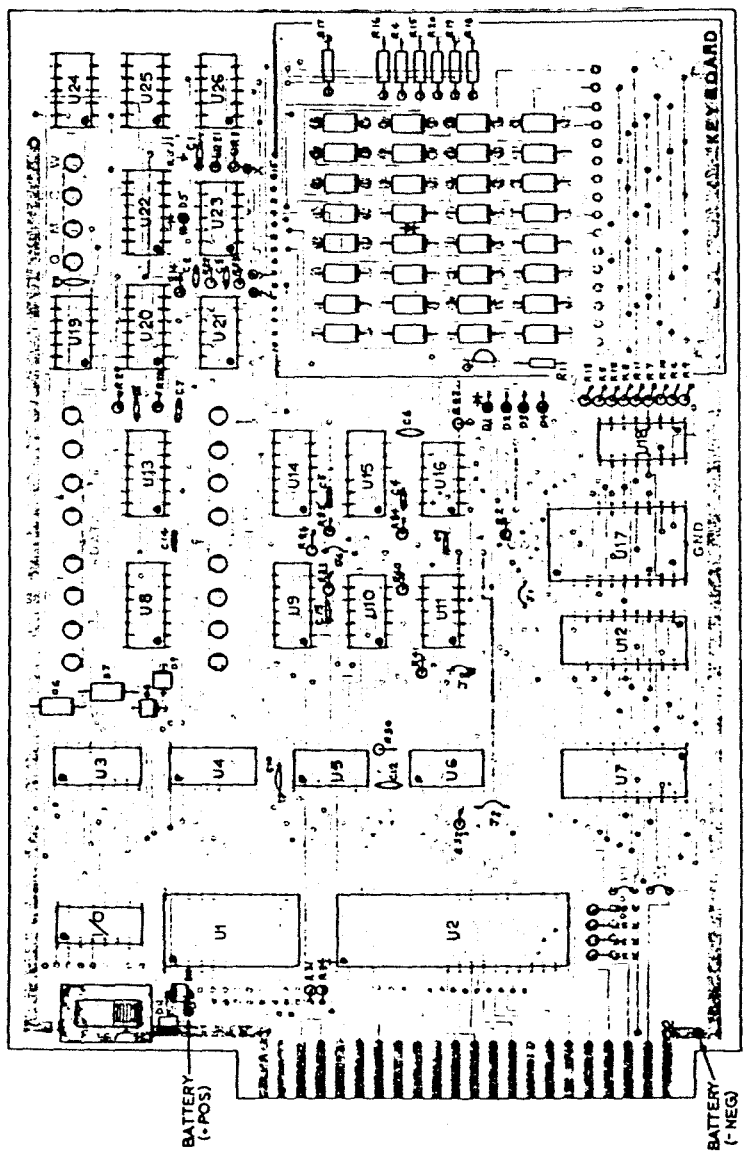
```
1535  F8 00          LDI 00     ;Else replace with min. value
1537  B9             PHI 9      ;Save as next X value
1538  3B 4F          BM CR1     ;If neg. go to lower routine
153A  52             STR 2      ;
153B  9B             GHI B      ;Get X2
153C  F7             SM         ;Find X2 - X, check for end
153D  3A 45          BNZ CR2    ;Ahead if not = , don't stop
153F  8C             GLO C      ;Now check if correct Y
1540  52             STR 2      ;Get YC
1541  8B             GLO B      ;Get Y2
1542  F7             SM         ;Find Y2 - YC, jump to stop
1543  33 80          BPZ STOPC  ;routine if + or 0
1545  99       CR2   GHI 9      ;If not, get X
1546  12             INC 2      ;Point R2 to XC - R
1547  F7             SM         ;Find X - (XC - R)
1548  22             DEC 2      ;
1549  33 2A          BPZ CR3    ;Back for more if O.K.
154B  99             GHI 9      ;Else add 01 to X to fix it,
154C  FC 01          ADI 01     ;
154E  B9             PHI 9      ;and continue in lower rout.
154F  D4 1A88  CR1   CALL DRAW  ;Find and draw this part. 01
1552  01             DATA       ;is data byte for lower rout.
1553  D4 18F4        CALL RAS   ;Get P1, P2 & PC
1556  99             GHI 9      ;Get X
1557  FC 0i          ADI 01     ;Find X + 01, next point
1559  C7             LSNF       ;Skip if no carry, O.K.
155A  F8 FF          LDI FF     ;Else replace with max. value
155C  B9             PHI 9      ;Save as next X value
155D  33 2A          BDF CR3    ;If carry, go to upper rout.
155F  52             STR 2      ;
1560  9B             GHI B      ;Get X2
1561  F7             SM         ;Find X2 - X, check for end
1562  3A 6A          BNZ CR4    ;Ahead if not = , don't stop
1564  8C             GLO C      ;Now check if correct Y
1565  52             STR 2      ;Get YC
1566  8B             GLO B      ;Get Y2
1567  F5             SD         ;Find YC - Y2, jump to stop
1568  33 80          BPZ STOPC  ;if positive or zero
156A  99       CR4   GHI 9      ;If not get X
156B  12             INC 2      ;
156C  12             INC 2      ;Point R2 to XC + R
156D  F5             SD         ;Find (XC + R) - X
156E  22             DEC 2      ;
156F  22             DEC 2      ;
1570  33 4F          BPZ CR1    ;Back for more if O.K.
1572  99             GHI 9      ;Else subtract 01 from X to
1573  FF 01          SMI 01     ;restore it,
1575  B9             PHI 9      ;
1576  30 2A          BR CR3     ;and continue in upper rout.
1578  00             IDL        ;1578 to 157F blank
157F  00             IDL        ;
```

```
1580  D4 18F4 STOPC CALL RAS   ;Get P1, P2 & PC
1583  8B            GLO B      ;Put Y2 into RF.0
1584  AF            PLO F      ;as current Y
1585  D4 1AAC       CALL DR3   ;Get into DRAW halfway
1588  D4 1BB2       CALL RABC  ;Clear out R0 stack
158B  12            INC 2      ;Clear out R2 stack
158C  12            INC 2      ;
158D  7A            REQ        ;Clear out Q
158E  D5            RETURN     ;Clear out!
158F  00            IDL        ;
*************************************************************
Add.  Code    Label Mnemonic         Comments
1480  80      PICT  GLO 0      ;Hide R0 in RF
1481  AF            PLO F      ;
1482  90            GHI 0      ;
1483  BF            PHI F      ;
1484  81            GLO 1      ;Put R1 (picture stack) in R0
1485  A0            PLO 0      ;
1486  91            GHI 1      ;
1487  B0            PHI 0      ;
1488  D4 1BB2       CALL RABC  ;Use RABC to fill RA, RB and
148B  80            GLO 0      ;RC with whatever
148C  A1            PLO 1      ;Put R0 back in R1 to save
148D  90            PHI 0      ;picture stack
148E  B1            PHI 1      ;
148F  8F            GLO F      ;Put RF in R0 to restore
1490  A0            PLO 0      ;normal data stack
1491  9F            GHI F      ;
1492  B0            PHI 0      ;
1493  11            INC 1      ;Advance picture stack and
1494  01            LDN 1      ;get picture instruction add.
1495  A3            PLO 3      ;in this page, jam it into
1496  00            IDL        ;program counter to go and do
1497  00            IDL        ;whatever with whatever
*************************************************************
Add.  Code    Label Mnemonic         Comments
1400  D4 1892 PILI  CALL DLINE;Draw a line from RA to RB,
1403  30 80         BR PICT    ;ignore RC
1405  00            IDL        ;
1406  00            IDL        ;
1407  00            IDL        ;
1408  D4 1500 PICI  CALL CIRCL;Draw a counterclockwise
140B  30 80         BR PICT    ;circle:
140D  00            IDL        ;RA = P1, start of arc
140E  00            IDL        ;RB = P2, end of arc
140F  00            IDL        ;RC = PC, centre
1410  D4 1856 PIZE  CALL ZSCR  ;Clear screen memory
1413  30 80         BR PICT    ;Ignore RA, RB, RC
1415  00            IDL        ;
1416  00            IDL        ;
1417  00            IDL        ;
```

```
1418  D4 1869        CALL CSCR ;Complement whole screen
141B  30 80          BR PICT   ;Ignore RA, RB, RC
141D  00             IDL       ;
141E  00             IDL       ;
141F  00             IDL       ;
1420  8A       PIDE  GLO A     ;Delay for a time propor-
1421  C6             LSNZ      ;tional to RA, ignore RB, RC
1422  9A             GHI A     ;
1423  E2             SEX 2     ;
1424  32 80          BZ PICT   ;
1426  2A             DEC A     ;
1427  30 20          BR PIDE   ;Loop 'till RA = 0000
1429  00             IDL       ;1429 to 142E blank
142E  00             IDL       ;
142F  D5       PIRE  RETURN    ;Picture finished, return to
                               ;calling program, ignore R's
1430  91       PICS  GHI 1     ;Push R1 picture stack onto
1431  73             STXD      ;R2 stack
1432  81             GLO 1     ;Load R1 with RA as a
1433  73             STXD      ;"picture subroutine", ignore
1434  8A             GLO A     ;RB, RC
1435  A1             PLO 1     ;
1436  9A             GHI A     ;
1437  B1             PHI 1     ;
1438  30 80          BR PICT   ;
143A  60       PIRS  IRX       ;Pull old R1 value from R2
143B  72             LDXA      ;stack, return from "picture
143C  A1             PLO 1     ;subroutine"
143D  F0             LDX       ;
143E  B1             PHI 1     ;
143F  30 80          BR PICT   ;
1441  00             IDL       ;1441 to 1447 blank
1447  00             IDL       ;
1448  8A       PIFO  GLO A     ;Output RA.0 to video board
1449  52             STR 2     ;as a screen format inst.
144A  63             OUT 3     ;Ignore RA.1, RB, RC
144B  22             DEC 2     ;
144C  30 80          BR PICT   ;
144E  00             IDL       ;
144F  00             IDL       ;
1450  D4 1B75  PIMO  CALL MOVE ;Move data in RA to RB inc-
1453  30 80          BR PICT   ;lusive to new home at RC
1455  00             IDL       ;
1456  00             IDL       ;
1457  00             IDL       ;
*************************************************************
```

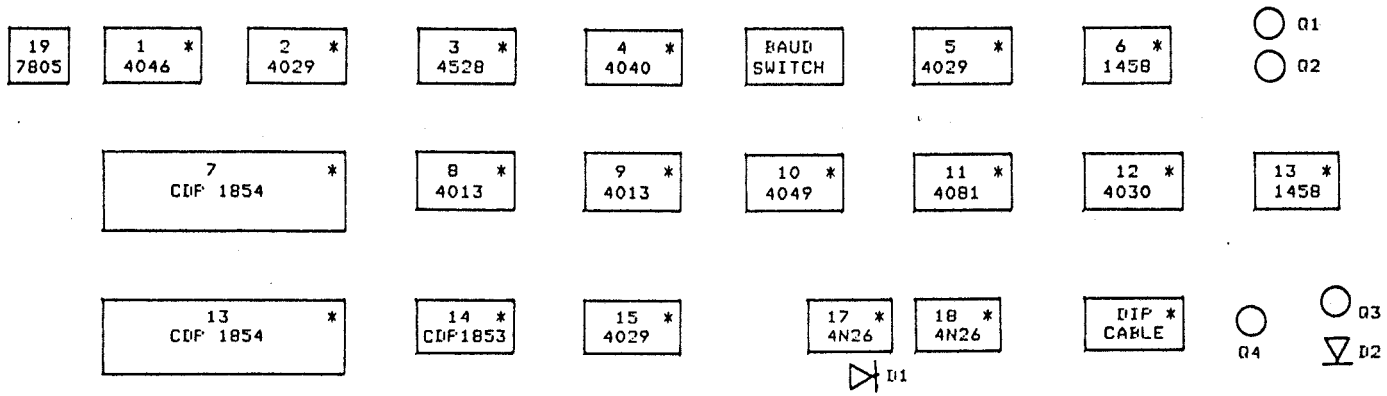
APPENDIX D

BOARD LAYOUTS

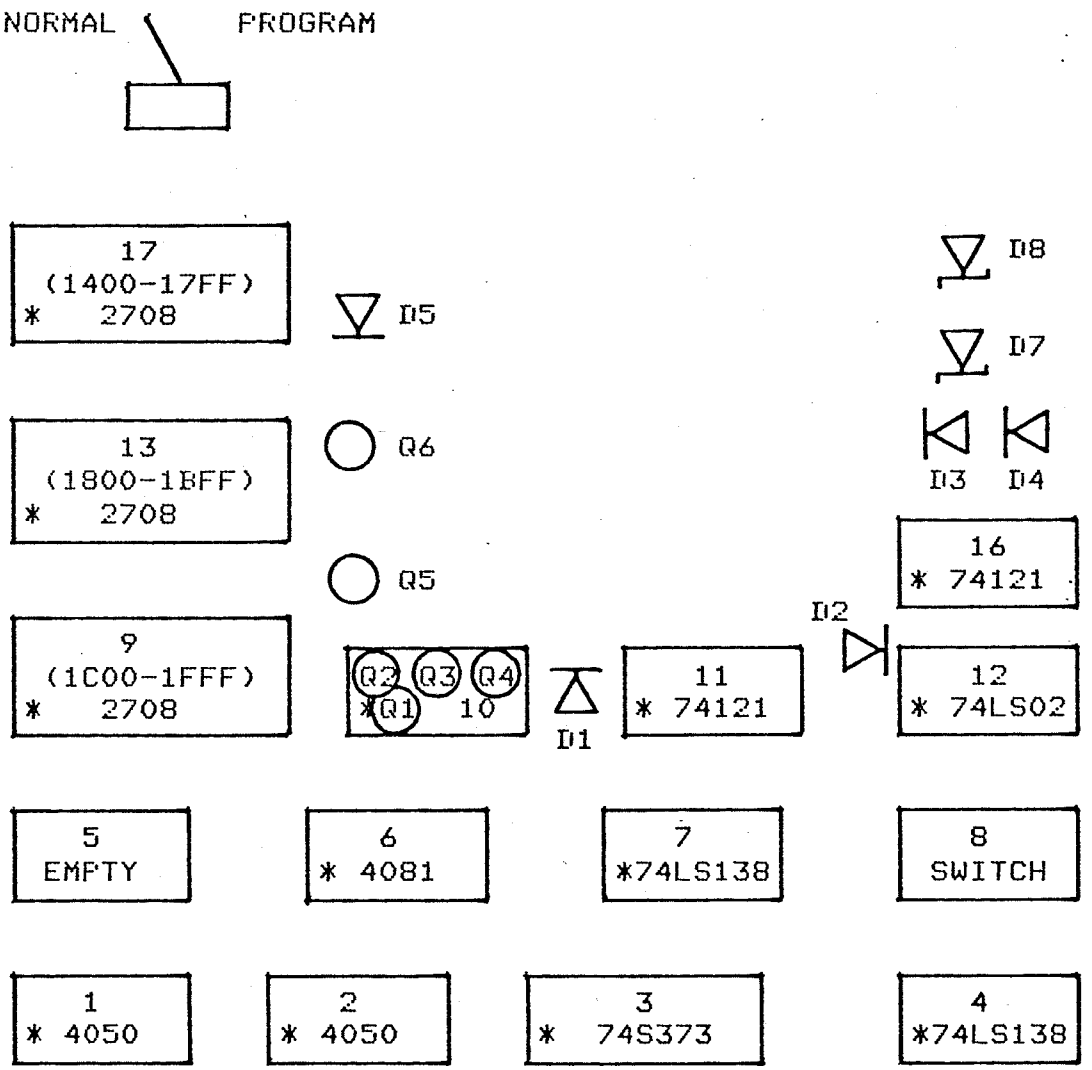


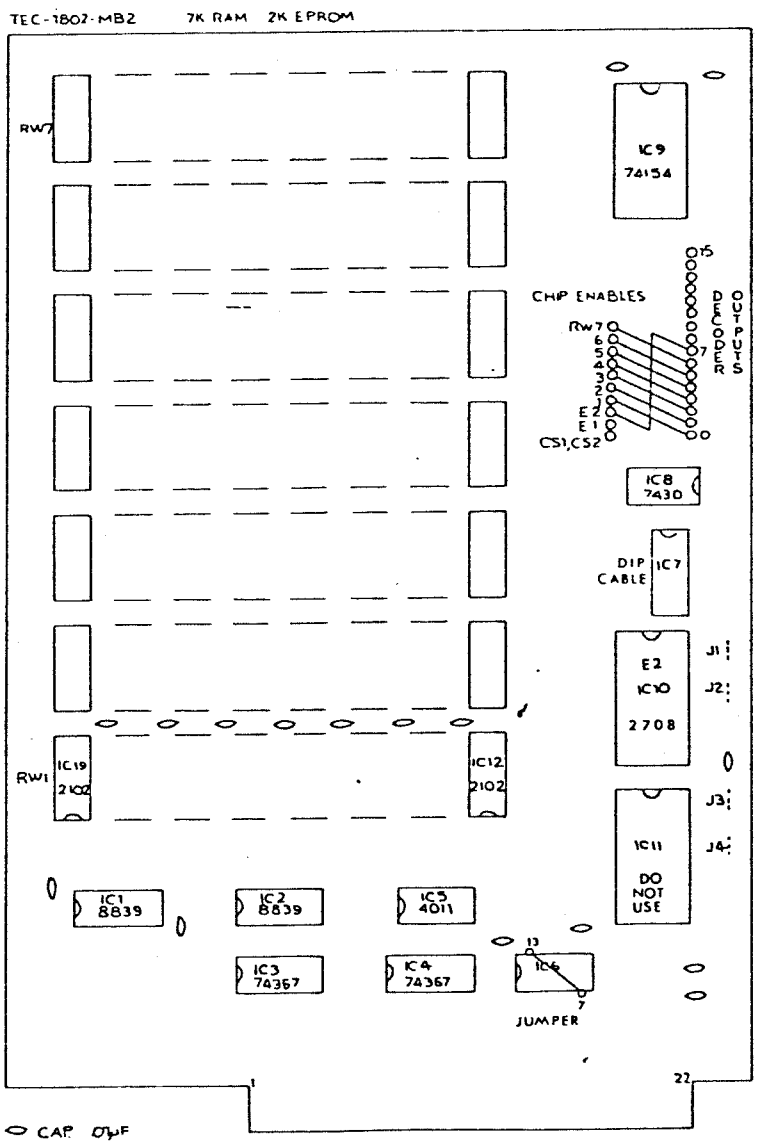
CPU Board Layout

SERIAL I/O BOARD LAYOUT



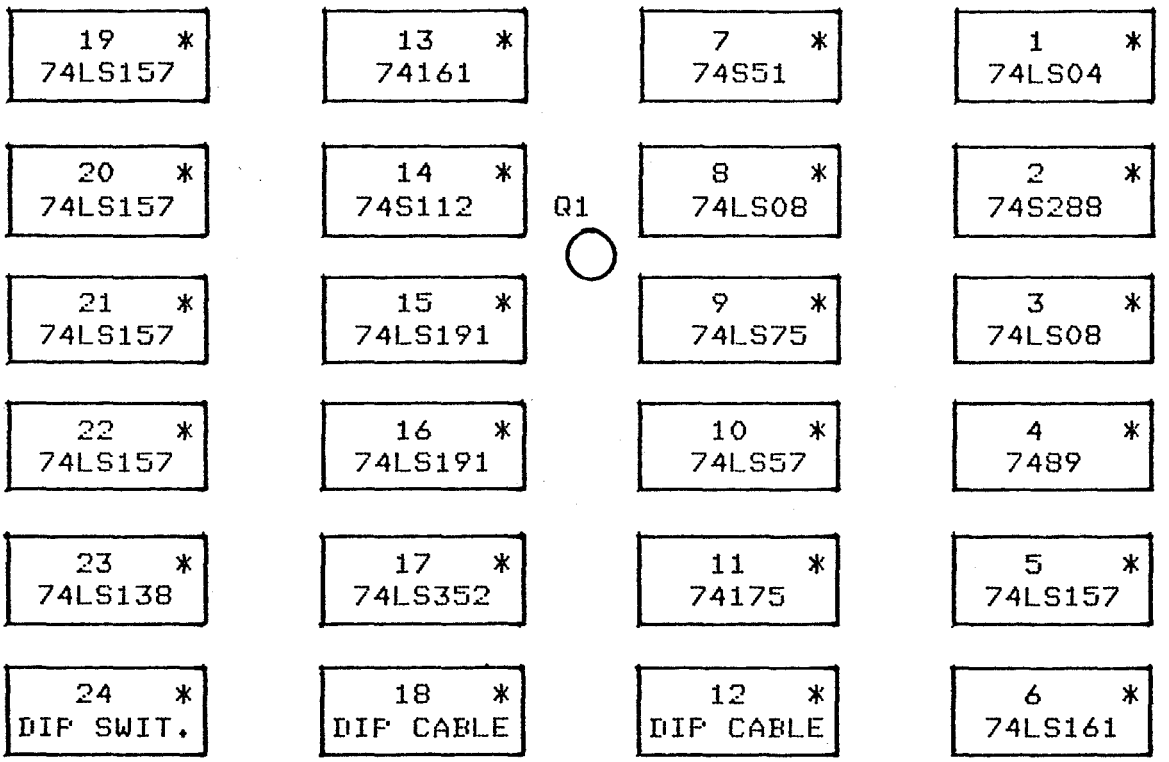
ADDRESS BUFFER / PROM BOARD LAYOUT



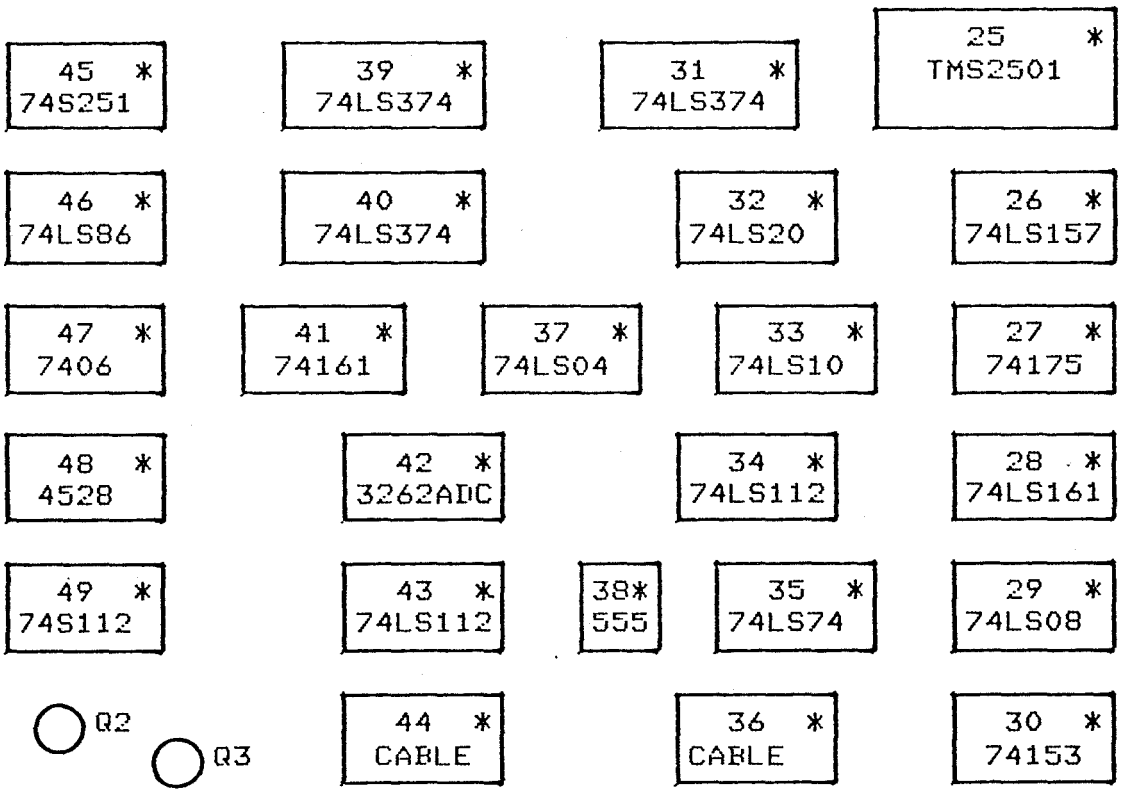


RAM Board Layout

TV ADDRESS BOARD LAYOUT



TV DATA BOARD LAYOUT



BUS CONNECTIONS

CPU BOARD

+5	>	23	1	<	+5
SR	<	24	2	<	CS1, CS2
CLK	>	25	3	<>	WAIT
DMAI	>	26	4	>	CLEAR
DMAO	>	27	5	>	Q
INT	>	28	6	>	SC1
MWR	<	29	7	>	SC0
TPA	<	30	8	>	MRD
TPB	<	31	9	<>	uD7
uA7	<	32	10	<>	uD6
uA6	<	33	11	<>	uD5
uA5	<	34	12	<>	uD4
uA4	<	35	13	<>	uD3
uA3	<	36	14	<>	uD2
uA2	<	37	15	<>	uD1
uA1	<	38	16	<>	uD0
uA0	<	39	17	<	EF3
N2	<	40	18	<	EF2
N1	<	41	19	<	EF1
N0	<	42	20	>	DN2
		43	21	<	CE1
GND	>	44	22	<	GND

DIP CABLE FOR PARALLEL I/O PORT

I/O	DI7	>	1	16	> DO7
	DI6	>	2	15	> DO6
	DI5	>	3	14	> DO3
	DI4	>	4	13	> DO4
	DI3	>	5	12	> DO0
	DI2	>	6	11	> DO2
	DI1	>	7	10	> DO1
	DI0	>	8	9	> DO5

BUS CONNECTIONS

SERIAL I/O BOARD

		A	1		
-5	>	B	2	<	-5
		C	3	<	+12
		D	4	<	CLEAR
		E	5		
		F	6		
		H	7		
TPA	>	J	8	<	MRD
TPB	>	K	9	<>	uD7
		L	10	<>	uD6
		M	11	<>	uD5
		N	12	<>	uD4
		P	13	<>	uD3
		R	14	<>	uD2
		S	15	<>	uD1
		T	16	<>	uD0
		U	17		
N2	>	V	18		
N1	>	W	19		
N0	>	X	20	>	DN2
DN3	<	Y	21		
GND	>	Z	22	<	GND

DIP CABLE TO REAR PANEL

CASSETTE IN	>	1	16	>	CASSETTE OUT
GND	<	2	15	>	GND
RS232 OUT	<	3	14	>	RTS
		4	13	>	20 ma OUT +
+ 20 ma DRIVE	<	5	12	>	20 ma OUT -
+/- 20 ma IN	>	6	11		
+/- 20 ma IN	>	7	10	<	RS232 IN
GND	<	8	9	>	GND

BUS CONNECTIONS

ADDRESS	BUFFER	/	FROM	BOARD
+5	> A		1 <	+5
uA0	> B		2 <	-5
uA1	> C		3 <	+12
uA2	> D		4 >	CE1
uA3	> E		5 >	WAIT
uA4	> F		6 <	MWR
uA5	> H		7 <	TPA
uA6	> J		8 <	MRD
uA7	> K		9 <>	uD7
uA8'	> L		10 <>	uD6
uA9'	< M		11 <>	uD5
uA10'	< N		12 <>	uD4
uA11'	< P		13 <>	uD3
uA12'	< R		14 <>	uD2
uA13'	< S		15 <>	uD1
uA14'	< T		16 <>	uD0
uA15'	< U		17	
MRD'	< V		18	
MWR'	< W		19	
	X		20	
	Y		21 <	12 VAC
GND	> Z		22 <	GND

BUS CONNECTIONS

7K RAM BOARD

+5	>	23	1	<	+5
-5	>	24	2	<	-5
TDO 7	<	25	3	<	+12
TDO 6	<	26	4	>	uD7'
TDO 5	<	27	5	>	uD6'
TDO 4	<	28	6	>	uD5'
TMWR	>	29	7	>	uD4'
		30	8	<	TMRD
		31	9	<>	uD7
TA7	>	32	10	<>	uD6
TA6	>	33	11	<>	uD5
TA5	>	34	12	<>	uD4
TA4	>	35	13	<>	uD3
TA3	>	36	14	<>	uD2
TA2	>	37	15	<>	uD1
TA1	>	38	16	<>	uD0
TA0	>	39	17		
TDO 3	<	40	18	>	uD3'
TDO 2	<	41	19	>	uD2'
TDO 1	<	42	20	>	uD1'
TDO 0	<	43	21	>	uD0'
GND	>	44	22	<	GND

DIP CABLE TO TV ADDRESS BOARD BUS SOCKET

IC 7			1	16		
	TA9'	>	2	15	<	TA8'
			3	14		
			4	13		
	TA10'	>	5	12	<	TA11'
			6	11		
	TA12'	>	7	10		
	GND	<	8	9		

BUS CONNECTIONS

TV ADDRESS BOARD

	+5	>	A	1	<	+5
	TA0	<	B	2	<	uA0
	TA1	<	C	3	<	uA1
	TA2	<	D	4	<	uA2
	TA3	<	E	5	<	uA3
	TA4	<	F	6	<	uA4
	TA5	<	H	7	<	uA5
	TA6	<	J	8	<	uA6
	TA7	<	K	9	<	uA7
	TA8	<	L	10	<	uA8'
via DIP	TA9	<	M	11	<	uA9'
cable to	TA10	<	N	12	<	uA10'
7K RAM	TA11	<	P	13	<	uA11'
	$\overline{\text{TA12}}$	<	R	14	<	uA12'
	$\overline{\text{TMRD}}$	<	S	15	<	uA13'
	$\overline{\text{TMWR}}$	<	T	16	<	uA14'
	$\overline{\text{MRD'}}$	>	U	17	<	uA15'
	$\overline{\text{MWR'}}$	>	V	18		
	TPB	>	W	19		
	OE	<	X	20		
	DN3	>	Y	21		
	GND	>	Z	22	<	GND

DIP CABLES TO TV DATA BOARD

IC 12	uD0'	>	1	16	<	16L
	uD1'	>	2	15	>	EOF
(to IC 36)	uD2'	>	3	14	<	FS
	uD3'	>	4	13	<	CCLK
	uD4'	>	5	12		
	uD5'	>	6	11	<	RCLK
	uD6	>	7	10	>	PD3
	uD7'	>	8	9	>	PD4
IC 18	BL	>	1	16	>	EOL
			2	15		
(to IC 44)			3	14		
	32CL	>	4	13		
	16CL	>	5	12		
	64CL	>	6	11		
	GND	<	7	10		
	DCL	<	8	9		

BUS CONNECTIONS

TV		DATA	BOARD		
+5	>	A	1	<	+5
		B	2	<	TDO 0
		C	3	<	TDO 1
		D	4	<	TDO 2
		E	5	<	TDO 3
		F	6	<	TDO 4
		H	7	<	TDO 5
REVERSE VIDEO	>	J	8	<	TDO 6
		K	9	<	TDO 7
		L	10	>	GND
		M	11	>	VIDEO
+5	<	N	12	>	SYNC
-12	>	P	13	>	GND
VERT. POSITION	>	R	14	<	uD0'
GND	<	S	15	<	uD1'
HORIZ. POSITION	>	T	16	<	uD2'
		U	17	<	uD3'
		V	18	<	uD4'
		W	19	<	uD5'
		X	20	<	uD6'
		Y	21	<	uD7'
GND	>	Z	22	<	GND

	DIP	CABLES	TO	TV	ADDRESS	BOARD
IC 36	uD0'	< 1	16	>	16L	
	uD1'	< 2	15	<	EOF	
	uD2'	< 3	14	>	FS	
	uD3'	< 4	13	<	CCLK	
	uD4'	< 5	12			
	uD5'	< 6	11	>	RCLK	
	uD6	< 7	10	<	PD3	
	uD7'	< 8	9	<	PD4	
IC 44	BL	< 1	16	<	EOL	
		2	15			
		3	14			
	32CL	< 4	13			
	16CL	< 5	12			
	64CL	< 6	11			
	GND	< 7	10			
	DCL	> 8	9			