Tiny Assembler 6800

Version 3.1

DESIGN AND IMPLEMENTATION OF A MICROPROCESSOR SELF ASSEMBLER

by Jack Emmerichs

BYTE Publications, Inc.
70 Main Street
Peterborough, New Hampshire 03458

Library of Congress Cataloging in Publication Data

Emmerichs, Jack.
Tiny assembler 6800, version 3.1.

1. Motorola 6800 (Computer)--Programming.
2. Assembling (Electronic computers). I. Title.
QA76.8.M67E45 001.6'425 78-7997
ISBN 0-931718-08-2

Printed in the United States of America

About This Book

Note that the first two sections of this book, "Designing the Tiny Assembler" and "Implementing the Tiny Assembler," are reprints of articles which first appeared in BYTE magazine's April and May 1977 issues. These articles refer to Tiny Assembler version 3.0, which is reproduced in listing form as Appendix A. In particular, the specific addresses mentioned in "Implementing the Tiny Assembler" refer to the listing of Appendix A. With the publication of this book, version 3.1 of Tiny Assembler is made available for the first time. The third section of the book is a reprint of "Expanding the Tiny Assembler" which appeared in the September 1977 issue of BYTE, and the fourth section of the book contains the "Tiny Assembler 6800 User's Guide" as updated to reflect version 3.1. The complete listing and bar code representation of version 3.1 are found in Appendices B and C respectively. Version 3.1 can be used "as is" by making patches to refer to the MIKBUG monitor IO routines of most Motorola 6800 systems, or equivalent patches to IO routines of other monitors in systems which have the first 4 K bytes of memory address space available for use by the assembler. A strategy similar to that described in "Implementing the Tiny Assembler" can be used to relocate the assembler to other regions of memory and provide simulations of IO devices for memory to memory assemblies.

Table of Contents

Cover Design: Dawson Advertising Agency, Inc.
Concord, New Hampshire

Artist: George Lallas

Designing the Tiny Assembler

Defining the Problem

When I first became aware of the small systems industry, I was particularly interested in finding out what software had been developed for personal computers, especially software that would run on a minimal system configuration without much optional hardware. One of the most useful software products for such small systems is an assembler to relieve the programmer of the tedious job of programming in machine language. I found no assemblers, however, that would run on a small machine configuration.

Most assemblers required at least 8 K bytes of memory (which was beyond my initial budget projections). As two pass assemblers, many of the existing products also required that source programs be entered twice so that all symbols in the source code could be resolved. This may be difficult or cumbersome if a system does not have IO devices that can easily handle high volumes of data. Furthermore, none of the assemblers that I found could be used interactively because of this two pass design. For a minimum system without offline file storage, this can be a major problem. In many cases several IO interfaces were required to handle the input source code, printed listing, and the generated object code (machine language). Finally, most of the assemblers then available provided all the bells and whistles available in much larger programs designed to run on large time-sharing systems. These large assemblers (called cross assemblers) are much more complex than I felt a small system assembler needs to be. I therefore decided to write a small but powerful assembler that would run on what to me was an affordable machine. This article is a description of the design and construction of such an assembler using structured programming techniques.

The first thing to do in any development project (though it is unfortunately often skipped in small projects) is to define the program's specifications. For this project the requirement was to develop a memory resident M6800 assembler which would:

- Assemble source code written in a free format subset of the M6800 assembly language as described in the Motorola publication *M6800 Microprocessor Programming Manual.*
- Operate completely within the first 4 K of memory including all tables, buffers, and other memory requirements.
- Completely assemble source programs in one pass to minimize IO operations.
- Require no more than one IO interface using the Motorola MIKBUG monitor.
- Support interactive operation.
- Facilitate modification and customization through the use of structured code.

Editor's Note:

At the present time (January 1977) Jack Emmerichs' assembler, described here and in the second part which follows next month, is my principal means of assembling programs for my homebrew 6800 based system. I've very successfully used it to assemble its own patches to fit my system, a text editor which is now part of my old hand assembled monitor, an extensive music text editor program, and two different versions (so far) of a multiprocessing music interpreter program to drive my synthesizer peripherals. After having suffered systems software withdrawal pains for over a year, it is great to get my "fix" of this intoxicating elixir. . . . **CH**

- Have sufficient capacity to be able to assemble itself so that no other software would be required to generate modified versions of the assembler program.

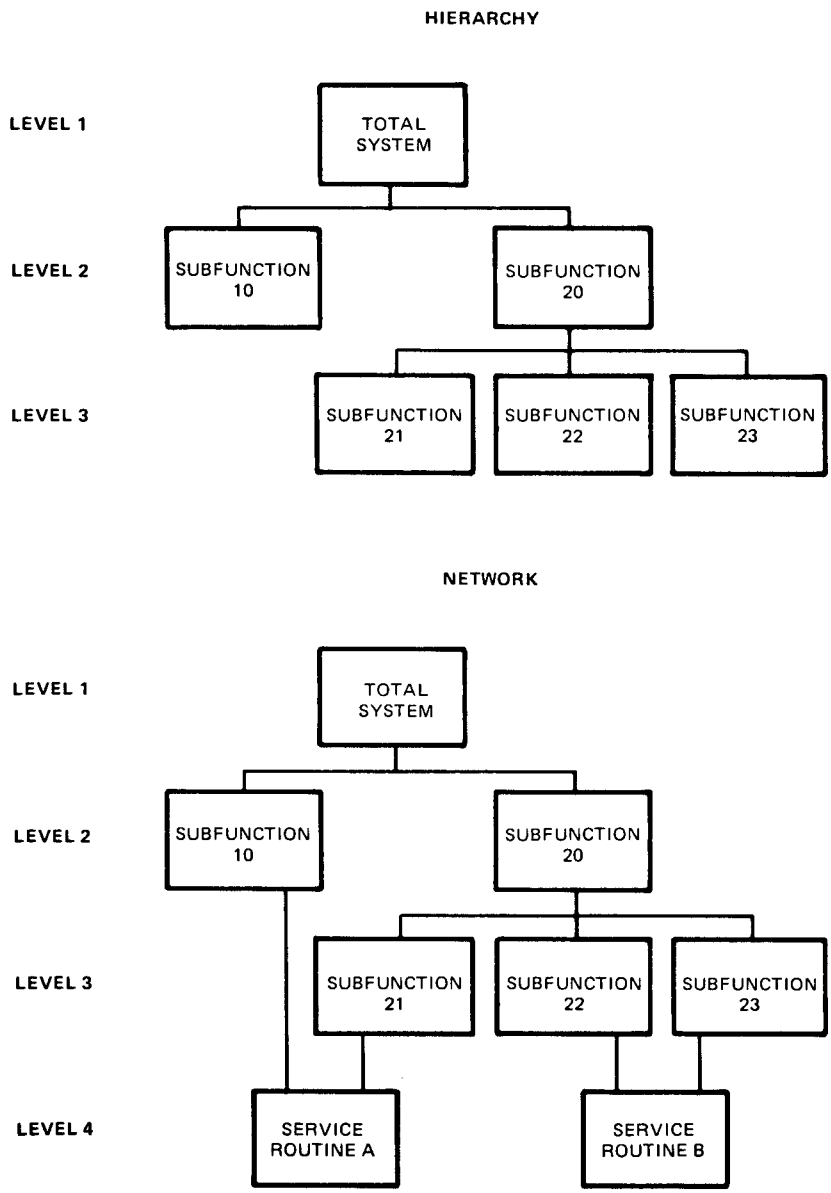


Figure 1: The top chart shows the structure of a single hierarchy diagram. The functions at each level are broken down into subfunctions at the next lower level. The relationships are shown as a simple tree structure. The bottom chart shows the structure of a network diagram. Here routines may be common to more than one high level function, and the structure of relationships can become much more complex. General subroutines that are common to many functions may be shown as a separate structure.

After the design criteria were thus set down, and before the program coding was started, it was necessary to define the functions that would be needed to meet the above specifications. Functions were defined from the highest level down to the lowest, and were related to each other through hierarchy and network diagrams (see figure 1). The first series of functions that we shall consider here are those that are common to all assemblers. These functions are discussed in more detail in the article "Jack and the Machine Talk" by Robert Grappel and Jack Hemenway, published in the August 1976 BYTE, page 52.

Figure 2 shows how the network diagram worked out for an initial conception of the assembler. At the top level the assembler is shown as a single function, level 1. Below this on level 2, the first major function is usually a table initialization and housekeeping routine (START) that need not be closely examined here. The second major function is to parse and process each line of incoming code with its labels, mnemonic instructions, operands, and comments (PARSE). The final major function is usually a termination and reporting function which again need not be closely examined here (CLEANUP).

At the next lower level, level 3, several functions must be defined which will be called from the parsing routine. The first of these is a function to convert English-like mnemonic instructions into machine language operation codes (opcodes). This is complicated in the M6800 by having the opcodes for some instructions vary with the accumulator or type of addressing used. This information is developed as each line of input is processed, so the parse and translate functions must work together to develop the correct opcode. The second function required at level 3 is to process assembler directives that control the assembly process from the input stream. These may include such operations as reserving sections of memory, setting up constants, assigning values to symbols, changing the program counter, and whatever else may be defined in the language. The third function required at this level is the maintenance of a table of user defined symbols with their associated values. Entries may be added to the table, or retrieved from it as the parsing routine directs. The final function at this level is an IO routine which receives source code from an input device, and writes listings and generated code to output devices.

The last level in this structure is shown here as a generalized error handling routine that flags and describes any ambiguities or errors in the source code. This can be

described as a general service routine, and may be invoked from any function detecting an error.

Several aspects of this project required the addition of new functions or modifications to existing functions within this structure, and will be treated here in some detail. The most severe functional changes were the result of requiring the source program to be processed completely in one pass.

The problem of making the assembler a one pass type in general is: What is to be done when the incoming code references a symbol that is not yet in the symbol table but may be defined later on? This is known as a forward reference and is quite likely to occur in almost all programs of nontrivial length.

The solution to this problem will require coordination between the assembler program and a loader program. The loader is run as a separate operation that converts the object code developed by the assembler into an executable program in memory. In this case, the assembler will generate one, two or three bytes of object code for each line of source code. It will also generate the address in memory where these bytes are to be loaded. When a forward reference is made, "dummy" code will be generated instead of the normal object code. This dummy code is simply an address or offset with a value of zero. The address of where the final address is to be loaded will be generated and saved in a table. When the reference is later resolved, the correct object code must be generated with an address which will cause it to overlay the original dummy code putting in the correct value. If several references have been made to a symbol by the time its value is resolved, several sets of addresses and patches to the dummy code must be developed. The opcode which is developed and put out with the dummy code will never be changed. The operand will be the dummy value to be patched. The loader required to properly handle this type of generated code will be shown in detail next month. *[Such a loader program is effectively a second "assembly" pass; however, by making the output routine include a loader and using extra memory as we've done with Jack's assembler, the entire process can be carried out within memory nearly instantly . . .* **CH***]*

Before considering the functions needed to achieve the generation of dummy code and later patches, two specific problems must be considered: What kind of opcodes and addressing formats must be handled, and how are expressions containing a forward reference handled?

First, consider the relationship between opcodes and addressing modes. For the

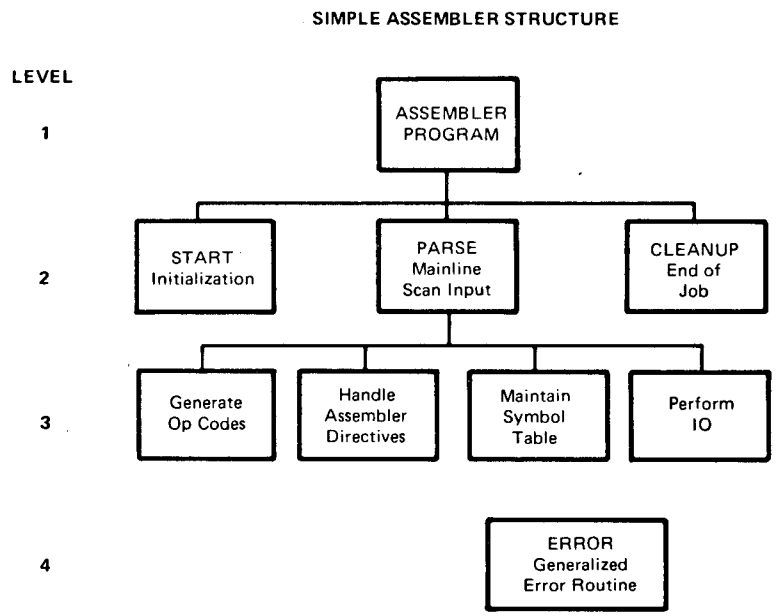


Figure 2: This is a simple network diagram showing the primary functions required by assembler programs in general. The error routine is common to all other functions.

M6800, opcodes are one byte in length, and may vary with the accumulator or mode of addressing being used. The operand may be a one or two byte address value which is a direct reference to a location in the first 256 positions of memory (zero through 255), an extended reference to any position in memory, an offset from the index register, or an offset relative to the address of the next instruction. It may also be one or two bytes of data immediately following the opcode. As the incoming instruction containing the forward reference is being processed, the addressing mode being used is usually well defined and indicates a unique opcode and instruction length. This is not the case, however, when the instruction making a forward reference is not relative, immediate, or indexed. It can then be either direct (one byte) or extended (two bytes). The difference depends entirely upon whether the final resolved address is greater than 255 or not, and at this point the address value is not known. Therefore, I established the following convention: When no addressing mode is specified in a forward reference, extended addressing will be assumed since it can refer to any position in memory. Some efficiency may be lost by not allowing the shorter form of addressing here, but most forward references are not to locations in the first 256 locations anyway, so the restriction is not a severe one. *[Using an equate (EQU pseudo operation) to define*

a page zero address ahead of its first use neatly solves the problem when using the assembler . . . **CH***]*

The second problem is the handling of arithmetic expressions containing a forward reference. It will require a solution built upon the logic developed so far, and will require additional coordination between the assembler and the loader. If the loader combines overlaying code with whatever is already in memory instead of using it as a replacement, the dummy code can be used to contain part of the information needed to resolve such an expression. This requires that the loader clears memory before starting the loading process and can correctly combine all single and double byte values. An expression with a forward reference is best illustrated by an example.

To load accumulator A with the address of five less than the sum of Y and Z (both unknown at this point) in TABLE1, the following code could be used:

```
EXAMPLE1 LDAA TABLE1+Y+Z-5
```

The expression (TABLE1+Y+Z–5) is evaluated as far as it can be, and the value of (TABLE1–5) is put out as the dummy code. When Y and Z are resolved (in any order), they are each put out as a correcting value. The loader then combines all three values (TABLE1–5, Y and Z) to arrive at the final value. If the loader simply adds each correction to what is currently in memory, a certain amount of caution is required when using the minus sign (–) or symbols with a negative value. A further consideration is that the assembler can only check the range of relative or indexed offsets for each occurrence of dummy or correcting code. When these are combined by the loader, they may exceed allowable offset ranges. It is up to the programmer to see that they do not exceed the limits. Since the requirements for handling forward references have been examined, and methods of achieving these requirements have been developed, the first new functions required by this assembler can now be considered.

The first major change is a new functional requirement. A routine is needed to record all pending forward references in a table. The table must contain a pointer to the symbol that was referenced, the address in the source program where the reference occurred and dummy code was generated, and the type of reference that was made (one byte or two, absolute or relative). Each time the routine is invoked, a new entry is added to the table. If the table is full, an error condition is raised. The table size was arbitrarily set at 25, which has proven to be more than sufficient if the source program is properly organized. *[In writing a new version of my monitor and text editor program (about 1500 source statements) with Jack's assembler this has proved to be* the case . . . **CH***]* The Set Forward Reference routine has the logical structure shown in listing 1. Note that in this structured programming notation there is only one logical entrance and one logical exit from each function. The logical exit (return) may have several physical locations, however, to reduce the amount of code used. *[The use of structured pseudocode such as that shown in listing 1 for program design was discussed in the article "Programming for the Beginner" by Ronald Herman published in the June 1976* BYTE, *page 22.]*

The second major change is another new functional requirement. A routine is needed to generate the correcting object code when the value of an item in the new table is resolved. This routine must search the entire forward reference table for pointers to the newly resolved symbol and generate a correcting reference for each. The address at which the dummy code was generated is retrieved from the table and used as the location for the correcting reference. If the previous reference was a relative instruction, the difference between the previous address and the resolved address is calculated and a single byte is put out. If the previous reference required a one byte address (indexed or immediate one byte), one byte is generated. If it required a two byte address (extended or immediate two byte), two are generated. As each correcting item is complete, its position in the forward reference table is cleared for future use. Therefore, a source program may have as many forward

---

Set Forward Reference

```
(start at beginning of forward reference table)
DO UNTIL (past end of table)
:   IF (current slot is an empty slot) THEN
:   :   (store pointer to symbol in symbol table)
:   :   (store current address)
:   :   (store type of reference)
:   :   (return)
:   ELSE
:   :   (increment to next slot in table)
:   ENDIF
ENDO
(signal error since forward reference table is full)
(return)
```

Listing 1: Set Forward Reference Routine. This routine is called once for each reference to a symbol that is not resolved in the symbol table and is not being used as a label.

references as it needs, but the number of forward references pending at any one time is limited by the size of the table. If any pointers to the symbol table are found at the end of the assembly, the undefined symbols are printed as potential errors. The Resolve Forward Reference routine has the logical structure shown in listing 2.

The third major change is a functional modification. A routine has already been defined to maintain the symbol table, but now it will also be required to recognize symbols that are unresolved, and to determine when the new functions developed above must be called. The first time an unresolved symbol is used, it will not be found in the symbol table, so it must be entered into the next available slot. If there is no available slot, a symbol table full error condition is raised. Set Forward Reference will then be called and a zero address returned. Additional references to this symbol will now be found in the table. Because each is flagged as unresolved in the symbol table, Set Forward Reference is again called and a zero address is returned. If the symbol is used as a label, the current address is entered into the symbol table and returned as the label's value. Resolve Forward Reference cannot be called at this time because the current instruction may modify the value of the symbol just resolved by equating it to a user defined value. Resolve Forward Reference must be called after the current line of source code has been completely processed. Now if another label for this symbol is encountered, a duplicate label error condition is raised.

To avoid using a separate byte of memory for the unresolved flag for each symbol in the symbol table, a zero address is used to indicate an unresolved symbol. This allows a zero value to be normally returned for such symbols as required by the above logic. The only effect this will have on source code programming is that a label at location zero cannot be resolved. Therefore, a symbol should not be defined at or equated to zero. Each reference to it would add an entry to the forward reference table and soon fill it up. Instead, a decimal value of zero should be used. The Maintain Symbol Table routine now has the logical structure shown in listing 3.

These functional modifications enable the assembler to completely process a source program in one pass. At this point the functional requirements for processing with only one IO interface can be considered.

Logically, each program to be assembled has three basic sets of data to be handled: the source code used as input to the assembler, the object code developed by the

Resolve Forward References

```
(start at beginning of forward reference table)
DO UNTIL (end of table)
:   IF (this reference to symbol table = current symbol) THEN
:   :  (save current address)
:   :  (current address = address from forward reference table)
:   :  IF (relative addressing) THEN
:   :  :  (calculate relative offset)
:   :  :  (write it out)
:   :  ELSE
:   :  :  IF (two bytes required) THEN
:   :  :  :  (write out symbol's high byte)
:   :  :  ELSE
:   :  :  ENDIF
:   :  :  (write out symbol's low byte)
:   :  ENDIF
:   :  (clear table position for future use)
:   :  (restore current address)
:   ELSE
:   ENDIF
:   (increment to next table position)
ENDDO
(return)
```

Listing 2: Resolve Forward Reference Routine. This routine is executed each time a symbol flagged as unresolved in the symbol table is used as a label.

Maintain Symbol Table

```
(start at beginning of symbol table)
DO UNTIL (past end of table)
:   IF (current symbol = incoming symbol) THEN
:   :  IF (incoming symbol is a label) THEN
:   :  :  IF (table address = zero) THEN
:   :  :  :  (load symbol into table)
:   :  :  :  (set symbols to be resolved condition)
:   :  :  :  (return table address)
:   :  :  ELSE
:   :  :  :  (signal error, duplicate symbol definition)
:   :  :  :  (return)
:   :  :  ENDIF
:   :  ELSE
:   :  :  IF (table address = zero) THEN
:   :  :  :  (call SET FORWARD ADDRESS)
:   :  :  :  (return zero value)
:   :  :  ELSE
:   :  :  :  (return table address)
:   :  :  ENDIF
:   :  ENDIF
:   ELSE
:   :  IF (current table slot is an empty space) THEN
:   :  :  (load symbol into table)
:   :  :  IF (incoming symbol is a label) THEN
:   :  :  :  (return current address)
:   :  :  ELSE
:   :  :  :  (call SET FORWARD REFERENCE)
:   :  :  :  (return zero value)
:   :  :  ENDIF
:   :  ELSE
:   :  :  (increment to next slot in symbol table)
:   :  ENDIF
:   ENDIF
ENDDO
(signal error, symbol table full)
(return)
```

Listing 3: Process Symbol Table Routine. This routine is called each time a symbol is used. If the symbol has a resolved value it is returned to the calling module, otherwise a zero value is returned.

assembler, and a program listing showing the object code that is generated for each line of source code. Physically, an IO interface is usually used to read and write data to or from one external device. The problem of handling three logical sets of data on one physical device has been solved by combining the generated object code with the listing. This reduces the data handling requirements to one input file (source code), and one output file (the combined listing) which the single interface can easily handle. The generated object code and its associated address that the loader uses are already produced on the left side of the listing as shown in listing 4. Therefore, it is only necessary to provide a way for the loader to directly read the listing to find the data that it needs.

How the listing is made available to the loader will vary with different physical configurations, and there are unlimited variations in possible hardware arrangements for small systems. This assembler was initially written to run on an ASR model Teletype with both tape reader and tape punch control characters enabled. A permanent machine readable copy of the listing can be produced on paper tape during the one pass operation. The listing, however, must be modified to enable the loader to find the required data when reading this tape.

In the 4 K version of this assembler, items that the loader needs can be surrounded by special nonprinting characters that can be recognized while scanning the listing. This isolates the needed items from source code, headings, comments and other extraneous items. These characters must not appear in any other context within the listing, so control characters (which are not valid assembler input) are the most logical choice. To the loader, then, the listing shown in listing 4a would appear as shown in listing 4b where the '(' and ')' represent nonprinting characters used to signify the start of an address, and the end of any bytes to load respectively. The high level form of the logical structure for a loader which would be able to use such a listing is shown in listing 5.

The combined assembly listing is produced by starting each line of source code in the middle of the page. The carriage return at the end of each line returns the current print position to the front of the line. The generated code is then printed in front of the source code. Before prompting for the next line of source code, a line feed is issued to position the paper for the next line. The result is most interesting to watch on the Teletype since one normally expects a line feed at the beginning or end of a line. The loader recognition characters are combined with "punch on" and "punch off" characters to keep all unwanted items off the object tape.

There are IO timing considerations worth mentioning that apply regardless of a system's configuration. If the input is coming from a tape file, input operations must stop to allow the generated code to be put out because the Motorola MIKBUG IO routines cannot support concurrent input and out-

Listing 4a: Combined listing as seen by the user. The generated code shown under the headings B1, B2, and B3 are to be loaded starting at the address shown under LOCN. This was generated at BYTE using Jack's assembler as modified for Carl Helmers' homebrew system.

Sample Listing

```
LOCN  B1  B2  B3
0001  CE  00  20    >CLEAR  LDX  #TBL          LOAD IX WITH START OF TBL
0004  6F  00        >LOOP   CLR  X             CLEAR TBL LOCATION
0006  08            >       INX                INCR TO NXT TABLE POSN
0007  8C  00  30    >       CPX  #TBL+LENGTH   END OF TABLE?
000A  26  F8        >       BNE  LOOP          IF NOT, DO IT AGAIN
```

Listing 4b: Combined listing as seen by the loader. The generated code and associated addresses are separated from the rest of the listing by special characters shown here as '(' and ')'. This is a simulation based on part (a) of this listing.

Loader's View of the Listing

```
LOCN  B1  B2  B3
(0001 CE  00  20)   >CLEAR  LDX  #TBL          LOAD IX WITH START OF TBL
(0004 6F  00)       >LOOP   CLR  X             CLEAR TBL LOCATION
(0006 08)           >       INX                INCR TO NXT TABLE POSN
(0007 8C  00  30)   >       CPX  #TBL+LENGTH   END OF TABLE?
(000A 26  F8)       >       BNE  LOOP          IF NOT, DO IT AGAIN
```

put. If stopping the input is not feasible for a given system, additional routines may have to be written to overcome the MIKBUG limitations. For interactive operation, there is rarely a problem as the assembler is usually faster than the user. For a truly minimum system, the source code can be entered by hand from the terminal, and the generated object code can be copied from the combined listing and loaded, again by hand. In this case, no intermediate files are required at all!

The functional changes necessary to allow the assembler to operate with only one IO interface are handled as modifications to the existing IO routine to enable it to create the listing as shown above. Specific changes may differ for various system configurations, and will not be shown here. We have now considered functional modifications to accommodate our requirements that the assembler operate in one pass, and that it use only one IO interface. The final requirement which will modify some of the original functions is that it run in only 4 K of memory.

The first thing to do is to look at all the functions available in existing M6800 assemblers and eliminate alternate ways of accomplishing things. In a tiny assembler, alternates are usually expendable overhead. For example, operands for the A and B accumulators may optionally be separated from the mnemonic instructions in 8 K assemblers (for example, LDA A instead of LDAA). This has been changed to require that the 'A' and 'B' be attached to the mnemonic which eliminates an additional scan for the accumulator operand. The choice of entering data in decimal, hexadecimal, octal or binary form has been reduced to decimal and hexadecimal. Alternate indicators of data type (eg: using a leading '$' or a trailing 'H' to indicate hexadecimal) are restricted to leading indicators to simplify parsing routines. Other such redundancies can be eliminated whenever a significant amount of code is needed to support multiple ways of doing the same thing.

The next area where space can be conserved is in table structure optimization. The largest table is the symbol table, so symbol size has been reduced from six characters to four. The first three and the last one character of a label of any size over four are combined as the character string entered into the table. This matches the four character size of the mnemonic plus accumulator instruction mentioned above, and the four character format of a hexadecimal address. The input routines may therefore be built around a standard four character format. Instead of using a binary search in the operand translation table, a hash index into the opcode table based on the mnemonic's first letter is used. *[See Terry Dolhoff's article on hashing techniques on page 18 of January 1977* BYTE.*]* This keeps the number of table entries searched to an average of 4.5 per opcode, and reduces the length of the stored mnemonic. By combining this access method with routines making use of the M6800's opcode structure, all of the 114 four character mnemonic instruction and assembler directives can be kept in a table keyed by 79 two character strings.

The next area where space can be conserved is in the simplification of complex routines at the possible cost of some efficiency or function. Because the symbol table in this assembler is smaller than those in full scale assemblers, and because we've got a dedicated personal machine to work with, complex hashing routines can be dropped in favor of a simple sequential search which burns a bit of computation time. Many complex error detection routines have been simplified to catch the most common occurrences of a problem. All physical IO operations are accomplished through the Motorola MIKBUG monitor rather than through uniquely written routines.

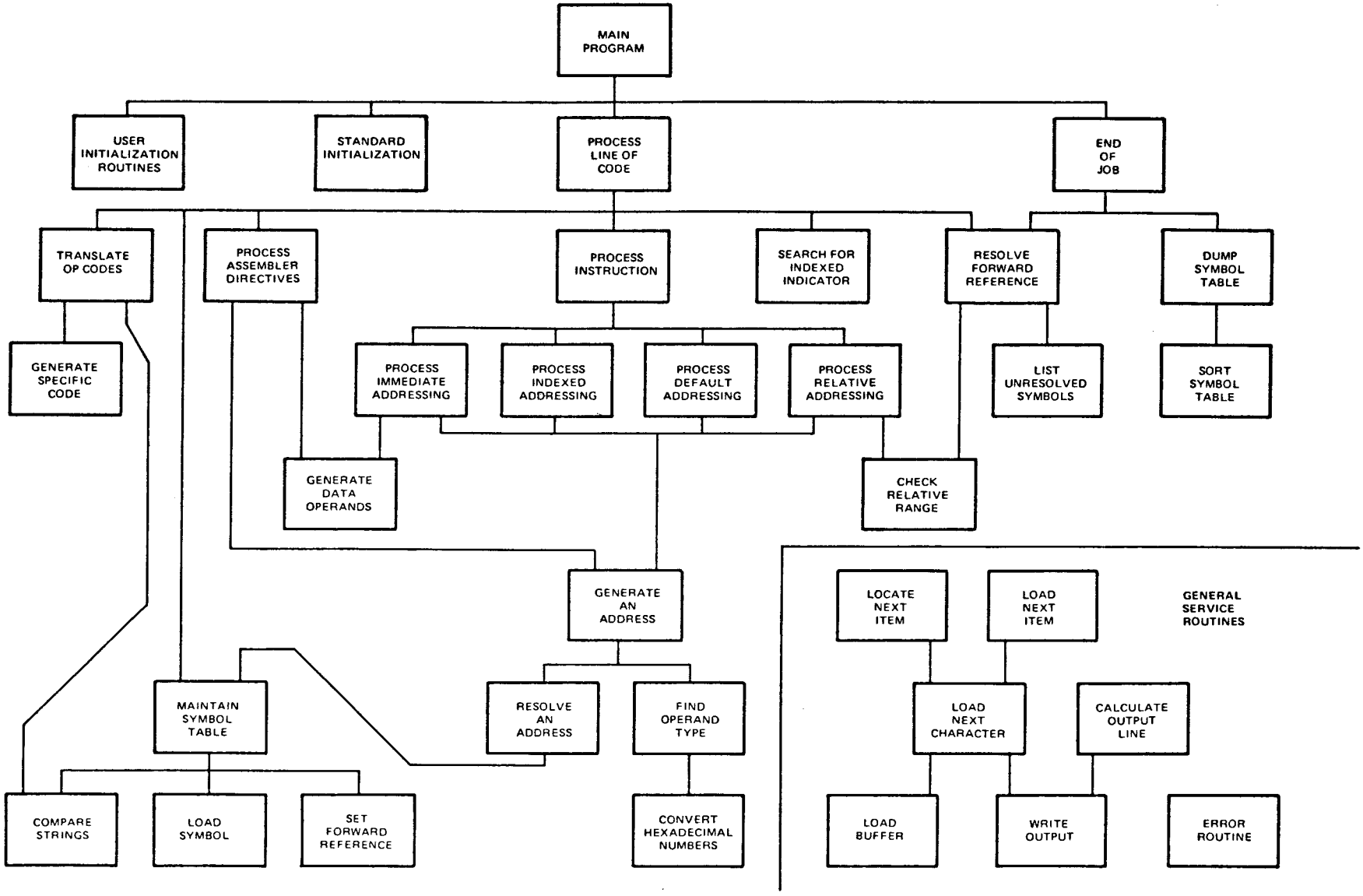
The final method used to save space is to simply eliminate functions that are of limited use and require significant code to implement. There is no provision for multiplication or division in the expressions within an instruction's operand. Multiple additions or subtractions are possible, however, and can often be used where multiplication

Loader Structure

```
DO UNTIL (end of listing)
:   DO UNTIL (character read is a '(')
:   :   (read character)
:   :   IF (character is null) THEN
:   :   :   (stop)
:   :   ELSE
:   :   ENDIF
:   ENDDO
:   (read address into index)
:   DO UNTIL (character read is a ')')
:   :   (read character)
:   :   IF (character is a blank) THEN
:   :   :   (read a byte from two hex characters)
:   :   :   (add incoming byte to indexed address)
:   :   :   IF (carry bit is set) THEN
:   :   :   :   (increment byte beyond indexed address)
:   :   :   ELSE
:   :   :   ENDIF
:   :   :   (increment index)
:   :   ELSE
:   :   ENDIF
:   ENDDO
ENDDO
(stop)
```

Listing 5: Logical structure for a directed loader capable of processing the listing shown in listing 4b. See listing next month for the implementation of a loader based in part on this structure.

Figure 3: The Final Assembler Structure. This is a detailed structure diagram of the assembler as it was actually written.



and division are usually found. There are no provisions for naming programs or for selecting options in generating the combined listing. Some error checking, such as invalid address or byte overflow, has been completely eliminated. The code generated is as compact as possible to fit in small machines, but is not relocatable. If it is desired in another location, it must be reassembled.

The result of all this compaction effort is an assembler whose basic executable code fits within 2170 bytes, and will run completely within 4096 bytes of programmable memory with a symbol table capacity of 200 symbols. Modifications have been made to almost all of the existing functions to accomplish the above changes. This completes the functional additions and modifications required for the assembler defined by the project specifications as shown in figure 3.

Implementing the Tiny Assembler

Listing 1: The external address table. This series of jumps is the sole interface to the outside world from Tiny Assembler 6800. As originally written, terminal oriented IO is done with these five external references to MIKBUG, Motorola's Monitor Program.

```
                        |*       ********** EXTERNAL ADDRESS TABLE
                        |*
0079D1   7E  E1  01     | CHROUT JMP  $E101
0079D3   7E  E1  AC     | CHRIN  JMP  $E1AC
0079D6   36             | HEXOUT PSHA
0079D7   8D  E0  67     |        JSR  $E067
0079DA   32             |        PULA
0079DB   7E  E0  6B     |        JMP  $E06B
0079DE   7E  E0  CC     | BLKOUT JMP  $E0CC
0079E1   7E  E0  7E     | STROUT JMP  $E07E
```

Notes on External Linkages

If MIKBUG is used (for example, in a Southwest Technical Products machine, or an expanded version of one of Motorola's development board kits) then this code should be used as is. If you interface to your own custom monitor, this code must be replaced by references to equivalent routines as follows:

CHROUT: The contents of the accumulator, treated as an ASCII character, are dumped on the output device.

CHRIN: The contents of the accumulator are defined by an ASCII code obtained from the input device.

HEXOUT: The contents of accumulator A are dumped to the output device as two hexadecimal encoded ASCII digits.

BLKOUT: A blank is dumped on the output device.

STROUT: A string, terminated by an EOT (hexadecimal 04) code is dumped on the output device. The index register points to the string on entry, and is updated during the operation.

When all the functional requirements for Tiny Assembler had been described in a well structured format, the final structural network of the program was realized, as given at the end of last month's introduction to this program. Combining figure 3 of page 66, April 1977 BYTE[1], with detailed table definitions, I was at long last ready to begin actual coding of the assembler itself.

Coding the Program

I was immediately confronted with two problems. First, I did not yet have an assembler to work with. Second, I had no M6800 machine available! I therefore decided to start by writing a cross assembler in PL/I. This would let me find and correct any logical bugs in the assembler's design. It would also provide an assembler capable of producing the final M6800 version of the developed program.

Because structured design techniques are language independent, no redesign was necessary for the development of the cross assembler. (In fact, assemblers written in 8080 or Z-80 code could be developed just as easily.) During testing of this program, several minor logical errors were corrected, and an operational design was available for the final version of the program. In a very real sense, the assembler was used to write and debug itself.

The next step was to translate each function from the PL/I code to M6800 assembler code. As each block of code was developed, it was added to existing functions and run through the cross assembler to catch any errors and to find the length of the generated code. Final debugging of each function had to be postponed until a machine was available. The code was basically developed in a top down manner starting with the mainline and working down the hierarchy of functions. However, the order of listing different parts of the program was

[1] Page 14 in this edition

M6800 Structured Code Conventions

	General Structure	M6800 Implementation

IF THEN ELSE	IF (test condition) THEN
	: (then processing)
	ELSE
	: (else processing)
	ENDIF
		IF TST CONDITION
		BNE ELSE
		*** } then processing
		*** }
		BRA ENDIF
		ELSE *** } else processing
		*** }
		ENDIF *** } continue
		*** }

DO WHILE	(set initial cond) DO WHILE (condition) : (do group) : (may change cond) ENDDO	* SET INITIAL COND DO TST CONDITION BNE ENDDO *** } *** } do group processing *** } BRA DO ENDDO *** ***
DO UNTIL	(set initial cond) DO UNTIL (condition) : (do group) : (may change cond) ENDDO	* SET INITIAL COND DO *** } *** } do group processing *** } TST CONDITION BNE DO ENDDO *** ***
CASE	A general case structure: SELECT (based on case) : CASE (condition 1) : : (process 1) : CASE (condition 2) : : (process 2) : CASE (condition 3) : : (process 3) ENDSELECT The case structure, recoded using the DO UNTIL to check case: SELECT : (point to first case) : DO UNTIL (selected condition) : : (advance to next case) : ENDDO : (execute case pointed to) ENDSELECT	JSR SELECT JMP ENDSELECT SELECT LDX #CASETABL LOOP JSR NXTCASE Calculate condition TST CONDITION BEQ EXECUTE INX INX BRA LOOP Move to next CASE EXECUTE JSR X Execute CASE RTS Return from select * CASETBL BRA CASE1 } BRA CASE2 } Table of CASE Conditions BRA CASE3 } * CASE1 *** } *** } First CASE procedure RTS } CASE2 *** } *** } Second CASE procedure RTS } CASE3 *** } *** } Third CASE procedure RTS } ENDSELECT *** } *** } continue *** }

*Table 1: The general programming structures shown in the table above have been implemented as shown in the right hand column where *** represents the particular code which is executed within these control structures.*

influenced by the one pass design of the assembler. All data declarations, output text, tables, and commonly called subroutines were placed at the front of the listing to reduce the number of forward references. The program mainline follows the section containing common subroutines.

While the M6800 has a very powerful instruction set, it cannot directly support the functions shown in the pseudocode used in the listings given as examples. Therefore, control functions such as DO, THEN, ELSE, UNTIL, WHILE and CASE need to be performed by in line routines of M6800 instructions. Table 1 shows some of the common functional structures with their associated assembly language routines. The basic structures of these in line routines will

Text continued on page 24

Table 2: Hexadecimal object code. This table contains the complete object code of Tiny Assembler 6800 version 3.0, assembled at an origin of 0000 hexadecimal in memory address space. This version can be used if necessary to load Tiny Assembler 6800 by hand. This type was set by machine from the same data which was used to create the bar code listings of figure 1. Note, however, that to make this version readable, fixed length lines were used (as opposed to variable length lines in figure 1).

Address	Data															
0000	7E	07	A4	00	20	00	00	00	00	20	20	20	00	00	00	00
0010	00	00	2A	20	20	20	00	0D	00	04	00	0D	00	00	00	00
0020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0040	58	20	20	20	0D	00	00	2A	2A	2A	20	45	52	52	4F	52
0050	20	2D	20	00	0A	00	00	20	04	0D	0A	0A	0A	00	00	00
0060	4C	4F	43	4E	20	42	31	20	42	32	20	42	33	20	0A	0A
0070	00	00	04	0D	00	00	12	04	0D	0A	0A	0A	00	00	2A	2A
0080	2A	20	55	4E	52	45	53	4F	4C	56	45	44	20	49	54	45
0090	4D	53	3A	0D	0A	00	00	04	0D	0A	00	00	2A	2A	2A	20
00A0	53	59	4D	42	4F	4C	53	2C	20	53	4F	52	54	3F	0D	0A
00B0	0A	00	04	00	00	20	20	20	00	00	00	00	00	00	00	00
00C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00F0	00	00	06	06	11	17	08	1F	04	23	03	26	03	00	00	00
0100	00	29	03	2C	02	00	00	2F	04	00	00	32	02	34	02	36
0110	02	00	00	38	05	3D	0A	47	07	00	00	00	00	4F	01	00
0120	00	00	00	00	00	42	41	1B	50	44	43	89	10	44	44	8B
0130	10	4E	44	84	10	53	4C	48	30	53	52	47	30	43	43	24
0140	40	43	53	25	40	45	51	27	40	47	45	2C	40	47	54	2E
0150	40	48	49	22	40	49	54	85	10	4C	45	2F	40	4C	53	23
0160	40	4C	54	2D	40	4D	49	2B	40	4E	45	26	40	50	4C	2A
0170	40	52	41	20	40	53	52	8D	40	56	43	28	40	56	53	29
0180	40	42	41	11	50	4C	43	0C	50	4C	49	0E	50	4C	52	4F
0190	30	4C	56	0A	50	4D	50	81	10	4F	4D	43	30	50	58	8C
01A0	11	41	41	19	50	45	43	4A	30	45	53	34	50	45	58	09
01B0	50	4E	44	00	F0	4F	52	88	10	51	55	00	F0	43	42	01
01C0	F0	43	43	02	F0	44	42	03	F0	4E	43	4C	30	4E	53	31
01D0	50	4E	58	08	50	4D	50	4F	31	53	52	8D	31	44	41	86
01E0	10	44	53	8E	11	44	58	CE	11	53	52	44	30	45	47	40
01F0	30	4F	50	01	50	52	41	8A	10	52	47	04	F0	53	48	36
0200	00	55	4C	32	00	4D	42	05	F0	4F	4C	49	30	4F	52	46
0210	30	54	49	3B	50	54	53	39	50	42	41	10	50	42	43	82
0220	10	45	43	0D	50	45	49	0F	50	45	56	0B	50	54	41	87
0230	20	54	53	8F	21	54	58	CF	21	55	42	80	10	57	49	3F
0240	50	41	42	16	50	41	50	06	50	42	41	17	50	50	41	07
0250	50	53	54	4D	30	53	58	30	50	58	53	35	50	41	49	3E
0260	50	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0270	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0280	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0290	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
02A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
02B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
02C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
02D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	01	01
02E0	01	01	39	00	00	00	00	00	00	00	00	00	00	00	00	00
0780	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0790	7E	E1	D1	7E	E1	AC	36	BD	E0	67	32	7E	E0	6B	7E	E0
07A0	CC	7E	E0	7E	BD	02	DE	CE	02	61	6F	00	08	8C	07	8F
07B0	26	F8	86	39	B7	02	DE	CE	00	B8	DF	2A	7F	00	1C	7F
07C0	00	31	CE	07	B7	FF	A0	48	8E	A0	7F	CE	00	59	BD	07
07D0	A1	BD	09	2A	BD	0F	5A	7E	E0	E3	CE	00	B8	86	3E	BD
07E0	07	90	BD	07	93	81	21	26	08	8C	00	B8	27	F4	09	20
07F0	F1	A7	00	91	1B	26	01	39	8C	00	F0	27	E5	81	20	2B
0800	E1	81	5F	22	DD	08	20	DA	DF	24	DE	2A	96	17	91	1B
0810	26	14	7D	00	31	27	03	BD	08	B2	86	01	97	31	BD	07
0820	DA	CE	00	B8	DF	2A	A6	00	97	17	08	DF	2A	DE	24	39
0830	96	17	91	0A	27	01	39	BD	08	08	20	F6	DE	0A	DF	05
0840	DF	07	CE	00	05	D6	17	D1	0A	27	1D	D1	1B	27	19	C1
0850	2B	27	15	C1	2D	27	11	C1	2C	27	0D	E7	00	8C	00	08
0860	27	01	08	BD	08	08	20	DD	39	97	53	DF	20	CE	00	44
0870	BD	07	A1	DE	20	7C	00	03	39	7D	00	03	23	01	39	96
0880	16	D6	31	C1	01	27	19	C1	02	26	07	97	B6	7C	00	31
0890	20	1A	C1	03	26	07	97	B7	7C	00	31	20	0F	BD	08	B2
08A0	DE	2F	DF	B3	96	16	97	B5	86	02	97	31	DE	2F	08	DF
08B0	2F	39	DF	24	CE	00	73	BD	07	A1	CE	75	30	6D	00	09
08C0	26	FB	86	01	BD	07	90	96	B3	BD	07	96	96	B4	BD	07
08D0	96	C6	01	CE	00	B5	D1	31	2B	17	26	0A	86	01	BD	07
08E0	90	86	14	BD	07	90	BD	07	9E	BD	07	9E	BD	07	9E	20
08F0	08	BD	07	9E	A6	00	BD	07	96	08	5C	C1	04	26	D7	96
0900	31	81	04	26	0A	86	01	BD	07	90	86	14	BD	07	90	BD
0910	07	9E	DE	2F	DF	B3	DE	24	86	0A	BD	07	90	96	0A	97
0920	B5	97	B6	97	B7	86	01	97	31	39	CE	00	04	D6	0A	7F
0930	00	03	7F	00	3F	E7	00	08	8C	00	0A	26	F8	96	1B	97
0940	17	BD	08	08	91	1B	27	E2	81	2A	27	DE	91	0A	27	09
0950	BD	08	3C	7C	00	26	BD	0D	1B	BD	08	30	91	1B	27	3D
0960	BD	08	3C	86	02	97	19	BD	09	DD	86	04	97	19	7D	00
0970	03	22	B7	96	18	81	0E	26	04	BD	08	B2	39	D6	2E	81
0980	0F	26	05	BD	0E	DB	20	15	BD	08	30	81	23	26	05	97
0990	04	BD	08	08	BD	08	30	BD	09	A3	BD	0A	CA	BD	0E	15
09A0	7E	09	2A	DE	2A	96	17	81	58	26	1E	A6	00	91	0A	27
09B0	0E	91	1B	27	0A	81	2B	27	06	81	2D	27	02	20	0A	86
09C0	58	97	04	86	30	97	17	20	13	A6	00	08	91	0A	27	0C
09D0	91	1B	27	08	81	2C	26	F1	A6	00	97	04	39	96	05	81
09E0	41	2C	06	86	41	BD	08	69	39	81	5A	2F	06	86	41	BD
09F0	08	69	39	84	1F	4A	48	CE	00	F1	4D	27	04	08	4A	20
0A00	F9	E6	01	D7	29	E6	00	4F	58	49	58	49	CE	01	25	DF
0A10	3D	DB	3E	99	3D	D7	3E	97	3D	CE	00	06	DF	32	7D	00
0A20	29	26	06	86	42	BD	08	69	39	DE	3D	DF	34	BD	0A	A5
0A30	7D	00	11	23	09	BD	0A	4E	7D	00	11	23	01	39	08	08
0A40	08	08	DF	3D	CE	00	06	DF	32	7A	00	29	20	D0	A6	02

Address	Data															
0A50	97	2E	A6	03	16	C4	0F	D7	1D	44	44	44	44	97	18	D6
0A60	08	D1	0A	26	0B	7D	00	1D	22	3A	81	02	23	33	20	34
0A70	81	03	22	2D	7D	00	1D	22	28	C1	41	27	19	C1	42	26
0A80	20	D6	2E	4D	26	06	CB	01	D7	2E	20	0A	81	03	27	02
0A90	CB	30	CB	10	D7	2E	4C	85	02	26	09	86	05	97	18	20
0AA0	03	7F	00	11	39	36	DE	20	9E	38	7F	00	11	96	19	97
0AB0	3C	DE	32	9E	34	34	32	A1	00	26	09	08	7A	00	3C	26
0AC0	E5	7C	00	11	DE	20	9E	38	32	39	96	18	D6	2E	81	03
0AD0	23	0F	D7	16	BD	08	79	96	18	81	04	26	03	BD	0B	05
0AE0	39	96	04	81	23	26	04	BD	0B	4F	39	81	58	26	06	CB
0AF0	20	BD	0B	8B	39	91	0A	26	06	CB	10	BD	0B	9D	39	86
0B00	44	BD	08	69	39	BD	0B	D5	7D	00	3F	23	08	96	0D	97
0B10	16	BD	08	79	39	CE	00	2F	96	0C	D6	0D	E0	01	A2	00
0B20	C0	01	92	2C	D7	16	BD	0B	2F	01	01	BD	08	79	39	91
0B30	2C	27	06	81	FF	27	02	20	04	98	16	2A	11	86	47	BD
0B40	08	69	DE	2F	7D	00	1F	22	01	09	DF	2F	DF	B3	39	96
0B50	18	81	01	27	06	86	43	BD	08	69	39	37	BD	0B	D5	33
0B60	D7	16	BD	08	79	96	05	81	27	27	04	BD	0B	77	39	96
0B70	06	97	16	BD	08	79	39	7D	00	1D	23	07	96	0C	97	16
0B80	BD	08	79	96	0D	97	16	BD	08	79	39	37	BD	0B	D5	33
0B90	D7	16	BD	08	79	96	0D	97	16	BD	08	79	39	37	BD	0B
0BA0	D5	33	7F	00	3C	7D	00	0C	26	0D	7D	00	3F	22	08	96
0BB0	18	81	03	27	02	20	05	CB	20	7C	00	3C	D7	16	BD	08
0BC0	79	7D	00	3C	27	07	D6	0C	D7	16	BD	08	79	D6	0D	D7
0BD0	16	BD	08	79	39	7F	00	26	BD	0C	2D	BD	0C	1A	96	17
0BE0	81	2B	27	05	81	2D	27	01	39	97	09	DE	0C	DF	0E	BD
0BF0	08	08	BD	0C	2D	BD	0C	1A	96	09	81	2B	27	0E	96	0F
0C00	90	0D	97	0D	96	0E	92	0C	97	0C	20	D2	96	0F	9B	0D
0C10	97	0D	96	0E	99	0C	97	0C	20	C4	7D	00	03	22	0D	7D
0C20	00	1F	22	08	7D	00	10	23	03	BD	0D	1B	39	7C	00	10
0C30	7C	00	1F	7F	00	0C	7F	00	0D	96	17	81	24	26	0A	BD
0C40	08	08	BD	08	3C	BD	0C	DD	39	7F	00	1F	81	26	26	03
0C50	BD	08	08	BD	08	3C	CE	00	05	A6	00	91	0A	27	08	81
0C60	30	2B	3D	81	39	22	39	91	0A	27	31	8C	00	09	27	2C
0C70	85	30	26	06	86	4A	BD	08	69	39	84	0F	A7	00	C6	09
0C80	D7	3C	D6	0D	96	0C	DB	0D	99	0C	7A	00	3C	26	F7	EB
0C90	00	99	2C	D7	0D	97	0C	08	A6	00	20	CB	7F	00	10	39
0CA0	CE	00	40	DF	32	CE	00	05	DF	34	BD	0A	A5	7D	00	11
0CB0	23	0A	DE	2C	DF	05	DF	07	7F	00	10	39	CE	00	12	DF
0CC0	32	CE	00	05	DF	34	BD	0A	A5	7D	00	11	23	0E	DE	2F
0CD0	96	18	81	04	26	01	09	DF	0C	7F	00	10	39	CE	00	05
0CE0	A6	00	91	0A	27	34	8C	00	09	27	2F	81	30	2B	06	81
0CF0	46	22	02	20	06	86	4A	BD	08	69	39	81	41	2B	02	8B
0D00	09	84	0F	A7	00	96	0C	D6	0D	58	49	58	49	58	49	58
0D10	49	EB	00	D7	0D	97	0C	08	20	C6	39	96	05	81	27	26
0D20	01	39	81	40	22	06	86	4B	BD	08	69	39	CE	02	DF	C6
0D30	01	C1	C9	26	06	86	45	BD	08	69	39	DF	34	DF	3A	CE
0D40	00	05	DF	32	BD	0A	A5	7D	00	11	23	2C	DE	3A	7D	00
0D50	26	22	0C	EE	04	DF	0C	9C	2C	26	03	BD	0D	B3	39	EE
0D60	04	9C	2C	27	06	86	46	BD	08	69	39	BD	0D	95	7C	00
0D70	1E	DF	3A	EE	04	DF	0C	39	DE	3A	6D	00	26	0C	BD	0D
0D80	95	7D	00	26	22	03	BD	0D	B3	39	DE	3A	08	08	08	08
0D90	08	08	5C	20	9C	DE	3A	9F	38	9E	05	AF	00	9E	07	AF
0DA0	02	7D	00	26	23	06	9E	2F	AF	04	DF	27	EE	04	DF	0C
0DB0	9E	38	39	86	01	7F	00	3C	CE	02	61	81	19	2F	06	86
0DC0	49	BD	08	69	39	81	19	26	08	86	48	BD	08	69	7F	00
0DD0	03	E6	00	26	06	E6	01	26	02	20	0E	4C	08	08	08	08
0DE0	08	D6	3C	CB	05	D7	3C	20	D2	96	04	81	58	26	02	6A
0DF0	04	81	23	26	07	7D	00	1D	22	02	6A	04	96	18	81	04
0E00	26	02	6C	04	9F	38	9E	3A	AF	00	9E	2F	31	AF	02	9E
0E10	38	7C	00	3F	39	7D	00	1E	22	01	39	86	01	CE	02	61
0E20	DF	22	7D	00	03	22	07	81	19	2F	04	7F	00	1E	39	36
0E30	7D	00	1C	23	05	BD	0E	A9	20	61	DE	22	A6	00	E6	01
0E40	91	27	26	57	D1	28	26	53	DE	2F	DF	36	BD	08	B2	DE
0E50	22	EE	02	DF	2F	DE	22	6D	04	2B	08	26	14	DE	27	E6
0E60	04	D7	16	BD	08	79	DE	27	E6	05	D7	16	BD	08	79	20
0E70	1A	DE	27	A6	04	E6	05	CE	00	2F	E0	01	A2	00	DE	2F
0E80	09	DF	2F	D7	16	BD	0B	2F	BD	08	79	DE	36	DF	2F	DE
0E90	22	6F	00	6F	01	6F	02	6F	03	6F	04	DE	22	08	08	08
0EA0	08	08	DF	22	32	4C	7E	0E	22	DE	22	E6	00	26	05	E6
0EB0	01	26	01	39	EE	00	86	04	97	3C	A6	00	BD	07	90	08
0EC0	7A	00	3C	26	F5	BD	07	9E	DE	22	A6	02	BD	07	96	A6
0ED0	03	BD	07	96	CE	00	93	BD	07	A1	39	CE	0F	4E	5D	27
0EE0	05	08	08	5A	20	F8	BD	08	30	AD	00	39	BD	0B	D5	9F
0EF0	38	9E	0C	DE	27	AF	04	9F	B3	9E	38	39	7F	00	1D	20
0F00	06	7F	00	1D	7C	00	1D	BD	0B	D5	BD	0B	77	96	17	81
0F10	2C	27	01	39	BD	08	08	20	EE	BD	0B	D5	96	0D	97	3C
0F20	26	01	39	BD	08	08	97	16	BD	08	79	7A	00	3C	20	F0
0F30	BD	0B	D5	DE	0C	DF	2F	DF	B3	39	DE	2F	DF	B3	BD	0B
0F40	D5	96	30	D6	2F	9B	0D	D9	0C	97	30	D7	2F	39	20	9C
0F50	20	AA	20	C5	20	AB	20	D8	20	E0	CE	00	78	BD	07	A1
0F60	CE	00	6E	BD	07	A1	7C	00	1C	86	01	97	1E	BD	0E	15
0F70	BD	0F	79	86	0A	BD	07	90	39	CE	00	98	BD	07	A1	BD
0F80	07	93	81	59	26	03	BD	0F	C8	CE	02	DF	86	05	97	3C
0F90	DF	20	CE	00	93	BD	07	A1	DE	20	86	04	97	1A	A6	00
0FA0	27	25	BD	07	90	08	7A	00	1A	26	F3	BD	07	9E	A6	00
0FB0	BD	07	96	08	A6	00	BD	07	96	08	BD	07	9E	BD	07	9E
0FC0	7A	00	3C	26	D5	20	C5	39	CE	02	DF	7F	00	11	20	12
0FD0	C6	06	A6	00	A1	06	2D	06	26	12	08	5A	20	F4	08	5A
0FE0	26	FC	6D	06	26	EA	7D	00	11	26	DD	39	D7	1A	D7	11
0FF0	A6	00	E6	06	A7	06	E7	00	08	7A	00	1A	26	F2	20	E2

Notes on Memory to Memory Operation

(Personal use notes by Carl Helmers)

As used in my homebrew 6800 system at BYTE, Jack Emmerichs' assembler is set for memory to memory operation without any source input or object file output peripheral operations at all. During the assembly process, the only normal IO operation is output of the listing to my Asciscope terminal at 2400 bps. This form of operation is extremely fast, and would be even faster if it were not limited much of the time by the data rate of the listing to the terminal. The speed is attained by use of large regions of memory instead of peripherals. At this writing, my 6800 is equipped with a little more than 24 K bytes of working main memory which is allocated to the software development process in the first 28 K of memory address space as follows:

0000-00FF	*Scratchpad direct addressing region used by all programs (like registers on a big machine).*
0100-0FFF	*IO Device Address Allocations.*
1000-1FFF	*Write protected programmable memory for the Interactive Manipulator Program (monitor, IO routines, text editor).*
2000-2FFF	*Assembler object file output built from 2000 upward, assembly time stack built from 2FFF downward.*
3000-5FFF	*Source string text area prepared by IMP's editor.*
6000-60F1	*Master initialization copy of directly addressed scratchpad memory contents.*
60F2-6FFF	*Tiny Assembler 6800 relocated by hand.*

In memory to memory operation, the patches include programmed simulations of the nonexistent parts of IO, as well as a memory loader program which is patched into an appropriate part of the assembler to get data prior to conversion into external form.

An Input Simulator

The normal CHRIN input routine of the MIKBUG monitor, or other similar systems software, simply returns the next character from the terminal device, which might be a Teletype, video terminal, etc. For memory to memory operation this is accomplished by use of a pointer variable and a routine which fetches the next byte and increments the pointer. If the pointer exceeds a maximum value, the assembly is terminated with an error message, so my program assumes that the end of the assembly will be signalled by an END statement prior to running out of data in the text string. End of line to the assembler is indicated by the same ASCII carriage return (hexadecimal value D) which marks the end of line for the text editor. The input simulator I use is illustrated in listing 5; this routine also assumes that initialization of the assembly sets the pointer named IPTR equal to the beginning address of the text buffer. It also is used to echo the original source listing of the assembly to the logical output devices. This always includes the 2400 bps video terminal, but might also include my Teletype when I need a hard copy listing and set an appropriate monitor flag.

The Output Simulator

Output for the character images of the object code of a line of assembled text are handled exactly as in MIKBUG using routines that accomplish the same purpose. The data goes to a terminal or Teletype hard copy device in human readable form but is not stored in machine readable form as assumed by Jack. However, I wanted to be able to load the memory region from 2000 to 2FFF with data generated by the assembler, rather than record the character format data on tape and later load it with a program such as that supplied by Jack in listing 4.

Thus using the source listing of the assembler, I identified the location in the program (the beginning of WRITE) at which all data for a given byte of memory is ready in binary form, then patched in the loader of listing 6. This loader is patched into the assembler's WRITE routine, using code shown by comments at the end of listing 6. This loader has been designed to calculate a pointer into the output buffer area with the value 2000+X where X is the present output program counter value minus the starting program counter value for the assembly. This technique works well provided that the starting program counter can be properly set. This is accomplished by setting the starting program counter to an initial value of FFFF during initialization, then making a requirement that the first statement of the assembly reference the starting address of the object code. This requirement is met by making the first statement of each assembly be an ORG statement defining the starting location of the program being assembled.

Figure 1: *Bar code representation of Tiny Assembler 6800. This figure, spread over several pages, contains the complete bar code representation of Tiny Assembler 6800, version 3.0. The copy was sent to BYTE using Kansas City (BYTE) Standard tape format, and was processed by Walter Banks and his associates at the University of Waterloo to prepare the code in the form seen here. Table 2 was prepared from the same data using a different program for the Photon phototypesetter. The bar code text here uses variable length records in the frame format described earlier in BYTE. This format is repeated here, along with the object code format, for reference by those individuals experimenting with bar code input techniques:*

Standard Format

- *Sync character, binary 10010110*
- *Frame checksum (arithmetic sum of all bytes in frame ignoring carry)*
- *Relative record identification, 8 bit ascending integer*
- *Length of frame, "n"*
- *Data of frame, total of "n" bytes, as follows:*

Specific Data Format

- *High order address for first byte of data*
- *Low order address for first byte of data*
- *"n-2" bytes of data loaded beginning at address of first byte of data*

These records are absolute binary data, an image of the Tiny Assembler 6800 beginning at location 0000 and extending to location 0FFF in memory address space. An extended segment of null (hexadecimal 00) data in the table area of the assembler is skipped, so not every byte in the assembler is found in this bar code representation (the same applies to table 2).

Listing 2: The main procedure loop of Tiny Assembler 6800. The assembly begins by calling an optional subroutine, assumed to exist at the start of the symbol table region of memory, to perform user initialization. As assembled and presented in this article, the subroutine is a dummy procedure represented by an RTS (hexadecimal 39) at location 02DE. The JSR STBL-1 (hexadecimal BD 02 DE at location 7A4) is the main entry point of the assembler. It can also be patched to a permanent initialization routine outside the assembler if the user's machine has more memory than the 4 K assumed by this program.

After user initialization, the symbol table is cleared. If the symbol table was present with a transient user initialization routine starting at location 02DE, this routine is replaced by a dummy return instruction for the second and subsequent invocations of the assembler following a load of the code.

Location ENTRY is the normal "continue assembly" entry point for the program. If assembling line by line, it is possible to invoke the assembler for one or more lines, issue an END pseudooperation to terminate the assembly. Then it is possible to continue operation by returning to ENTRY from your monitor (this listing assumes Motorola's MIKBUG). Note that it is not safe to do such an END operation if there are any unresolved references, unless symbol table sorting is suppressed.

Overhead of interfacing Tiny Assembler to the Motorola MIKBUG monitor is part of the initialization that precedes the call to CMD, the main assembly procedure. After an END pseudooperation, control returns from CMD, EOJ is called to perform housekeeping operations, and MIKBUG is reentered at $EOE3 (hexadecimal).

This listing and listing 1 were produced using the PL/I cross assembler version of Tiny Assembler 6800.

> **Note:** In order to interface this program to a nonMIKBUG monitor, the details of the stack initialization at locations 07C2 to 07C8 may need to be changed. In the earlier version 2 of Tiny Assembler used at BYTE, for example, the stack was initialized to fall at the end of the 4 K byte region in which the assembler was located. In the newer version 3, adapted to use at BYTE, we initialize the stack pointer to the end of the output buffer area of memory.

```
                     | *
                     | *
                     | *         ***************************************************************
                     | *                       MAIN PROCEDURE LOOP
                     | *
(07A4)  BD  02  DE   | MAIN    JSR  STBL-1        EXECUTE USFR ROUTINES
(0001)  07  A4       |
(07A7)  CE  02  61   |         LDX  #FTBL        CLEAR THE SYMBOL TABLE AND THE
(07AA)  6F  00       | CLR1    CLR  X              FORWARD JUMP TABLE
(07AC)  08           |         INX
(07AD)  8C  07  BF   |         CPX  #STBL+1200
(07B0)  26  FA       |         BNE  CLR1
(07B2)  86  39       |         LDAA #$39          BLOCK FURTHER
(07B4)  B7  02  DE   |         STAA STBL-1          EXECUTION OF USER PROCS
                     | *
                     | *        THIS IS THE CONTINUE ENTRY POIN!
                     | *
(07B7)  CE  00  B8   | ENTRY   LDX  #BUF1          INITIALIZE
(07BA)  DF  2A       |         STX  NXCHAR           INCOMING CHARACTER POINTER
(07BC)  7F  00  1C   |         CLR  EOJFL          CLEAR FLAGS
(07BF)  7F  00  31   |         CLR  P_POS
(07C2)  CE  07  B7   |         LDX  #ENTRY         LOAD SECONDARY ENTRY POINT FOR MIKBUG MONITOR
(07C5)  FF  A0  48   |         STX  $A048
(07C8)  8E  A0  7F   |         LDS  #$A07F         LOAD STACK POINTER FOR MIKBUG MONITOR
(07CB)  CE  00  59   |         LDX  #HEADR         WRITE OUT
(07CE)  BD  07  A1   |         JSR  STROUT           HEADING
(07D1)  BD  00  00   |         JSR  CMD            MAIN PROCEDURE LOOP - CALL PROC-COMMANDS
(07D4)  BD  00  00   |         JSR  EOJ            CALL END OF JOB
(07D7)  7E  E0  E3   |         JMP  $E0E3          JUMP TO MONITOR MAINLINE
```

Figure 1, part 1:

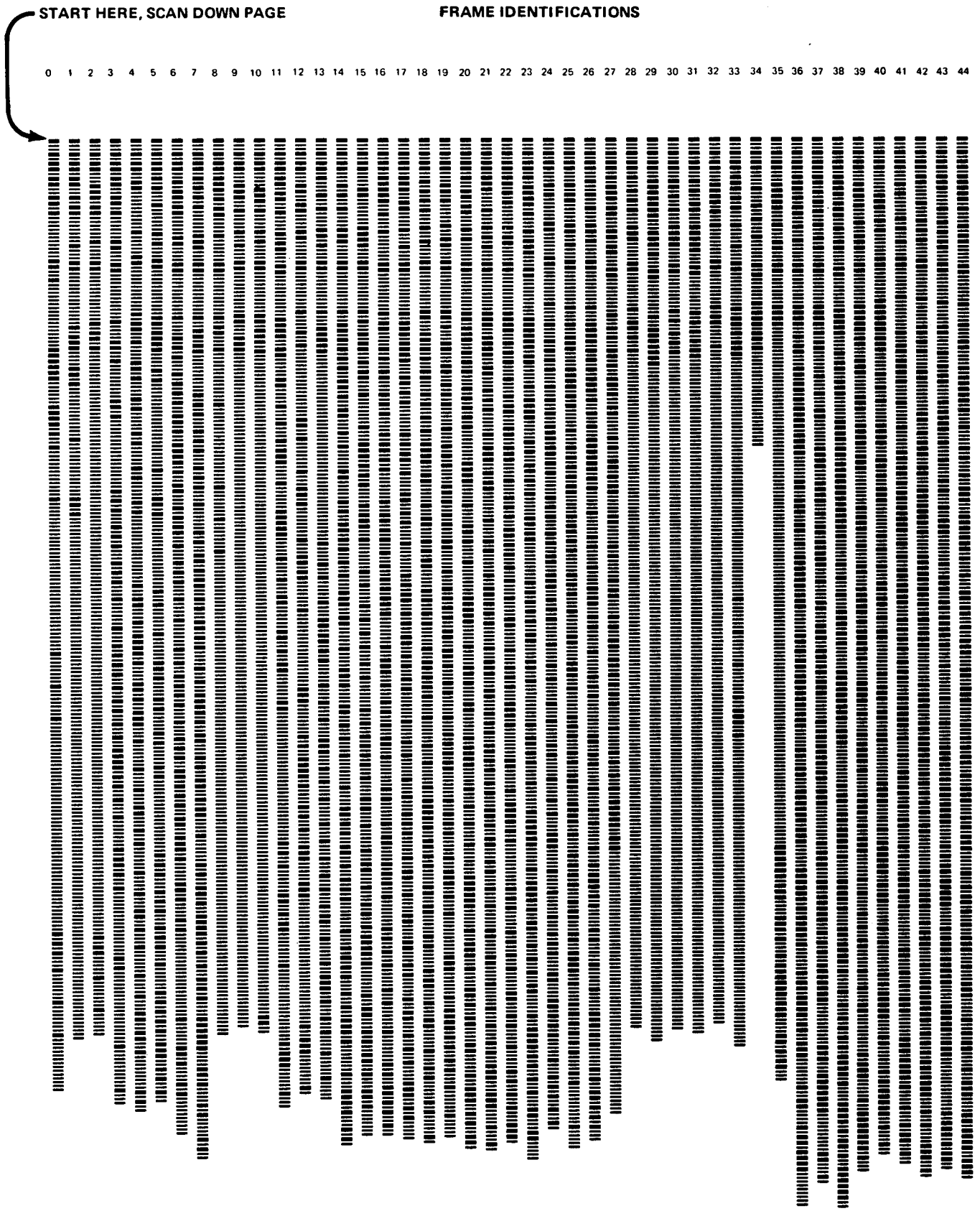


Figure 1, part 2:

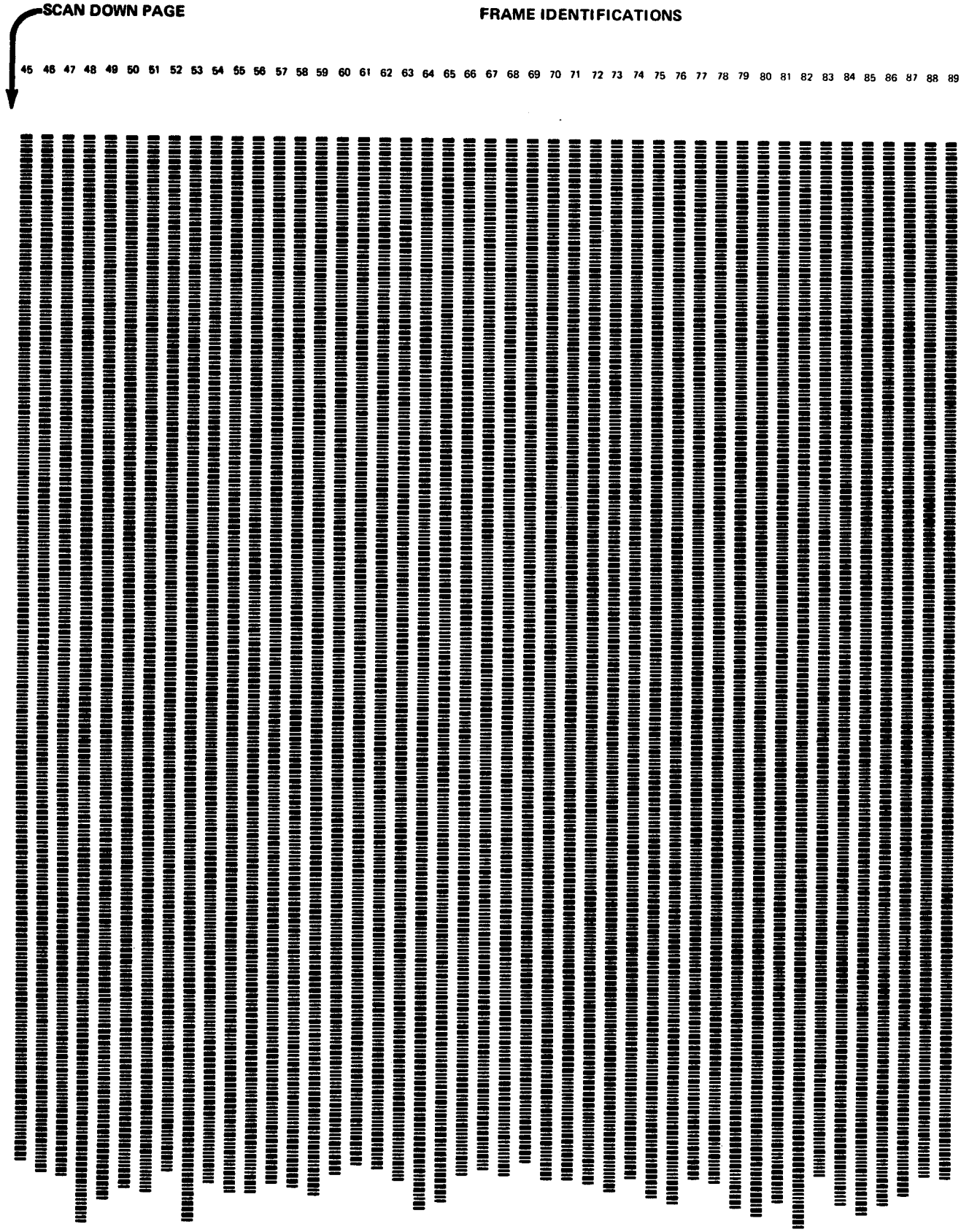
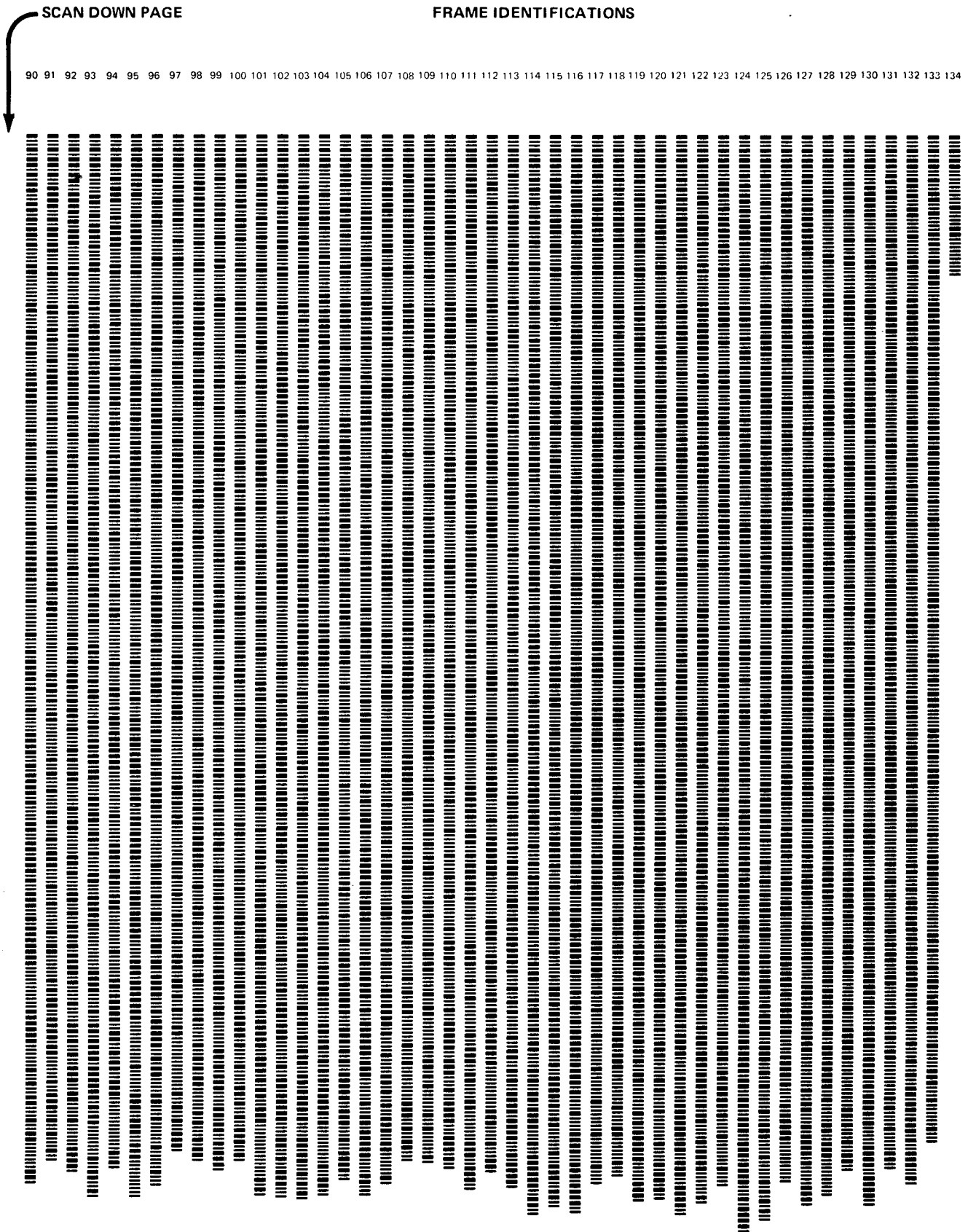


Figure 1, part 3:



Continued from page 17

be similar from one microprocessor machine language to another even though specific instructions will differ. Like pseudocode, they can be combined into extremely complex procedures. One special structure is not shown in table 1: the code to handle external program linkages.

External linkage can be handled in two ways. One is to equate the name of an external symbol to an address and use it throughout the program. This will generate the absolute address of the external symbol

Table 3: Error messages from Tiny Assembler 6800 version 3.0. The format of labels, operation codes and operands of Tiny Assembler 6800 is a subset of the full Motorola Assembler specification. These error messages indicate conditions which Tiny Assembler could not handle; those conditions which are noted as "terminal" tend to propagate through the rest of the assembly generating additional errors. Error codes are single letters, listed here with some explanation.

ERROR A

Verb does not start with an alphabetic character. A verb is an operation code or pseudooperation of the assembly language. This error can result from a prior error, or from mistyping source data.

ERROR B

Verb not in table. This is the primary indication of an invalid mnemonic operation code.

ERROR C

Verb does not allow immediate addressing. An attempt was made to use immediate addressing (signified by a '#' character immediately preceding the operand) with an operation that has no immediate mode. For example,

```
ASR        #00PS
```

would generate this message.

ERROR D

Invalid verb modifier.

ERROR E

Symbol table overflow. This is a terminal error, in the sense that code following a symbol table overflow can have spurious errors. To recover, the program being assembled should be partitioned so that a smaller number of symbols is required in assembling each part. Note that the capacity of Tiny Assembler 6800 is 200 symbols in this version.

ERROR F

Label already in use. The following are several sets of duplicate labels (as interpreted by Tiny Assembler 6800):

In Label Field	In Symbol Table
THINKS THIS THIMBLES	THIS
AOUT1X AOUT2X AOUT3X	AOUX

While symbols can be arbitrarily long, the first three characters and last character must be unique.

ERROR G

Relative branch greater than 128 locations away. This error occurs if the target of a branch instruction is too far away for a signed 8 bit displacement from the present location counter value. This can usually be fixed by using a jump instruction. For example, if

```
BMI      TARG
```

gives error G, then it might replaced by the complementary condition and a jump as follows

```
BPL      DUM1
JMP      TARG
DUM1 *** continue code ***
```

ERROR H

Forward reference table full. This is a warning message that indicates imminent collapse of an assembly if another reference to an undefined symbol is found prior to the resolution of any previously unresolved reference. Tiny Assembler 6800 has a table of 25 unresolved forward references maximum.

ERROR I

Forward reference table overflow. This is the terminal message that indicates an attempt to exceed 25 unresolved forward references. After this error is noted, there will typically be numerous spurious error messages. Eliminate forward references to undefined labels to cure this problem. One way to eliminate an unresolved forward reference is to define the data or code involved prior to its first reference.

ERROR J

Invalid character in hexadecimal or decimal number. Only two types of numeric constant are allowed by Tiny Assembler 6800: up to four decimal digits, or a dollar sign ($) character followed by up to four hexadecimal digits.

ERROR K

Invalid symbol. A symbol in the label field of an operation begins with a number, or an invalid character is detected in a symbol.

for each instruction which references it. A better way is to create a table of linkages to external symbols and have all references go through the table. In this way, only one statement must be changed if the location of an external symbol should change. If all references are in the form of subroutine calls, a simple jump statement can be used to transfer control from the table to the external location. In this assembler, all calls to the monitor's IO routines go through the table shown in listing 1.

The CASE statement is somewhat more complex. The pseudocode for CASE is shown in table 1 in two forms. The pseudocode can be broken down into two functions: calculate the CASE conditions, and

Listing 3: Symbol table allocations. The $01 code at 02DE is a NOP instruction which gets replaced by a $39 to block further attempts at initialization after the assembler has cleared the symbol table. A dummy transient initialization routine of three NOPs and an RTS illustrates how such code would be entered into the assembler's source; in a real situation, one would assemble the initialization routine separately and then move the code in on top of the existing assembly code of this dummy.

```
                        *       ************** SYMBOL TABLE
                        *
02DE  01                        FCB   $01
02DF                    STBL    RMB   0
                        *
                        *         LABEL NAME  CHAR (4)
                        *         LABEL ADDR  CHAR (2)
                        *
02DF  01                        NOP                 SPACE FOR
02E0  01                        NOP                 USER DEFINED
02E1  01                        NOP                 ROUTINES
02E2  39                        RTS
078F                            ORG   STBL+1200     SET FOR 200 SIX BYTE ENTRIES
078F  00                        FCB   $00           TABLE STOP BYTE
```

Listing 4: An example of a loader program. This loader is based on the functional structure shown in the first part of this article, and is one of several listings accompanying this article which show how Tiny Assembler 6800 works.

```
LOCN B1 B2 B3

0000            >*****************************************
0000            >*                                       *
0000            >*     TINY ASSEMBLER DIRECTED LOADER    *
0000            >*                                       *
0000            >*****************************************
0000            >*          EXTERNAL REFERENCES          *
0000            >*****************************************
0000            >       ORG   0
0000 7E 00 00   >       JMP   CLEAR
0F79            >       ORG   $0F79
0F79            >BEGIN  EQU   *           LABEL BEGINING OF CODE
0F79 7E E0 E3   >BUG    JMP   $E0E3       MONITOR MAINLINE
0F7C 7E E0 47   >INADDR JMP   $E047       ADDRESS IN
0F7F 7E E0 55   >INHEX  JMP   $E055       HEX BYTE IN
0F82 7E E1 AC   >INCHAR JMP   $E1AC       CHARACTER IN
0F85 7E E0 7E   >PUTSTR JMP   $E07E       PUT STRING OUT
0F88            >*****************************************
0F88            >*          GENERAL DECLARES             *
0F88            >*****************************************
000D            >CR     EQU   $D          DEFINE CONTROL
000A            >LF     EQU   $A            CHARACTERS
0004            >EOT    EQU   4
0F88 0D 0A 00   >PROMPT FCB   CR,LF,,     DEFINE OUTPUT STRINGS
0F8B 00
0F8C 20 42 45   >       FCC   6, BEGIN
0F8F 47 49 4E
0F92 04         >       FCB   EOT
0F93 0D 0A 00   >DONE   FCB   CR,LF,,
0F96 00
0F97 20 44 4F   >       FCC   5, DONE
0F9A 4E 45
0F9C 04         >       FCB   EOT
0F9D 0D 0A 00   >ERRMSG FCB   CR,LF,,
0FA0 00
0FA1 20 42 41   >       FCC 15, BAD CHAR ERROR
0FA4 44 20 43
0FA7 48 41 52
0FAA 20 45 52
0FAD 52 4F 52
0FB0 04         >       FCB   EOT
0FB1            >
0FB1            >
0FB1            >*****************************************
0FB1            >*            START LOADER               *
0FB1            >*****************************************
0FB1 CE 00 00   >CLEAR  LDX   #0          BLANK OUT FIRST 4K
0001 0F B1
0FB4 6F 00      >LOOP   CLR   X             OF MEMORY
0FB6 8C 0F 79   >       CPX   #BEGIN        EXCEPT LOADER
0FB9 27 00      >       BEQ   GO
0FBB 08         >       INX
0FBC 20 F6      >       BRA   LOOP
0FBE CE 0F 88   >GO     LDX   #PROMPT     WRITE OUT
0FBA 03
0FC1 BD 0F 85   >       JSR   PUTSTR        PROMPT STRING

0FC4 BD 0F 82   >FNDADR JSR   INCHAR      READ NEXT CHARACTER
0FC7 81 00      >       CMPA  #0          IF END OF TAPE THEN
0FC9 27 00      >       BEQ   EOJ           GO TO END PROC
0FCB 81 28      >       CMPA  #'(         IF NOT '(' THEN
0FCD 26 F5      >       BNE   FNDADR        TRY NEXT CHAR
0FCF BD 0F 7C   >       JSR   INADDR      ELSE READ START ADDR INTO IX
0FD2 BD 0F 82   >NXTBYT JSR   INCHAR      READ NEXT CHAR
0FD5 81 29      >       CMPA  #')         IF END OF BLOCK THEN
0FD7 27 EB      >       BEQ   FNDADR        GO FIND NEXT BLOCK
0FD9 81 20      >       CMPA  #$20        IF NOT A BLANK THEN
0FDB 26 00      >       BNE   ERROR         GO TO ERROR PROC
0FDD BD 0F 7F   >       JSR   INHEX       READ NEXT CHAR INTO 'A'
0FE0 E6 00      >       LDAB  X           LOAD CURRENT VALUE INTO 'B'
0FE2 1B         >       ABA               ADD 'A' AND 'B'
0FE3 A7 00      >       STAA  X           STORE ANSWER IN MEMORY
0FE5 24 00      >       BCC   COUNT       IF CARRY THEN
0FE7 09         >       DEX               . INCREMENT HIGH ORDER BYTE
0FE8 6C 00      >       INC   X
0FEA 08         >       INX
0FEB 08         >COUNT  INX               SET IX FOR NEXT BYTE
0FE6 04
0FEC 20 E4      >       BRA   NXTBYT      LOOK FOR NEXT CHAR IN BLOCK
0FEE            >
0FEE            >
0FEE            >*****************************************
0FEE            >*             ERROR ROUTINE             *
0FEE            >*****************************************
0FEE CE 0F 9D   >ERROR  LDX   #ERRMSG     PUT OUT ERROR MSG
0FDC 11
0FF1 BD 0F 85   >       JSR   PUTSTR
0FF4 7E 0F 79   >       JMP   BUG         RETURN TO MONITOR
0FF7            >*****************************************
0FF7            >*           END OF JOB ROUTINE          *
0FF7            >*****************************************
0FF7 CE 0F 93   >EOJ    LDX   #DONE       PUT OUT FINISH MSG
0FCA 2C
0FFA BD 0F 85   >       JSR   PUTSTR
0FFD 7E 0F 79   >       JMP   BUG         RETURN TO MONITOR
1000            >*
1000            >*
1000            >       END

*** UNRESOLVED ITEMS:

*** SYMBOLS, SORT?

Y
BEGN 0F79  BUG  0F79  CLEA 0FB1  COUT 0FEB  CR   000D
DONE 0F93  EOJ  0FF7  EOT  0004  ERRG 0F9D  FREE 0FEE
FNDA 0FC4  GO   0FBE  INAD 0F7C  INCH 0F82  INHX 0F7F
LF   000A  LOOP 0FB4  NXTT 0FD2  PROT 0F88  PUTS 0F85

*
```

Listing 5: The input simulator routine. This routine, called exactly like the equivalent routine of MIKBUG, simulates an input operation by fetching the next byte of text from the text area. This byte is written to the terminal device to echo the progress of the assembly. The character string output sequence referenced on input overrun error is designed to use a string which terminates with hexadecimal FF. For users of the equivalent STROUT routine of MIKBUG, replace the last character of the message string with hexadecimal 04.

```
LOCN B1 B2 B2          CE
2000            >      ORG  $2000
2000            >* ORIGIN CHOSEN TO EASE RELOCATION BY USERS OF LISTING
2000            >*
2000            >* MEMORY TO MEMORY ASSEMBLY INPUT ROUTINE
2000            >* FOR USE WITH JACK EMMERICHS'
2000            >*          "TINY ASSEMBLER 6800" PROGRAM
2000            >*
2000            >*    WRITTEN BY CARL HELMERS CIRCA NOVEMBER 15 1976
2000            >*    ORIGINAL VERSION HAND ASSEMBLED
2000            >*    THIS VERSION EDITED AND ASSEMBLED FEBRUARY 14 1977
2000            >*       AS ILLUSTRATION FOR JACK EMMERICHS' ARTICLE
2000            >*       IN MAY 1977 BYTE...
2000            >*
2000            >* EQUATES OF VARIABLES
00F4            >TEMX EQU $F4      TEMPORARY INDEX SAVE AREA
00F2            >IPTR EQU $F2      POINTER TO INPUT TEXT AREA
2000            >*
2000            >*
2000            >*   ASSEMBLER PATCH FOR OUTPUT TO TERMINAL
FFFF            >OUTC EQU $FFFF [REPLACE WITH ACTUAL VALUE]
2000            >*
2000            >*
2000            >*   IMP6800 MONITOR EQUATES
2000            >*      FOR STRING OUTPUT PACKAGE DATA
1041            >PSTR EQU  $1041      PUT STRING ROUTINE
0020            >TXTX EQU  $20        PUT STRING POINTER INPUT ARGUMENT
00FC            >CLEA EQU  $FC        PSTR CLEAR SCREEN COMMAND
00FF            >STOP EQU  $FF        PSTR END OF STRING COMMAND
0007            >BELL EQU  $7         ASCII BELL
000D            >CR   EQU  $D         ASCII CARRIAGE RETURN
2000            >*
2000            >*
2000            >*  EXECUTABLE CODE OF NXCH "NEXT CHARACTER" ROUTINE
2000 DF F4      >NXCH STX TEMX
2002 DE F2      >     LDX IPTR
2004 A6 00      >     LDAA 0,X      A := @IPTR  [GET NEXT INPUT CHARACTER]
2006 08         >     INX           IPTR := IPTR + 1;
2007 8C 5F FF   >     CPX  #$5FFF   IF IPTR = $5FFF THEN
200A 27 00      >     BEQ  OOPS     [OOPS WE HAVE AN ERROR]
200C DF F2      >     STX  IPTR
200E 36         >     PSHA
200F 37         >     PSHB
2010 BD FF FF   >     JSR  OUTC     PRINT THE CHARACTER
2013 33         >     PULB
2014 32         >     PULA
2015 39         >     RTS           NORMAL RETURN
2016            >*
2016            >* ERROR DISASTER
2016 CE 00 00   >OOPS LDX  #OOPM    X := [ADDRESS OF MESSAGE]
2019 DF 20      >     STX  TXTX     TXTX := ADDRESS OF MESSAGE
201B BD 10 41   >     JSR  PSTR     PRINT MESSAGE [IMP MONITOR ROUTINE]
201E 20 FE      >GAGA BRA  GAGA     RUN AMUCK IN CONTROLLED LOOP TILL RESET
                >OOPM FCB  CLEA,BELL,CR,BELL,CR
2020 FC 07 0D
2023 07 0D
2017 20 20
2025 49 4E 50   >     FCC  22,INPUT BUFFER OVERRUN !
2028 55 54 20
202B 42 55 46
202E 46 45 52
2031 20 4F 56
2034 45 52 52
2037 55 4E 20
203A 0D
203B 07 0D 07   >     FCB  BELL,CR,BELL,CR,BELL,STOP
203E 0D 07 FF
2041            >     END


*** END - UNRESOLVED ITEMS:

*** SYMBOLS:

BELL 0007  CLEA 00FC  CR   000D  GAGA 201E  IPTR 00F2
NXCH 2000  OOPM 2020  OOPS 2016  OUTC FFFF  PSTR 1041
STOP 00FF  TEMX 00F4  TXTX 0020
```

then execute the proper case procedure. In table 1, the second form of the CASE construct and the detailed code shows this restructuring. The implementation of this structure resembles a computed GO TO, and is shown in detail in table 1. Unlike a computed GO TO, however, there is only one logical exit from the routine. This is a return from subroutine (RTS) instruction at the end of the select procedure. In the assembler, the CASE structure is used to select which procedure is to be executed for assembler directives. In our model CASE of table 1, a branch around the structure is shown for completeness; this is not necessarily required in all cases.

During the process of translating each function into assembler code, it was not difficult to follow the coding techniques developed above. Control logic is not complex because control instructions generally fall into two simple categories. First, within a given function there are almost always branch instructions which are limited in range from −128 to +127 bytes relative to the location of the next instruction. Each function is usually compact enough to fit within this range. Second, between functions control instructions are always subroutine calls. The only way for one function to affect another is through a jump to subroutine, or by changing values in program variables or machine registers. This makes the program very easy to understand and modify. *[Proven in practice here at BYTE where Jack's assembler has been undergoing a bit of customization.]*

Trying It On a Real Machine

The problem of machine access was finally solved by my friendly local Milwaukee Computer Store which donated the use of a demonstrator Southwest Technical Products M6800 for debugging the final version of the assembler. The final phase of the project could now begin.

The main line and IO modules were debugged first so that the program could communicate with the user. Functions were tested in the same top down order used in design. Once a function worked properly it was almost never necessary to make further corrections based on the testing of lower level modules. Listing 2 shows the main line procedure of the assembler, with some comments on its operation. Listing 3 shows the symbol table allocations with comments on transient user initialization performed by code in the symbol table area.

Examples of how the assembler works are shown in listings 4, 5 and 6. Listing 4 is a loader program based in part on the functional description in pseudocode developed earlier and shown in listing 5 of part 1 of this article. The style of this loader reflects its use as an example of assembler source code more than its possible use as an efficient, compact loader. Listings 5 and 6 supplied by Carl Helmers are examples of what can be done to adapt the assembly program.

Listing 6: An object file output simulator. This routine, which is patched into the assembler's WRITE routine of hexadecimal location 8B2 (68B2 in BYTE's relocated version), is used to directly load the output of the assembler into the output buffer starting at location 2000. A starting address pointer is initialized in BASE by the first address generated in the assembly, and all other addresses are calculated by subtracting the starting address pointer from the current address then adding the origin of the output buffer. The load routine results in a memory image of the final program out of the assembler, complete with all patches and fixups required by the one pass nature of the assembler. After an assembly, the MOVE routines of the monitor used at BYTE relocate the program at its intended position in memory. No loader or linkage editor is used in this system, and absolute text is all it produces; relocation of programs is done by reassembly with a different ORG value at the start of the text. [Note . . . This memory loader routine assumes that the output area of the assembly is cleared by initialization prior to entry into the assembler. If this is not done, change locations 203A and 203B to NOP instructions (hexadecimal 01). In either form, this routine will not calculate undefined forward reference expressions properly. For an example of the correct code for the undefined forward reference expression special case, see locations FE0 to FEB of Jack's loader program in listing 4. . . . CH]

```
LOCN B1 B2 B2
2000            >     ORG  $2000
2000            >* MEMORY TO MEMORY ASSEMBLY LOADER
2000            >*    MODIFICATION FOR JACK EMMERICHS' "TINY ASSEMBLER"
2000            >*    PATCHED INTO WRITE ROUTINE DUMP DATA TO
2000            >*    LOCATION 2000-2FFF FOR LATER MOVEMENT TO FINAL LOC
2000            >*    BASE MUST BE INITIALIZED TO "FFFF" PRIOR TO ENTRY
2000            >*
2000            >* WRITTEN BY CARL HELMERS, DECEMBER 9 1976 AS UPDATE
2000            >*    TO ORIGINAL MOD CIRCA NOVEMBER 15 1976
2000            >*
2000            >* VERSION OF FEBRUARY 14 1977 ASSEMBLED FOR JACK EMMERICHS
2000            >*    MAY 1977 BYTE ARTICLE TO ILLUSTRATE ADAPTATION.
2000            >*    EQUATES TO ASSEMBLER VARIABLES ADJUSTED FOR VERSION
2000            >*    3 ASSEMBLY OF "TINY ASSEMBLER 6800"
2000            >*
2000            >*    ORG TO LOCATION $2000 TO EASE RELOCATION ...
2000            >*       ACTUAL LOCATION IS INSIDE IMP6800 MONITOR BUT
2000            >*       IT'S EASIER TO HAND RELOCATE IF LOW ORDER ZEROS
2000            >*       ARE FORCED INTO ADDRESSES
2000            >*
2000            >*    TAPE #200A LOC 186 FILE=LOADER
2000            >*
0031            >PPOS EQU $31 ASSEMBLER PRINT POSITION
00FC            >BASE EQU $FC CURRENT BASE ADDRESS, FFFF INITIALLY
2000            >*    NOTE:  ASSEMBLIES MUST START WITH "ORG" TO SET BASE
00F6            >PTR  EQU $F6 OUTPUT POINTER, 2000-2FFF RANGE
00B5            >BYT1 EQU $B5 ASSEMBLER'S OUTPUT STRING ADDRESS
00B3            >OUTA EQU $B3 ASSEMBLER'S CURRENT OUTPUT ADDRESS
00F4            >TEMX EQU $F4 TEMPORARY SAVE AREA
00FE            >ISX3 EQU $FE REALLOCATION OF ASSEMBLER VARIABLE TO AVOID
2000            >*                          MONITOR CONFLICT
2000            >*
2000 DF FE      >LDER STX  ISX3     SAVE INDEX AS IN ORIGINAL CODE
2002 DE FC      >      LDX  BASE
2004 8C FF FF   >      CPX  #$FFFF  IS BASE EQUAL TO NULL INITIALIZATION?
2007 26 00      >      BNE  NORL    IF NOT THEN GO TO NORMAL EXECUTION
2009 96 B4      >      LDAA OUTA+1  ELSE COMPUTE A BASE ADDRESS
200B 80 00      >      SUBA #0
200D 97 FD      >      STAA BASE+1        BASE = OUTA - $2000
200F 96 B3      >      LDAA OUTA
2011 82 20      >      SBCA #$20
2013 97 FC      >      STAA BASE
2015 96 B4      >NORL LDAA OUTA+1     CALCULATE OUTPUT BUFFER ADDRESS
2008 0C
2017 90 FD      >      SUBA BASE+1         PTR = OUTA - BASE
2019 97 F7      >      STAA PTR+1               = OUTA(N) - OUTA(0) + $2000
201B 96 B3      >      LDAA OUTA
201D 92 FC      >      SBCA BASE
201F 97 F6      >      STAA PTR
2021 96 F7      >      LDAA PTR+1   IS PTR LESS THAN $2FFF?
2023 80 FF      >      SUBA #$FF
2025 96 F6      >      LDAA PTR
2027 82 2F      >      SBCA #$2F
2029 24 00      >      BCC  NOLO    IF NOT THEN DO NOT WRITE DATA AT ALL
202B CE 00 B5   >      LDX  #BYT1   POINT TO ASSEMBLER OUTPUT STRING
202E C6 01      >      LDAB #1      SIMULATION OF ORIGINAL OUTPUT
2030 D1 31      >LDR1 CMPB PPOS
2032 2C 00      >      BGE  LDR2
2034 A6 00      >      LDAA 0,X
2036 DF F4      >      STX  TEMX
2038 DE F6      >      LDX  PTR
203A AB 00      >      ADDA 0,X     COMBINE NEW DATA WITH OLD
203C A7 00      >      STAA 0,X
203E 08         >      INX          POINT TO NEXT OUTPUT BYTE
203F DF F6      >      STX  PTR
2041 DE F4      >      LDX  TEMX    PREPARE FOR ITERATION
2043 08         >LDR2 INX
2033 0F
2044 5C         >      INCB
2045 C1 04      >      CMPB #4
2047 26 E7      >      BNE  LDR1
2049 CE 00 73   >NOLO LDX  #$73     SET INDEX AS IN ORIGINAL CODE
202A 1E
204C 7E 68 B7   >      JMP  $68B7   JUMP BACK TO ASSEMBLER AFTER PATCH
204F            >* THE FOLLOWING ABSOLUTE CODE IS PATCHED INTO THE
204F            >* ASSEMBLER AT ITS LOCATION STARTING AT ADDRESS 6000...
204F            >*    6882   7E XX XX     WRITE JMP LDER
204F            >*       (NOTE XX XX IS ACTUAL ADDRESS OF LDER,
204F            >*       NOT DUMMY 2000 USED FOR THIS ASSEMBLY ONLY)
204F            >*
204F            >     END

*** END - UNRESOLVED ITEMS:


*** SYMBOLS:

BASE 00FC  BYT1 00B5  ISX3 00FE  LDER 2000  LDR1 2030
LDR2 2043  NOLO 2049  NORL 2015  OUTA 00B3  PPOS 0031
PTR  00F6  TEMX 00F4
```

Summary and Reflection

What has been presented here as a chronological listing of project activities was in fact accomplished with a fair amount of overlap. For example, a complete description of the Motorola M6800 language was not available to me until after the cross assembler was partially completed. Final table optimization was not achieved until after a version of the program was operational on the M6800 machine. Ideas came up late in development which could increase capacity or decrease overhead. In all such cases, the structured format of the program allowed modifications to be easily incorporated into existing code.

There are several possible modifications that are too major to be incorporated into the initial version of this assembler and which fall outside of the original specifications. These can be considered for future projects or may suggest still other modifications to some readers.

For example, the biggest constraint upon the use of a small assembler is the small size of the symbol table. One way to overcome this problem is to be able to delete a symbol when it is no longer needed and reuse the table space. A good way to do this is to implement a BEGIN statement. This would push the location of the most recent symbol added to the table onto a table stack. All symbols defined below this point would belong to this BEGIN block. An END statement would pull the top location off the table stack and delete all symbols below but not including this location. If there were no items in the table stack, end of program would be signaled. The nesting level of BEGIN blocks is only limited by the table stack size. This is similar to the way BEGIN and PROCEDURE structures work in PL/I and should be quite easy to implement in a sequentially searched symbol table.

If more memory were available, this assembler could be combined with an editor

Listing 7: A multifaceted sample of various features with comments. The comments in this assembly refer to features of the assembler; the actual code is arbitrary and is not intended as a coherent program. Errors listed in table 3 are illustrated at the end of the assembly.

```
LOCN B1 B2 B3
0000            >*************************************************************
0000            >*          S A M P L E   ILLUSTRATES WITH COMMENTS          *
0000            >*************************************************************
0100            >      ORG  256            DEFAULT IS A DECIMAL NUMBER
0003            >CUNT  EQU  &3             "&" INDICATES DECIMAL, TOO
0100 BE 00 00 >STRT  LDS  #STAK           A FORWARD REFERENCE
0103 FE 00 00 >      LDX  ADDR           ANOTHER FORWARD REFERENCE
0106 C6 03    >      LDAB #CUNT          IMMEDIATE ADDRESSIN IF "#"
0108 96 0A    >BACK LDAA 10             DIRECT ADDRESSING IF <= 255 ($FF)
010A A1 02    >      CMPA 2,X            INDEXED ADDRESSING
010C 27 00    >      BEQ  FUND           RELATIVE (BRANCH) ADDRESSING
010E 09       >      DEX                 IMPLIED ADDRESSING
010F 5A       >      DECB                ACCUMULATOR ADDRESSING
0110 26 F6    >      BNE  BACK
0112 A1 00    >      CMPA X              DEFAULT 0 OFFSET INDEXED ADDR.
0114 3E       >      WAI                 WAIT FOR INTERRUPT
0115 BD 00 00 >FUND JSR  SRTN           JUMP TO SUBROUTINE (CALL)
010D 07
0118            >*     (NOTE FORWARD REFERENCE IS RESOLVED HERE AND
0118            >*        PATCHED INTO EARLIER BRANCH.)
0118 B6 01 00 >      LDAA STRT           EXTENDED ADDRESSING
011B 7E 01 15 >      JMP  FUND           JUMP UNCONDITIONAL
011E 16       >SRTN TAB                 --- THESE ARE COMMENTS --
0116 01 1E
011F BA 00 00 >      ORAA BYTE           FORWARD REF, EXTENDED ADDR.
0122 39       >      RTS
0123            >      RMB  10             RESERVE 10 BYTES MEMORY
012D            >STAK RMB  1              STAK NOW RESOLVED
0101 01 2D
012E 80         >BYTE FCB  $80            FORM CONSTANT BYTE,"$" FOR HEX
0120 01 2E
012F            >*     (NOTE RESOLUTION OF EXTENDED ADDRESS,
012F            >*           AND BACKWARD PATCH...)
012F 10 00 04 > FCB $10,,4,32,$32       FORM SEVERAL CONSTANT BYTES,
0132 20 32
0134            >*     WITH NULL INDICATED BY ",,"; NOTE USE OF "$" FOR HEXA
0134            >*     DECIMAL, AND ONLY ONE SPACE AT START OF LINE .  FIRST
0134            >*     CHARACTER IS "*" FOR COMMENT, " " FOR UNLABELLED LINE
0134            >*     AND ALPHANUMERIC FOR SYMBOL DEFINITION.
0134 49 54 45 >DATA FCC 5,ITEMS         FORM 5 CHARACTER STRING
0137 4D 53
0139 01 34    >ADDR FDB DATA            FORM 2 BYTE ADDRESS
0104 01 39
013B
013B            >* NOW SOME ERRORS
013B 86 20      >      LDAA #1011$20      USE OF "!" AS DELETE CHAR.
*** ERROR - A >LDAA 5                    LEADING BLANK MISSING
013D
*** ERROR - B >      LDA A,5             "A" SEPARATE FROM "LDA"
013D
*** ERROR - C >      STAA #DATA          IMMEDIATE ADDRESS IMPOSSIBLE
013D
*** ERROR - D >      STAA 5,W            "W" SHOULD BE "X"
013D
*** ERROR - F >DATA FCB  10              DUPLICATE LABEL (SYMBOL)
013D
*** ERROR - G >      BRA  $FFFF          BRANCH OUT OF RANGE
013D 20
*** ERROR - J >      LDAA 12FF           INVALID DECIMAL CONSTANT
013D
*** ERROR - J >ITEM EQU  $00FX           INVALID HEXADECIMAL CONSTANT
000F
00FF            >      EQU  $FF            CORRECTION TO ERROR
013D 96 00      >      LDAA
013F            >* NULL OPERAND FIELD ABOVE EQUATED TO ZERO
*** ERROR - K >      LDAA %1010          MOTOROLA'S BINARY NOT SUPPORTED
013F
*** ERROR - K >      LDAA @3             MOTOROLA'S OCTAL NOT SUPPORTED
013F
013F B6 FF F1 >      LDAA STRT+2-DATA+$23       VALID EXPRESSION
0142 B6 00 00 >      LDAA STRT/2-DATA*$23       INVALID EXPRESSION
0145            >*   (NOTE: INVALID EXPRESSION RESULTS IN BAD DATA, NO
0145            >*   MESSAGE AND SEVERAL SPURIOUS ENTRIES IN SYMBOL
0145            >*   TABLE.)
0145            >*
0145 7E 00 00 >      JMP  UNDEFINED FORWARD REFERENCE LEFT UNRESOLVED AT
0148            >      END


*** END - UNRESOLVED ITEMS:

STR2
DAT3
UNDD

*** SYMBOLS:

ADDR 0139  BACK 0108  BYTE 012E  CUNT 0003  DAT3 0000
DATA 0134  FUND 0115  ITEM 00FF  LDAA 013D  SRTN 011E
STAK 012D  STR2 0000  STRT 0100  UNDD 0000
```

to produce an assembling editor. The editor's insert, find, delete and modify functions in addition to an assembling function and a reasonably sized source file buffer would provide quite a nice program development package using about the same amount of memory required by most other assemblers.

In a large system, the assembler could be reassembled for a higher memory location and then used to load the developed object code directly into available lower memory locations. This would eliminate many of the IO timing considerations mentioned last month and the need for a loader program.

While the current package can handle large and complex programs (such as itself) and could be developed into even greater things, it is initially intended to meet more modest requirements. As a Tiny Assembler it is well suited for small programs that can be entered by hand or from fairly short input tapes on a Teletype. It is an excellent tool for the interactive development of functional blocks for a large structured program. Different methods of coding a given process can be tried to allow the user to examine the generated code. Changes can be made as errors are flagged or new ideas come up by simply backing up the program counter and recoding. (Here the loader must replace what is in memory rather than add to it, or changes will be added to errors.) Finally, if your 6800 based machine only has 4 K of memory, this is probably the only assembler that you can use at all.

Expanding the Tiny Assembler

(Increasing Function without Building More Memory)

It is worth noting that the group of people who have been using the Tiny Assembler since the end of 1976 as listed on page 138 of the March 1977 BYTE excludes the author: ME! Well, I have finally gotten my own computer system up and running and have been able to get some use out of the assembler myself. I soon found that there were a few things here and a few things there that could be changed or added that would make the whole assembly process more convenient. I guess there is just no pleasing some people.

The problem is that, as predicted, I have a minimal configuration system. It is a SwTPC 6800 with (at the moment) no extras. The assembler just fits, so there is no place to put patches or additions except at the top of the symbol table. This of course reduces the capacity of the program. It became obvious that the proper approach to further modifications would be to increase the efficiency of the assembler so that additions could be made while maintaining or increasing the symbol table capacity. The first modification, therefore, would have to be the ability to delete symbols when no longer needed so that the symbol table space could be reused. This would allow a smaller table to handle a larger number of symbols.

As suggested in the article "Implementing the Tiny Assembler" (May 1977 BYTE, page 84[2], this has been accomplished by developing a "begin" statement (the assembler mnemonic is BGN). This statement causes the next available location in the symbol table to be pushed into a table stack and a structural level counter (which starts at 1 and keeps track of the nesting level of the BGN statements) to be incremented. Symbols entered beyond this point in the table belong to this BGN block. A label on the BGN statement itself will not belong to the block being defined, but to the group of

[2]Page 16 in this edition

```
LOCN B1 B2 B3

0000          >*************************************************
0000          >*                                               *
0000          >*        EXAMPLE OF AN INLINE BGN BLOCK         *
0000          >*                                               *
0000          >*************************************************
0100          >          ORG  $100
0100          >ROUTINEA RMB  2         SUBROUTINES AVAILABLE
0102          >ROUTINEB RMB  2           IN EXISTING CODE
0104          >ROUTINEC RMB  2
0106          >ROUTINED RMB  2
0108          >ROUTINEE RMB  2
010A          >ROUTINEF RMB  2
010C          >FLAG1    RMB  1         TEST CONDITIONS
010D          >FLAG2    RMB  1           AVAILABLE IN
010E          >FLAG3    RMB  1           EXISTING CODE
0200          >          ORG  $200
0200          >*************************************************
0200          >*                    INLINE CODE                *
0200          >*************************************************
0200          >
0200          >NEWBLOCK BGN              START OF INLINE BLOCK
0200 7D 01 0C >          TST  FLAG1
0203 2D 00    >          BLT  COND1      JUMP TO SUBROUTINE
0205 22 00    >          BHI  COND2        DEPENDING ON TEST
0207 7D 01 0D >          TST  FLAG2        CONDITIONS
020A 23 00    >          BLS  COND3
020C BD 01 00 >          JSR  ROUTINEA
020F 20 00    >          BRA  GO         FIRST USE OF SYMBOL 'GO'
0211 BD 01 02 >COND1     JSR  ROUTINEB
0214 0C
0214 20 00    >          BRA  GO
0216 BD 01 04 >COND2     JSR  ROUTINEC
0206 0F
0219 20 00    >          BRA  GO
021B BD 01 06 >COND3     JSR  ROUTINED
020B 0F
021F BD 01 08 >GO        JSR  ROUTINEE   SYMBOL 'GO' RESOLVED
0215 08
021A 03
0210 0D
0221          >          END             END OF INLINE BLOCK
*** UNRESOLVED:
*** SYMBOLS?
Y
COND1 0211  COND2 0216  COND3 021F  GO   021F

0221
0221 7D 01 0E >CONTINUE TST  FLAG3
0224 22 00    >          BHI  GO         REUSE OF SYMBOL 'GO'
0226 BD 01 0A >          JSR  ROUTINEF
0229 01       >GO        NOP             'GO' RESOLVED AGAIN
0225 03
022A          >*************************************************
022A          >*             INLINE CODE CONTINUES             *
022A          >*************************************************
022A          >
022A          >          END             END OF ASSEMBLY
*** UNRESOLVED:
*** SYMBOLS?
Y
CONE 0221  FLA1 010C  FLA2 010D  FLA3 010E  GO   0229
NEWK 0200  ROUA 0100  ROUB 0102  ROUC 0104  ROUD 0106
ROUE 0108  ROUF 010A

*
```

Listing 1: This simple in line BGN block shows the format of the BGN and END pseudooperations of the Version 3.1 Tiny Assembler. Any symbols defined within the block are deleted by the END statement for the block and can be reused without conflict by subsequent code. Thus the symbol GO at location 021E defined within block NEWBLOCK is not the same location as the later use of GO in the outer level of the hierarchy at location 0229. Note that the new version of Tiny Assembler still condenses symbols into 4 character names in the symbol table by using the first three and the last characters of the symbol as typed. Thus the NEWBLOCK name at location 200 condenses to NEWK in the symbol table.

symbols already in the table. The END statement must then be changed to pull a symbol table location out of the table stack and perform end of block processing on the symbols from this point to the end of the table. Everything relating to these symbols is then cleared from the assembler's tables. The structural level indicator is decremented, and if it becomes 0, end of program is signaled. Using this arrangement, the nesting level of BGN blocks is only limited by the space available for the table stack or the size of the structural level counter (256 in this case).

```
START OF PROGRAM (level A)
:    BGN (level B)
:    :    BGN (level C)
:    :    END
:    :    BGN (level D)
:    :    END
:    END
:    BGN (level E)
:    :    BGN (level F)
:    :    END
:    :    BGN (level G)
:    :    :    BGN (level H)
:    :    :    END
:    :    :    BGN (level I)
:    :    :    :    BGN (level J)
:    :    :    :    END
:    :    :    :    BGN (level K)
:    :    :    :    END
:    :    :    END
:    :    END
:    END
:    BGN (level L)
:    END
END (end of assembly)
```

Listing 2: This is a structured pseudocode representation which shows how the hierarchy of figure 1 would be implemented in a normal coding sequence. The assembly starts with an initial (outer or global is an equivalent term) level so there is one more END statement than the number of BGN statements. This is required to finally terminate the assembly. In this listing, indentation of the code has been used to highlight the various levels of nesting of the blocks from figure 1.

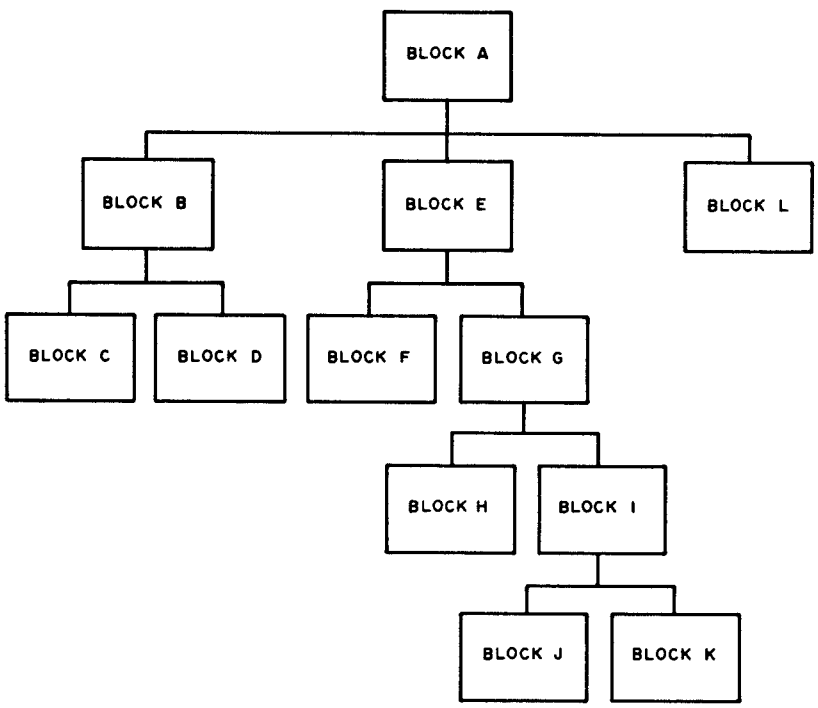


Figure 1: This hierarchy diagram shows the relationships between the functional blocks used in the example of listing 2. Of the 12 blocks in the structure of this program, only a maximum of five are ever active and using up space in the symbol table at any one time. Any entry point referenced at the beginning of block A during the 1 pass assembler's operation is available to any block in the hierarchy even if it is not defined until the end of the assembly.

In practice, the BGN block is used to break a program into individual segments that can each be treated as single functions or routines. Labels and variable names within such blocks are local to the block and are not known outside the block in the rest of the program. This can be most useful when employing structured programming techniques such as those that have appeared from time to time in BYTE articles. A properly structured BGN block should have only one entry point and one exit point. It may be used in either of two different ways.

First, a section of in line code that is only used in one place in a program may be defined as a structural block. Such a block is entered by "falling into" the first statement, and exited by "falling out of" the last statement as program steps are executed in sequential order. There will usually be no label associated with the BGN statement for such a block.

The second possibility is to define a block as a subroutine which can be called from one or more places in the program. Such a block is entered by a jump or branch to subroutine and is exited by a return statement. In this case the entry point (which is usually the BGN statement itself) does require a label. In both cases, once a block has been completed, the internal structure is of no interest to the rest of the program, and any entries in the symbol table and the forward reference table for the current block may be removed when the block is ended.

An example of a small in line BGN block is shown in listing 1. Once this group of branch and jump instructions has been completed, the symbols defined within it may be reused without conflict. A more complex program structure using BGN blocks as subroutines is shown in figure 1. This is similar to the hierarchy diagrams discussed in the first of these articles (see "Designing the Tiny Assembler," April 1977 BYTE, page 60[3], for a discussion of hierarchies and networks). The pseudocode to implement this structure is shown in listing 2. Within any block, references can be made to entries in the symbol table for any already active block (ancestors), the entry point of any block at the same level (siblings), the entry point of any block at the next lower level (direct descendants), and the entry point of any block which is at the same level as currently active ancestor blocks (aunts and uncles). References cannot be made to items which are across any level of siblings and then down another branch of the "family tree" (nieces, nephews and cousins), or to items developed within a lower level of the tree. This is a fairly standard structuring scheme.

[3] Page 7 of this edition

This general arrangement is significantly modified, however, by the 1 pass nature of the Tiny Assembler. In this case, a symbol "belongs to" the first block that references it rather than the block that defines it as a label. Therefore, a symbol that is defined and used at a low level is inaccessible to higher levels as usual, but symbols that are first referenced in a low level block of code and then defined as a label in a later higher level block of code cannot be handled by the assembler. This is because the symbol belongs to the low level block, and both the forward reference table and the symbol table will be cleared of all references to such a symbol at the end of the low level block.

This problem of what level the symbol belongs to can be avoided by requiring such symbols to be first entered into the symbol table by a high level block of code (as a forward reference if need be). In this way the symbol and all references to it will belong to the high level block and will stay in the tables while low level blocks are created and ended. When the symbol is finally used as a label, all unresolved references to it will be resolved, even those which were made in blocks which have been terminated and no longer exist. In general, a symbol must be entered into the symbol table at a level equal to or higher than every reference that is made to it. It is therefore a good idea to define common subroutines early in the program or to make reference to their entry point names in the highest levels of the structure.

To illustrate how these rules are applied, consider what symbols are "known to," or can be referenced by code within block G in the example in figure 1. Any items in A and E that have already been used can be referenced because they are ancestors. Any items within G itself can of course be referenced. This was the scope of the original versions of the Tiny Assembler. The entry points of H and I can be referenced because they are direct descendants. The entry point of B can be referenced because it is an ancestor's sibling that has already been defined. The entry point of F can be referenced because it is a sibling that has already been defined. Any items first used within H, I, J or K cannot be referenced by G because they belong to a lower level. Any information about C and D that was not originally referenced in A is unavailable because they are cousins. The entry point of L (if not yet used) and any items in A and E that have not yet been used cannot be made a forward reference because they will not be defined until after references to them from G have been removed.

When a program is structured as a network instead of a simple hierarchy, things become a bit more complex because of relationships that cross between branches of the structural tree. The same rules apply when determining what can be referenced from what, however, so a program's block structure should be planned so that multiple paths within the network can be contained within a single block to reduce or eliminate forward reference problems.

For programs with a moderate number of symbols or extremely complex forward references, the whole assembly may be considered a single block. In this case, no BGN statement need be used at all. The first END statement that is encountered ends the program much as it does in the original versions of the Tiny Assembler.

It should be pointed out that the entire table of currently active symbols is searched during symbol processing. Therefore, a block may not redefine a symbol used by an earlier but still active block. If this is tried, a duplicate symbol error will occur. This restriction is based on the fact that a 1 pass assembler allowing redefinition of symbols at a local level could not tell the difference between a forward reference to a redefined local symbol and a backward reference to an existing global symbol. Therefore, symbols may only be reused after they have been deleted from the symbol table by an END statement. As it turns out, the hierarchical structuring scheme of the Tiny Assembler is similar to other structural languages such as PL/I, but the restrictions on redefining symbols and the rules determining what level of the structure a symbol belongs to make the structuring of source programs for this assembler unique.

While developing the methods used to handle BGN and END statements it became evident that the deletion and possible reuse of symbols would make it very difficult to produce a complete symbol table dump at the end of the program. In fact, there may never be a complete symbol table during an entire assembly (a situation which requires a 1 pass design by the way). Therefore, each END statement terminates a structural block as if it were a complete program. All unresolved forward references to symbols first used in the current block are listed and the user is given a chance to abort the END statement. If it is aborted, the user will reenter the block, and may continue with corrections and additions as if the END had not been entered. If the user does not elect to abort the END statement, a sorted symbol table listing is provided for the completed block. In this way, every

In this book, the user has the opportunity to compare the source listing of version 3.0 of the Tiny Assembler (Appendix A) with the source listing of version 3.1 (Appendix B). The sections on "Expanding the Tiny Assembler" and the "Tiny Assembler 6800 User's Guide" reflect the changes made to the Tiny Assembler resulting in version 3.1.

occurrence of every symbol and unresolved reference in an assembly is listed without requiring that they ever really exist at the same time.

Does this "virtual symbol table" mean that we have an assembler of unlimited capacity that will run in 4 K? Well, no, not really. While this may be theoretically true, there are some practical limits which are likely to be reached. All programs have a root segment and a list of global symbols which exist throughout the entire assembly. As the program increases in size, the root segment usually grows along with it, but at a slower rate. At some point it becomes impractical to try to fit ever larger programs into the Tiny Assembler. The current design provides for 150 symbols when running in a 4 K machine. This is large enough to assemble simple programs with ease, significant programs through moderate use of structural blocking, and large complex programs if need be. I don't really know what its practical limit would be now, because as noted earlier, I haven't really used the Tiny Assembler enough.

As an additional improvement, the Version 3.1 Tiny Assembler can now expand or contract to the user's requirements. If an 8 K machine were used, the symbol table could be increased to hold over 830 symbols. This could be effectively expanded into the thousands through the use of structural blocking. However, the sequential search times required to find symbols in such a table would no doubt start to become noticeable. Remember, this is supposed to be a TINY assembler, not competition for a full-scale standard assembler.

Having addressed the capacity problem, I then returned to the minor changes that started this whole modification project in the first place. Several items which did not work as one might expect in the previous version have been changed to process in a more normal manner, and a larger subset of the Motorola 6800 assembly language definition is now supported. The following is a list of some of the major changes which have been made. They are illustrated (where possible) in listing 3.

- The FCC pseudooperation will allow the use of character string delimiters as well as the previous string length operand.
- The FCB statement will allow the use of literals following an apostrophe, and the FCB and FDB statements may now make forward references.
- Delimiters such as the comma (,), space ( ), plus (+) and minus (–) may be used as literals after an apostrophe. The apostrophe itself may be used as such a literal.
- If a label is used with an ORG statement, its value (as well as the program counter) is set to the value of the operand.
- Execution of the assembler after an assembly has been completed invokes the cold start rather than the warm

```
G

LOCN B1 B2 B3

0000            >*************************************************************
0000            >*                                                           *
0000            >*   EXAMPLES OF THE EXTENDED TINY ASSEMBLER FUNCTIONS       *
0000            >*                                                           *
0000            >*************************************************************
0000            >
0000 41 42 43 >           FCC  5,ABCDE   STRING LENGTH = 5
0003 44 45
0005 41 42 43 >           FCC  /ABCDE/   STRING DELIMITERS CAN BE ANY
0008 44 45
000A 41 42 43 >           FCC  #ABCDE#     NON NUMERIC VALID CHARACTER
000D 44 45
000F            >
000F 31 32 00 >           FCB  '1,'2,&&  USE OF APOSTROPHE IN FCB AND
0012 00 00      >           FDB  XX          FORWARD REFERENCE IN FCB & FDB
0014            >
0003            >&&       EQU  3         RESOLVE FORWARD REFERENCES IN
0011 03
1FFF            >XX       EQU  $1FFF       PREVIOUS FCB & FDB
0012 1F FF
0014            >
0014 2C 27 20 >           FCB  ',,'',' ,'+
0017 2B
0018            >*                       NOTE USE OF COMMA AS A LITERAL
0018            >*                         AND AS A DELIMITER ABOVE
0018            >
0100            >CONTINUE ORG  $100      FULL FUNCTION ORG STATEMENT
0100 01 00      >         FDB  CONTINUE
0102            >*PAGE


LOCN B1 B2 B3

0102            >*************************************************************
0102            >*          EXAMPLE OF FORWARD REFERENCE HANDLING            *
0102            >*************************************************************
0102            >
0102            >*#IFEF   BGN            START NEW STRUCTURAL BLOCK
0102 B6 00 00 >           LDAA AAAA      ESTABLISH FORWARD REFERENCES
0105 B6 00 00 >           LDAA BBBB
0108            >
0108            >NXTLEVEL BGN            START ANOTHER STRUCTURAL BLOCK
0108 B6 00 00 >           LDAA AAAA      REFERENCE ALREADY ESTABLISHED
010B B6 00 00 >           LDAA BBBB      BBBB ALSO ESTABLISHED
010E B6 00 00 >           LDAA CCCC      REFERENCE TO NEW ITEM
0111            >         END            END OF NXTLEVEL, CCCC IS UNDEFINED
*** UNRESOLVED:
CCCC 010F
*** SYMBOLS?
Y
CCCC 0000

0111
00FF            >AAAA     EQU  $FF       RESOLVE SYMBOL AAAA
0103 00 FF
0109 00 FF
0111            >         END            END #IFEF
*** UNRESOLVED:
BBBB 0106
BBBB 010C
*** SYMBOLS?
N

0111
0001            >BBBB     EQU  1         ADOPT END AND DEFINE BBBB
0106 00 01
010C 00 01
0111            >         END            END #IFEF AGAIN
*** UNRESOLVED:
*** SYMBOLS?
Y
AAAA 00FF  BBBB 0001  NXTL 0108

0111
0111            >         END            END OF ASSEMBLY
*** UNRESOLVED:
*** SYMBOLS?
Y
CONE 0100  #IFF 0102  &&   0003  XX   1FFF

*
```

Listing 3: In addition to the reorganization of symbol table management and introduction of the BGN pseudooperation, a number of minor incremental improvements were added to Version 3.1 of the assembler. These are illustrated here, and are described in more detail in the text of the article.

start entry point so that different programs being assembled at the same time are completely separated. Within a given assembly, however, restarts are still from a warm start entry point. A warm start uses the old symbol table contents, and a cold start clears the entire symbol table before entry.

- The Tiny Assembler can now be stored in its entirety in ROM or PROM, and it is more easily relocatable.
- The forward reference and symbol tables can be redefined at a location and size determined by the user.
- A comment line starting with *P will produce a page break and a new heading (eg: *PAGE).

While modifying the assembler, I found the need for a simple, quiet (on a Teletype) loader to try out various program changes. As mentioned in previous articles, there are two types of loaders for the Tiny Assembler. Those handling complex forward references must first zero memory and then add each byte of generated code to the current memory value (a single memory location may have several components added to it). Those handling corrections made by the user when the assembler is used interactively must simply replace whatever is currently in memory with the generated code. Since I tend to have many more errors to correct than complex references to handle, the loader shown in listing 4 was developed to complement the one listed in the *Tiny Assembler User's Guide*. If the assembler's stack is relocated from A07F to A042 (which is easy to do: simply change locations 06D3, 06D4 of Version 3.1), the assembler and the loader can remain resident in a 4 K machine at the same time. The code:

```
LDAA    #$3C
STAA    $8007
```

will suppress the echo on input for any program using the standard Motorola MIKBUG input routines. In this case it allows the Teletype to read the Tiny Assembler load tape without having the print mechanism chattering away.

The current version (3.1) of the Tiny Assembler is quite a powerful assembly program which will operate within very tight memory restrictions. In larger machines this may mean more space can be made available for an input source buffer, monitor, editor, assembled object modules or other memory resident programs. In small machines it may mean the difference between being able to assemble anything at all or not.

I should point out that the development of the Tiny Assembler was not a 1 person project, but rather a collective madness that infected quite a few individuals. Several sections of Version 3.1 reflect the ideas and work of Al Losoff, Chuck Bram and George Kuss. As the assembler evolves, I would like to hear from others who have found new and wonderful ways to improve, modify or adapt the program to different environments and requirements.

Listing 4: A "quiet" loader is one which demurely purrs as it loads in from the reader of a Teletype, ignoring the noisy printer. This is a loader capable of residing in the Motorola MIKBUG program's scratch pad and loading object code from the Tiny Assembler. Note that there is no end of tape character, nor facilities for rereading the loader tape for verification, nor error handling outside of that provided by the Motorola MIKBUG monitor's routines.

```
*G

LOCN B1 B2 B3

0000            >*****************************************
0000            >*                                       *
0000            >*  'QUIET' LOADER FOR THE TINY ASSEMBLER  *
0000            >*                                       *
0000            >*****************************************
0000            >
E055            >HEXIN    EQU  $E055    MIKBUG HEX BYTE IN
E047            >ADRIN    EQU  $E047    MIKBUG ADDRESS IN
E1AC            >CHARIN   EQU  $E1AC    MIKBUG CHARACTER IN
E1D1            >CHAROUT  EQU  $E1D1    MIKBUG CHARACTER OUT
0001            >START    EQU  1        START LOADING CHAR
0001            >STOP     EQU  1        STOP LOADING CHAR
0000            >
A04A            >         ORG  $A04A
A04A 00 00      >         FDB  LOADER   LOAD STARTING ADDR
A04A 86 3C      >LOADER   LDAA #$3C     SUPPRESS ECHO FOR
A04B A0 4A
A04C B7 80 07 >           STAA $8007      INPUT
A04F 86 3E      >         LDAA #'>      PRINT 'READY MARK'
A051 BD E1 D1 >           JSR  CHAROUT
A054 BD E1 AC >SEARCH     JSR  CHARIN   SEARCH FOR START
A057 81 01      >         CMPA #START     OF BLOCK
A059 26 F9      >         BNE  SEARCH
A05B BD E0 47 >           JSR  ADRIN    GET LOADING ADDRESS
A05E BD E1 AC >LOOP       JSR  CHARIN   READ NEXT CHARACTER
A061 81 01      >         CMPA #STOP    IF STOP CODE THEN
A063 27 EF      >         BEQ  SEARCH     SEARCH FOR NXT BLOCK
A065 BD E0 55 >           JSR  HEXIN    ELSE READ HEX BYTE
A068 A7 00      >         STAA X          STORE IT
A06A 08         >         INX             INCREMENT ADDRESS
A06B 20 F1      >         BRA  LOOP       AND CONTINUE
A06D            >
A06D            >         END           END OF ASSEMBLY
*** UNRESOLVED:
*** SYMBOLS?
Y
ADRN E047  CHAN E1AC  CHAT E1D1  HEXN E055  LOAD A04A
LOOP A05E  SEAR A054  STAT 0001  STOP 0001
```

Tiny Assembler 6800 User's Guide

User's Guide Contents

This User's Guide contains the following major sections:

Package Contents
System Definition
Language Specifications
- Character Set
- Source Statements
- Labels
- Operator Field
- Operand Field
- Addressing Modes

Error Messages
Operating Instructions
Modification Notes

Package Contents

To have sufficient material to operate this assembler on an M6800 machine with a MIKBUG™ monitor; the user needs:

- The object tape containing the assembler load module in the MIKBUG PUNCH/LOAD format or bar code reading equipment.
- This user's guide to help the user operate and understand the program.

If modifications or (perish the thought) corrections are to be made to the program, the combined listing showing both source code and generated machine code will also be necessary. This listing was made on a one pass cross assembler and differs in some respects from the listing produced by the Tiny Assembler.

System Definition

The Tiny Assembler differs in several respects from other M6800 assemblers. It will run in a machine with only 4K (4096) memory locations and a MIKBUG monitor. It processes source code in one pass and therefore can be used interactively. It requires only the system control console IO interface and uses MIKBUG routines to handle all IO operations. The easiest way to use this program (especially to become familiar with it) is to read this user's guide, load and execute the program, and start entering assembly instructions.

This documentation is for version 3.1 of the M6800 Tiny Assembler and should not be applied to other versions since functional specifications are likely to change for each new version. Furthermore, user modifications may make one copy of this version different from another after a while. Even though the program has been well tested, it is still covered by the laws of programming which state (in part) that there is no such thing as a bug-free program. If anyone finds that the assembler does not perform as indicated in this document, the author would like to know about it so that it can be corrected in future versions of the program and/or User's Guide.

This version was specifically developed for interactive use, and to interface with an Automatic Send/Receive Teletype with tape punch and tape reader control characters activated. Other types of terminals may require that unique IO modules be written to handle tape files (see the section on modification notes).

Language Specifications

In order to keep this document as concise as possible, the language specifications given here will consist of differences between the source language used by this assembler and that defined in the Motorola publication *M6800 Microprocessor Programming Manual*, rather than a complete language specification. If the above publication is unavailable, the source listing for this program will provide examples of valid code for this assembler. There is also a sample listing shown at the end of this User's Guide.

Character Set

Any ASCII encoded character with a hex value of 20 (space) through 5F (_) will be recognized by the assembler. These are referred to in this guide as 'valid characters'. Whatever character is defined as the end of line character at location HEX 067C (the default is a Carriage Return) is recognized as an end of input line character. All other characters are ignored by the assembler and can be used for IO equipment control or other user requirements. The following are

™**Note:** MIKBUG is a trademark of Motorola, Inc.

NOT special characters for this assembler as they are in the Motorola assembler:

@ (commercial at)
% (percent)
A (letter A)
B (letter B)
H (letter H)
O (letter O)
Q (letter Q)
HORIZONTAL TAB
/ (slash)

HORIZONTAL TAB may not be substituted for a space.

Delimiter characters cannot be used within a symbol. The delimiter characters are:

+ (plus)
– (minus)
, (comma)
SPACE
END OF LINE CHARACTER

Source Statements

The Tiny Assembler can recognize and process the first fifty-six (56) and the last one (1) characters in each input line. If comments extend beyond the fifty-sixth character they will be printed on the listing, and are therefore usable. The last character must be an end of line character. Characters that are not within the range of the valid character set are ignored.

Source code must start in column one. If line numbering is used the numbers must be located in the comment field, or the buffer loading routine must be changed to start at some position past the start of the input buffer (see modification notes below). Any source statement may have comments after the last required operand. The fields within a source statement are the same as those used by the Motorola assembler with one exception. Where an operand is required to indicate the accumulator being used, the 'A' and 'B' must be listed as the fourth character of the mnemonic instruction rather than as a separate source field. For example, the instruction

```
LDA A SYMBOL
```

must be written

```
LDAA SYMBOL
```

Blank lines may be entered at any time and do not affect the output code. A blank line may have a label and the label will be processed. Since the input line is buffered, any amount of backspacing can be done within the first 56 characters before the line is processed by the assembler. The default backspace character is '!' (exclamation point).

Labels

Labels may be of any length, but the value entered into the symbol table will consist of the first three (3) characters and the last one (1) character. This symbol compaction is also applied to all operand items, including decimal numbers. The following examples show how this is done.

ENTERED LABEL	STORED VALUE
W	W
WX	WX
ABD	ABD
ABCDEFGHI	ABCI
LABEL01	LAB1
LABEL21	LAB1

A label must start with a hex value greater than or equal to 41 (letter A). The rest of the characters in the label may be any valid character. It will be terminated when a delimiter character is found. A label must not consist of the letter X alone as this is used to indicate the location of the index register. It can be the letter A or B alone, however, because these are not special characters in this assembler. A label may be used with any statement or used alone without any other statement fields to simply mark the current location in a source program. This has the same effect as coding:

```
LABEL    EQU    *
```

Where '*' stands for the current value of the program counter. A label should not be defined at or equated to zero as this value is used to flag unresolved symbols (see below) in the assembler. The numeric constant zero (0) should be used instead.

Operator Field

The valid operators for this assembler include the 72 executable instructions of the M6800 microprocessor and the assembler directives ORG, EQU, FCB, FDB, RMB, FCC, BGN, and END.

Executable operators are the same as in the Motorola assembler and are used in the same way except when an accumulator operand is required. These operands must be entered as the last character of the mnemonic instruction as mentioned above.

It is possible to include delimiter characters in the FCC string (even a carriage return will work), but the listing may be affected by control characters and it is not a recommended practice.

The FCB and FDB assembler directives can contain any number of operands sepa-

rated by commas. Null items (a leading comma or multiple commas) are equivalent to zero.

The assembler directives SPC, NAM, OPT, MON, and PAGE are not supported. However, a page function can be invoked by entering a comment line starting with '*P' (eg. *PAGE).

The ORG assembler directive checks to see if the statement had a label by comparing the value of the last label encountered with the current value of the program counter. Therefore, if you code:

```
LABEL-1   EQU   *
          ORG   10
```

the value of LABEL-1 will be set to 10 as if it were a label on the ORG statement. This rarely happens, but it is handy to know about.

The BGN assembler directive starts a new structural block within the source program. Labels used for the first time beyond this point will be removed from the symbol table and the forward reference table when a matching END statement is encountered. Any symbol already in use may be referenced within a BGN block. (Note: After being removed by an END statement, a symbol is no longer 'in use.') Any symbol listed in the symbol table dump from an END statement for a BGN block can be reused after the END statement is finished. Any forward reference listed by the END statement for a BGN block must be resolved by aborting the END (not answering 'Y' to the 'SYMBOLS?' prompt) and defining a value for the symbol. Otherwise the assembler will *not* be able to resolve such a forward reference. Each BGN statement must have a matching END statement. Aborted END statements are ignored. BGN blocks may be nested to a level of 255. The final END statement terminates the assembly. The END processing for each block is handled as if each block were a separate program. Through the reuse of symbols and symbol table space, there is almost no limit to the size of the source program that can be assembled by version 3.1 of the Tiny Assembler.

Operand Field

An operand may consist of a number, a symbol, or an expression combining numbers and/or symbols.

Numbers may be hexadecimal or decimal. Octal and binary are not supported. Hexadecimal numbers MUST be prefixed with a '$' (dollar sign). Decimal numbers MAY be prefixed with an '&' (ampersand). When no prefix is used, a decimal number is assumed. A number alone will be converted to a direct or extended address depending on whether it is less than 256 or not. Leading zeroes need not be entered for hex or decimal numbers. No operand item may exceed four (4) characters in internal storage (see LABELS above for the compression algorithm), so the largest decimal number that can be entered at one time is 9999. The largest number that the assembler can handle is decimal 65535 (hexadecimal FFFF). Byte overflow and invalid address are not detected. Adding one to hex 'FFFF' will result in hex '0000' with no error.

Symbols are defined by the same rules that labels are. The special symbol '*' (asterisk) when used as an operand is converted to the value of the program counter at the beginning of the instruction containing the asterisk. A null or missing operand will default to zero. A symbol may be used in an operand before it is used in a label. This is an 'unresolved symbol' and will be corrected by the assembler when a value is assigned to the label containing the symbol. A total of twenty (20) references to unresolved symbols can be pending at any one time. At this point, an error for forward reference table full is printed (ERROR H). This 'error' is really a warning, and processing can continue. If another reference is made to an unresolved symbol before any existing unresolved symbols are given a value, however, the table will overflow causing a severe error. Any number of forward references can be made as long as the total number at any one time is never greater than 20. All references to unresolved symbols that are still pending at the end of each structural block are shown on the listing. The total number of symbols that can be defined at one time during an assembly is 150.

Expressions are processed the same as in the Motorola assembler except that multiplication and division are not supported, and unresolved forward references that are preceded by a minus sign will be combined with (added to) the rest of the expression value instead of subtracted from it. The one pass design cannot always resolve indirect evaluation of symbols where one unresolved symbol is defined in terms of another unresolved symbol. It is best to avoid such constructions.

Addressing Modes

Addressing modes are determined as in the Motorola assembler. When an operand is evaluated as less than 255 and the addressing mode may be either direct or extended, direct is always used to conserve space. For this reason the generated code is not relocatable. For an unresolved forward reference

where the addressing mode may be either direct or extended, extended is used since it is unknown whether or not the resolved value will be less than 256.

The sample listing included in "Implementing the Tiny Assembler" provides examples of both valid and invalid code with error messages shown where applicable. This listing has been developed solely for the purpose of demonstrating features of the assembler and does not represent a usable program.

Error Messages

Error conditions are shown on the listing as a standard error message followed by a one position alphabetic error code indicating the specific error. All errors shown are severe error conditions except ERROR H, which indicates that the forward reference table is full (twenty entries) and is only a warning. Processing is not affected by this condition and the user may continue with the next line of code (see below).

For all severe errors, processing of the current line of input is suspended and the program counter is returned to the value it had prior to the line in error. In most cases, therefore, corrections can be entered and processing can continue with the next line of input. There are three error conditions that can occur while processing a label. These are ERROR E (symbol table overflow), ERROR F (duplicate label), and ERROR K (invalid symbol). In these cases, if the symbol table is not full and the error was in fact in the label, the correct label must be entered with the correcting line of code. For all other severe errors, the label will have been processed before the error condition was raised and must not be reentered. If it is, an ERROR F (duplicate label) will be raised. The assembler keeps track of the last label processed, so reentering the label is not necessary. Even an EQU statement can be corrected by reentering all but the label as long as the label was processed before the error occurred (see sample listing). A complete listing of error codes and meanings follows.

ERROR A: Severe error, any label will have been processed. The mnemonic operator did not start with an alphabetic character (A through Z). Reenter the line of code with the correct mnemonic operator. Check to see if blanks were omitted before the mnemonic which may cause it to be treated as a label and the operand to be treated as the mnemonic operator.

ERROR B: Severe error, any label will have been processed. The mnemonic operator was not found in the table of allowable mnemonics. Reenter the line of code with the correct mnemonic operator. As above, check to see that the operator was not treated as a label. Also check to see that accumulator operands 'A' and 'B' were included where required and excluded where not acceptable, and that they were entered as the fourth letter of the mnemonic rather than separated from it by a blank.

ERROR C: Severe error, any label will have been processed. The instruction used does not allow immediate addressing. Either the instruction used or the mode of addressing must be changed. Reenter the corrected line of code. Check to see that a '#' was not entered instead of a '$' for a hexadecimal number.

ERROR D: Severe error, any label will have been processed. The indicator for addressing mode is an invalid character. An 'X' character is used for indexed addressing. Check to see if some other character has been used instead.

ERROR E: Severe error, the lable of the statement being flagged will not be processed, except in the case of a BGN statement where it may or may not be processed. The symbol table overflowed. Either a symbol or a BGN block being added would not fit. Symbols fill the table from the bottom up, while the BGN block stack fills the table from the top down. If the current BGN block is ended, one address will be removed from the BGN stack, and all of the symbols first used on the current BGN block will be removed from the table, allowing processing to continue. With proper management of a program's BGN blocks, this error should be avoidable.

ERROR F: Severe error, the label will NOT have been processed. The symbol being used as a label has already been used as a label. Reenter the line of code with a new label. If forward references have been made to the symbol in error, they may have been given an erroneous value from the previous entry in the table. Symbols must be unique by the first three characters and the last one character (see LABELS above).

ERROR G: Severe error, any label will have been processed. A relative displacement has been calculated which is more than −128 or +127 bytes from the location of the next instruction. This is beyond the

allowable range for branch instructions. The source program may be rewritten to shorten the distance to be branched, or a short branch to a jump instruction may be used.

ERROR H: Warning message, the entire line of output will have been processed. The forward reference table is now full. If any more forward references are made before any current references are resolved, the table will overflow (see below). The problem can be reduced in general if commonly used data items, literals, character strings, and subroutines are located at the start of the program. The maximum number of forward references allowable at any one time is twenty (20).

ERROR I: Severe error, any label will have been processed. The forward reference table overflowed. A reference in the current line of code was not entered into the forward reference table. If the number of entries in the table is reduced, the reference in error can then be entered into an empty slot. Because labels are required to reduce the number of entries in the table, a label on the line of code in error will be lost to the correction process, and must be reentered using a new symbol since the last one is already in the symbol table.

ERROR J: Severe error, any label will have been processed. A hexadecimal number contains a character with a hexadecimal value less than 30 (zero) or greater than 46 (F), or a decimal number contains a character with a hexadecimal value that does not start with three (3x). This will not catch characters from 3A (:) through 3F (?), but will catch any unwanted alphabetic characters. In either case, reenter the correct line of code.

ERROR K: Severe error, any label will have been processed if it is not the symbol in error. A symbol has been used where the first character has a hexadecimal value of less than 41 (A). Reenter the corrected line of code. Check to see that a missing operand has not caused comment symbols at the end of the line to have been picked up as the operand.

Operating Instructions

To operate the assembler, the load tape can be loaded into any system with a MIKBUG monitor just as it is. For systems without such a monitor, a loader will have to be used that can use the MIKBUG format. The logic for such a loader should include the following:

- Search for the character string 'S1'.
- Read in a single byte containing the length of the next segment of input including start address and check byte in bytes.
- Read in the address where the next segment of input is to start.
- Read the required number of bytes and load them starting at the required address.
- At the end of each line there is a one byte check sum which can be ignored by a minimal loader. The check sum is the complement of the sum of all characters after the 'S1'.
- Search for the next string of 'S1', and so on.

NOTE: All input is in hexadecimal format.

For individuals using optical bar code readers an appropriate loader must be used to recover the absolute object code in "Implementing the Tiny Assembler" and Appendix. The assembler can also be loaded by hand using the object code in the assembly listing printed here. This process takes roughly one to three evenings of work at a terminal. The assembler must be loaded into the first 4096 locations of memory. The initial starting position for execution of the assembler is HEX 067D. This is automatically set by the load tape. Upon beginning execution, the assembler will execute any user written routines which have been loaded into the symbol table area which is initially set to be from location HEX 010D through 0490. These routines may be used to name programs, set desired control characters where interactive overriding of the assembler defaults may be desired, or any other user defined requirement. One possibility is to dump to the output tape a copy of the Tiny Assembler loader in a format acceptable to the system's load format. Then, when the assembler is finished with the output tape containing the loader and object code, the tape can be read directly into the system without a separate operation to load a loader from some other source.

Upon completing the user written routines (if there are any), a return from subroutine (RTS) is loaded in front of the symbol table to prevent further attempts to access these routines. The symbol table and forward reference table are then cleared completely, destroying any user written routines.

After initializing several variables, the monitor's start execution pointer (HEX A048, A049) is changed from 067D to the address of a secondary entry point (HEX

06C2). In this way the assembler can be restarted during a run without changing the symbol table, forward reference table, program counter, or other program status variables. The stack is then positioned at location HEX A07F, the top of the MIKBUG RAM. If a total restart is desired, the assembler can be again restarted from location HEX 067D, but this time there will be no attempt to execute the user defined routines that were destroyed on the first run. The symbol table, forward reference table, and program status variables will be cleared again, however. Between complete assemblies, the cold start address (HEX 067D) is automatically invoked to clear all of the tables of the previous program.

For each END statement, the assembler prints the symbols that were first used in the block being ended. After any unresolved forward references are listed, the user is prompted with 'SYMBOLS?'. If there are no unresolved items, or the user does not want to resolve those that are listed, a response of 'Y' (yes) will continue the END processing with a symbol table dump. Any other response will abort the END statement as if it had never been issued. After the last block is ended, control is returned to the monitor with a Tiny Assembler restart address left in the monitor's restart pointer (HEX A048, A049).

The only output from this assembler is the combined listing showing the generated object code for each line of source code. The program was developed to run on an Automatic Send/Receive Teletype with all tape punch and tape reader control characters enabled. The tape punch is used to capture the generated object code in a machine readable form. The assembler starts each line of the listing with a tape punch on character (HEX 12) and a loader recognition character (HEX 01) followed by a pause to allow a magnetic tape (should that be used rather than paper tape) to come up to speed. A pause loop counter value of 30,000 (HEX 7530) causes about a half second pause. The counter may vary from 1 to 65535 (HEX 0001 to FFFF). If a magnetic tape is not being created, the pause loop may be replaced by NOP instructions (HEX 01) to speed up assembly time (see modification notes below). The last byte of object code on each line is followed by a loader stop character (HEX 01) and a tape punch off character (HEX 14). Therefore, only the object code gets punched on the output tape. This tape can then be loaded into memory as an executable program using the loader shown in figure 1. The tape on and tape off characters can be removed. This causes the entire listing to be recorded on tape, but does not cause lots of tape starts and stops. The loader will weed out the parts of the listing that it does not need.

The only input to this assembler is through the control console. For interactive use, a line of input is simply typed in after each prompt character (>). To load from a tape file, the prompt character must be changed to a start tape reader character (13, see notes below for modifications). Each line of input should then be terminated with a stop tape reader character (14), an end of line character (in that order for paper tape), and one or more characters outside the assembler's valid character range such as a null (00) or rubout (FF).

For an absolutely minimal system, source code can be entered through the console, and the object code can be copied from the listing by hand.

Modification Notes

The basic philosophy behind the development of this assembler is that hardware costs should be kept as low as possible. In maintaining this philosophy, it makes more sense to modify the program to fit the hardware than to modify or add to the hardware to accommodate the program. This section has therefore been included to point out areas where modifications may be desired.

As mentioned above, any user written initialization routines may be inserted into the symbol table area. This is not actually a modification, but rather an available assembler function.

The most severe modification that may be commonly required is to separate input and output operations. The assembler was written to use the MIKBUG monitor to handle all IO operations. This monitor cannot handle concurrent input and output operations. It therefore requires that all input operations stop while output is being written to the listing. If a separate input routine is written to allow concurrent interrupt driven input, it will replace the CHARIN subroutine shown in the linkage table at location HEX 0790. This routine loads one input character into the A accumulator and is the only input routine in the program. Several of the monitor's output routines are used.

- CHROUT writes out the contents of the A accumulator as an ASCII character.
- HEXOUT writes out the high nibble of the A accumulator as a hex number and then writes out the low nibble of the A accumulator as a hex number. This is currently implemented as two

calls to separate MIKBUG routines.

- BLKOUT writes out a blank character.
- STROUT writes out ASCII coded characters from the current location of the index register until an EOT character (HEX 04) is encountered.

Any of these routines may be replaced by user written routines. In all cases, only the linkage table needs to be changed to point to the location on a new module.

Several special characters or data items in the program may be changed to reflect the personal preferences of the user. The following table will help locate where changes to some of these items can be made. All addresses are in hex.

End of line character	067C
Space available for user routines	010D to 0490
Back space character	070D
Loader 'start object code' flag	0802
Loader 'stop object code' flag	081C & 0845
Start punch character	064B
Stop punch character	0821 & 084A
Prompting character	0705
Tape delay start loop	07F9 to 0800
Delay loop counter	07FA, 07FB
Stack location pointer	06D3, 06D4
Start of forward reference table pointer	06A5, 06A6
Entry point for user routines	0686, 0687
RTS to replace with NOP for user routines	0684
Start of symbol table pointer	06B4, 06B5
Stop byte for symbol table pointer	06AD, 06AE
Character-out subroutine pointer	0495, 0496
Character-in subroutine pointer	0498, 0499
Hexout subroutine pointer	049A to 04A1
Blank-out subroutine pointer	04A3, 04A4
String-out subroutine pointer	04A6, 04A7
Address jumped to at end of assembly	04A9, 04AA
Start of input buffer pointer	06C8, 06C9 0702, 0703
Stop backspace processing pointer	0711, 0712
End of input buffer pointer	0721, 0722
First character recognized within buffer pointer	074B, 074C

The output routine issues separate carriage returns and line feeds. If a terminal is used which combines line feeds with carriage returns, some modification of this routine may be necessary.

End of execution occurs at hexadecimal location 06F0. At this point the assembler returns to the MIKBUG mainline routine (HEX E0E3) through a jump instruction. If no monitor is being used or a different termination procedure is desired, this jump can be changed.

It is possible to relocate and/or change the size of the forward reference table and the symbol table. The two must be kept together, however, because the beginning of the symbol table defines the end of the forward reference table. The forward reference table must be an even multiple of five (5) bytes. There must be a single byte before the first byte of the symbol table to be used as the entry point for any user supplied initializing routines. The symbol table must be an even multiple of six (6) bytes. It must be followed by a single table stop byte and a two byte address space for use in the BGN stack. After this structure has been developed, the following items must be modified in the Tiny Assembler code (see the above table for addresses): start of the unresolved symbol table, entry point for user routines, start of symbol table, and the symbol table stop byte.

If user subroutines are put into the symbol table, they must start in the byte after the entry point. The RTS (HEX 39) at location HEX 0684 must be changed to a NOP (HEX 01) or the assembler will never get to the user's routines.

The input buffer can be relocated and its size changed by modifying the following locations (see the above table for addresses): start of buffer location (in two places), stop backspace processing location (usually the same as the start of the buffer), end of buffer, and first character recognized location (again usually the same as the start of the buffer). If line numbers are desired at the start of the input lines, set the first character recognized location to a point beyond the start of the buffer and all characters between the two addresses will be loaded but ignored.

```
                        *****************************************************************
                        *                                                               *
                        *                    M 6 8 0 0   A S S E M B L E R              *
                        *                                                               *
                        *                    J. F. EMMERICHS  -  DEC. 1976              *
                        *                            VERSION 3                          *
                        *                                                               *
                        *                                                               *
                        *                 COPYRIGHT JOHN F. EMMERICHS, 1976             *
                        *                                                               *
                        *        ANY ORGANIZATION OR INDIVIDUAL INTERESTED IN REPRODUCING*
                        *        THIS PROGRAM IN WHOLE OR PART OR IN USING IDEAS TAKEN  *
                        *        FROM THIS PROGRAM IS FREE TO DO SO.  HOWEVER, ALL SUCH *
                        *        ORGANIZATIONS AND INDIVIDUALS ARE REQUESTED TO REPRODUCE*
                        *        THIS COPYRIGHT MESSAGE AS PART OF THE INTRODUCTION TO A *
                        *        WRITTEN WORK, OR AT THE HEAD OF PROGRAM LISTINGS.      *
                        *                                                               *
                        *****************************************************************
                        *                                                               *
                        *   ERROR CODE MEANINGS                                         *
                        *                                                               *
                        *       A = VERB DOES NOT START WITH AN ALPHA CHARACTER          *
                        *       B = VERB NOT IN TABLE                                   *
                        *       C = VERB DOES NOT ALLOW IMMEDIATE (#) ADDRESSING        *
                        *       D = INVALID VERB MODIFIER                               *
                        *       E = SYMBOL TABLE OVERFLOW                               *
                        *       F = LABEL ALREADY IN USE                                *
                        *       G = RELATIVE BRANCH > 128 AWAY                          *
                        *       H = FWD JUMP TABLE FULL *.*** NOT AN ERROR BUT A WARNING ***
                        *       I = FWD JUMP TABLE OVERFLOW                             *
                        *       J = INVALID CHARACTER IN HEX OR DECIMAL NUMBER          *
                        *       K = INVALID SYMBOL                                      *
                        *                                                               *
                        *****************************************************************
                        *                                                               *
                        *   VERB CLASS MEANINGS                                         *
                        *                                                               *
                        *       0 = PSH AND PUL.  BOTH ARE CHANGED TO CLASS 5 FOR PROCESSING
                        *       1 = OP CODES WHERE IMMEDIATE ADDRESSING IS VALID (EG. LDA)
                        *       2 = OP CODES WHERE DIRECT ADDRESSING IS VALID BUT       *
                        *           IMMEDIATE IS NOT (EG. STA)                          *
                        *       3 = OP CODES WHICH CAN ADDRESS MEMORY (NOT IMMED OR DIRECT)
                        *           OR AN ACCUMULATOR ALONE.  IF AN ACCUMULATOR IS BEING
                        *           USED, CLASS IS CHANGED TO A 5 FOR PROCESSING (EG. CLR)
                        *       4 = OP CODES WHICH USE RELATIVE ADDRESSING (EG. BRA)    *
                        *       5 = OP CODES WHICH USE INHERENT ADDRESSING (EG. INX)    *
                        *       E = END STATEMENT (END)                                 *
                        *       F = ASSEMBLER DIRECTIVE (EG. ORG)                       *
                        *                                                               *
                        *   FLAGS (LOW NIBBLE OF CLASS) ARE FOR 16 BIT REGISTER OP CODES*
                        *                                                               *
                        *   OP CODE IN TABLE IS ACTUAL CODE USED FOR CLASSES 0, 4, 5, E, AND
                        *      F.  FOR 1, 2, AND 3, IT IS THE IMMEDIATE CODE (EVEN IF NOT
                        *      A VALID ADDRESSING MODE FOR A GIVEN INSTRUCTION)         *
                        *                                                               *
                        *****************************************************************
                        **SKIP                                                          *
                        *****************************************************************
                        *                                                               *
                        *                    S T A R T   O F   L I S T I N G            *
                        *                                                               *
                        *****************************************************************
(0000)                  |       ORG  0000
(0000) 7E  00  00       |       JMP  MAIN       PGM ENTRY POINT IS AT 0000
(0003)                  | ERORSW RMB  1         ERROR SWITCH
(0004) 20               | MOD    FCB  $20                              ITEMS       *
(0005)                  | NAME   RMB  4                                BLANKED FOR *
(0009) 20               | SIGN   FCB  $20                              EACH        *
(000A) 20  20           | BLNK   FCB  $20,$20                          COMMAND     *
(000C) 00  00           | AD01   FCB  $00,      GENERAL ADDRESS VARIABLE
(000E)                  | SAVADR RMB  2         SAVE ADDRESS
(0010)                  | ALPHA  RMB  1         ALPHA OPERAND SWITCH
(0011)                  | ANSW   RMB  1         STRING COMPARE ANSWER
(0012) 2A  20  20       | ASTK   FCC  4,*       FOR COMPARE WITH *
(0015) 20               |
(0016)                  | BYTOUT RMB  1         BYTE OUT
(0017) 0D               | CC     FCB  $0D       CURRENT CHARACTER
(0018)                  | CLASS  RMB  1         VERB CLASS
(0019) 04               | CMPLEN FCB  $04       COMPARE LENGTH
(001A)                  | COUNT  RMB  1         GENERAL COUNTER - 1
(001B) 0D               | CR     FCB  $0D       HEX VALUE FOR A CARRIAGE RETURN
(001C) 00               | EOJFL  FCB  $00       END OF JOB FLAG
(001D)                  | FLAGS  RMB  1         VERB FLAGS
(001E) 00               | FWD    FCB  $00       RESOLVE ADDRESS FLAG
(001F)                  | HEX    RMB  1         HEXADECIMAL OPERAND SWITCH
(0020)                  | IXS1   RMB  2         IX-SAVE-1
(0022)                  | IXS2   RMB  2         IX-SAVE-2
(0024)                  | IXS3   RMB  2         IX-SAVE-3
(0026)                  | LAB_SW RMB  1         LABEL-SWITCH
(0027)                  | LSTLAB RMB  2         ADDRESS OF LAST LABEL ENCOUNTERED
(0029)                  | LTRL   RMB  1         LETTER LENGTH
(002A)                  | NXCHAR RMB  2         NEXT CHARACTER TO BE USED FROM BUFFER
(002C) 00  00           | NULL   FCB  $00,      ZERO VALUE CONSTANT
(002E)                  | OPCODE RMB  1         OP CODE
(002F) 00  00           | PGMCNT FCB  $00,      PROGRAM COUNT
(0031) 00               | P_POS  FCB  $00       NEXT BYTE POSITION TO PRINT
(0032)                  | STR1   RMB  2         ADDRESS OF STRING 1
(0034)                  | STR2   RMB  2         ADDRESS OF STRING 2
(0036)                  | SVPC   RMB  2         SAVE PROGRAM COUNTER
(0038)                  | SVST   RMB  2         SAVE STACK POINTER
(003A)                  | TBLS   RMB  2         SYMBOL TABLE OFFSET
(003C) 00               | TALY   FCB  $00       GENERAL COUNTER - 2
(003D)                  | TRNOFF RMB  2         TRANSLATION TABLE OFFSET
(003F) 00               | UNRESL FCB  $00       NEW ITEM IN SYMBOL TABLE INDICATOR
(0040) 58  20  20       | XOPND  FCC  4,X       FOR COMPARE WITH X
(0043) 20               |
                        | **SKIP
                        | *      **********************     MESSAGES FOLLOW
(0044) 0D  00  00       | ERRA   FCB  $0D,,                   ERROR MESSAGE
(0047) 2A  2A  2A       |        FCC  12,*** ERROR -
(004A) 20  45  52       |
(004D) 52  4F  52       |
(0050) 20  2D  20       |
(0053)                  | ERRB   RMB  1
(0054) 0A  00  00       |        FCB  $0A,,,$20,$04
(0057) 20  04           |
(0059) 0D  0A  0A       | HEADR  FCB  $0D,$0A,$0A,$0A,$0A,,
(005C) 0A  0A  00       |
(005F) 00               |
(0060) 4C  4F  43       |        FCC  14,LOCN B1 B2 B3
(0063) 4E  20  42       |
(0066) 31  20  42       |
(0069) 32  20  42       |
(006C) 33  20           |
(006E) 0A  0A  00       | LINFED FCB  $0A,$0A,,,$04
(0071) 00  04           |
(0073) 0D  00  00       | NXLINE FCB  $0D,,,$12,$04
(0076) 12  04           |
                        | *                    NOTE: TAPE ON CHARACTER IS THE LAST
                        | *                          CHARACTER IN THE ABOVE LITERAL
                        | *
(0078) 0D  0A  0A       | MSG1   FCB  $0D,$0A,$0A,$0A,,
(0078) 0A  00  00       |
(007E) 2A  2A  2A       |        FCC  21,*** UNRESOLVED ITEMS:
(0081) 20  55  4E       |
(0084) 52  45  53       |
(0087) 4F  4C  56       |
(008A) 45  44  20       |
(008D) 49  54  45       |
(0090) 4D  53  3A       |
(0093) 0D  0A  00       | CRLF   FCB  $0D,$0A,,,$04
(0096) 00  04           |
(0098) 0D  0A  00       | MSG2   FCB  $0D,$0A,,
(0098) 0D               |
(009C) 2A  2A  2A       |        FCC  18,*** SYMBOLS, SORT?
(009F) 20  53  59       |
(00A2) 4D  42  4F       |
(00A5) 4C  53  2C       |
(00A8) 20  53  4F       |
(00AB) 52  54  3F       |
(00AE) 0D  0A  0A       |        FCB  $0D,$0A,$0A,,$04
(00B1) 00  04           |
                        | *  LINOUT
(00B3) 00  00           | OUTADR FCB  $00,
(00B5) 20               | BYT1   FCB  $20
(00B6) 20               | BYT2   FCB  $20
(00B7) 20               | BYT3   FCB  $20
(00B8)                  | BUF1   RMB  0         INPUT BUFFER 1 - NOT INTERRUPT FED
(00F1)                  |        ORG  *+057
```

```
                        **SKIP
                        *       ************************   TABLES FOLLOW
(00F1)                  TBL1    RMB   0           TRANSLATION TABLE OFFSET TABLE
(00F1)  00  06                  FCB   0,6         A
(00F3)  06  11                  FCB   6,17        B
(00F5)  17  08                  FCB   23,8        C
(00F7)  1F  04                  FCB   31,4        D
(00F9)  23  03                  FCB   35,3        E
(00FB)  26  03                  FCB   38,3        F
(00FD)  00  00                  FCB   0,0         G
(00FF)  00  00                  FCB   0,0         H
(0101)  29  03                  FCB   41,3        I
(0103)  2C  02                  FCB   44,2        J
(0105)  00  00                  FCB   0,0         K
(0107)  2E  06                  FCB   46,4        L
(0109)  00  00                  FCB   0,0         M
(010B)  32  02                  FCB   50,2        N
(010D)  34  02                  FCB   52,2        O
(010F)  36  02                  FCB   54,2        P
(0111)  00  00                  FCB   0,0         Q
(0113)  38  05                  FCB   56,5        R
(0115)  3D  0A                  FCB   61,10       S
(0117)  47  07                  FCB   71,7        T
(0119)  00  00                  FCB   0,0         U
(011B)  00  00                  FCB   0,0         V
(011D)  4E  01                  FCB   78,1        W
(011F)  00  00                  FCB   0,0         X
(0121)  00  00                  FCB   0,0         Y
(0123)  00  00                  FCB   0,0         Z
                        **SKIP
                        *       ***************  MNEMONIC LOOKUP TABLE
(0125)                  TTBL    RMB   0
                        *                     MNEMONIC OPERATOR KEY  CHAR (2)
                        *                     OP CODE                CHAR (1)
                        *                     CLASS AND FLAG         CHAR (1)
                        *
                        *         A
                        *
(0125)  42  41                  FCC   2,BA              ABA
(0127)  1B  50                  FCB   $1B,$50
(0129)  44  43                  FCC   2,DC              ADC
(012B)  89  10                  FCB   $89,$10
(012D)  44  44                  FCC   2,DD              ADD
(012F)  8B  10                  FCB   $8B,$10
(0131)  4E  44                  FCC   2,ND              AND
(0133)  84  10                  FCB   $84,$10
(0135)  53  4C                  FCC   2,SL              ASL
(0137)  48  30                  FCB   $48,$30
(0139)  53  52                  FCC   2,SR              ASR
(013B)  47  30                  FCB   $47,$30
                        *
                        *         B
                        *
(013D)  43  43                  FCC   2,CC              BCC
(013F)  24  40                  FCB   $24,$40
(0141)  43  53                  FCC   2,CS              BCS
(0143)  25  40                  FCB   $25,$40
(0145)  45  51                  FCC   2,EQ              BEQ
(0147)  27  40                  FCB   $27,$40
(0149)  47  45                  FCC   2,GE              BGE
(014B)  2C  40                  FCB   $2C,$40
(014D)  47  54                  FCC   2,GT              BGT
(014F)  2E  40                  FCB   $2E,$40
(0151)  48  49                  FCC   2,HI              BHI
(0153)  22  40                  FCB   $22,$40
(0155)  49  54                  FCC   2,IT              BIT
(0157)  85  10                  FCB   $85,$10
(0159)  4C  45                  FCC   2,LE              BLE
(015B)  2F  40                  FCB   $2F,$40
(015D)  4C  53                  FCC   2,LS              BLS
(015F)  23  40                  FCB   $23,$40
(0161)  4C  54                  FCC   2,LT              BLT
(0163)  2D  40                  FCB   $2D,$40
(0165)  4D  49                  FCC   2,MI              BMI
(0167)  2B  40                  FCB   $2B,$40
(0169)  4E  45                  FCC   2,NE              BNE
(016B)  26  40                  FCB   $26,$40
(016D)  50  4C                  FCC   2,PL              BPL
(016F)  2A  40                  FCB   $2A,$40
(0171)  52  41                  FCC   2,RA              BRA
(0173)  20  40                  FCB   $20,$40
(0175)  53  52                  FCC   2,SR              BSR
(0177)  8D  40                  FCB   $8D,$40
(0179)  56  43                  FCC   2,VC              BVC
(017B)  28  40                  FCB   $28,$40
(017D)  56  53                  FCC   2,VS              BVS
(017F)  29  40                  FCB   $29,$40
```

```
                        **SKIP
                        *
                        *         C
                        *
(0181)  42  41                  FCC   2,BA              CBA
(0183)  11  50                  FCB   $11,$50
(0185)  4C  43                  FCC   2,LC              CLC
(0187)  0C  50                  FCB   $0C,$50
(0189)  4C  49                  FCC   2,LI              CLI
(018B)  0E  50                  FCB   $0E,$50
(018D)  4C  52                  FCC   2,LR              CLR
(018F)  4F  30                  FCB   $4F,$30
(0191)  4C  56                  FCC   2,LV              CLV
(0193)  0A  50                  FCB   $0A,$50
(0195)  4D  50                  FCC   2,MP              CMP
(0197)  81  10                  FCB   $81,$10
(0199)  4F  4D                  FCC   2,OM              COM
(019B)  43  30                  FCB   $43,$30
(019D)  50  58                  FCC   2,PX              CPX
(019F)  8C  11                  FCB   $8C,$11
                        *
                        *         D
                        *
(01A1)  41  41                  FCC   2,AA              DAA
(01A3)  19  50                  FCB   $19,$50
(01A5)  45  43                  FCC   2,EC              DEC
(01A7)  4A  30                  FCB   $4A,$30
(01A9)  45  53                  FCC   2,ES              DES
(01AB)  34  50                  FCB   $34,$50
(01AD)  45  58                  FCC   2,EX              DEX
(01AF)  09  50                  FCB   $09,$50
                        *
                        *         E
                        *
(01B1)  4E  44                  FCC   2,ND              END
(01B3)  00  E0                  FCB   $00,$E0
(01B5)  4F  52                  FCC   2,OR              EOR
(01B7)  88  10                  FCB   $88,$10
(01B9)  51  55                  FCC   2,QU              EQU
(01BB)  00  F0                  FCB   $00,$F0
                        *
                        *         F
                        *
(01BD)  43  42                  FCC   2,CB              FCB
(01BF)  01  F0                  FCB   $01,$F0
(01C1)  43  43                  FCC   2,CC              FCC
(01C3)  02  F0                  FCB   $02,$F0
(01C5)  44  42                  FCC   2,DB              FDB
(01C7)  03  F0                  FCB   $03,$F0
                        *
                        *         (G, H), I
                        *
(01C9)  4E  43                  FCC   2,NC              INC
(01CB)  4C  30                  FCB   $4C,$30
(01CD)  4E  53                  FCC   2,NS              INS
(01CF)  31  50                  FCB   $31,$50
(01D1)  4E  58                  FCC   2,NX              INX
(01D3)  08  50                  FCB   $08,$50
                        **SKIP
                        *
                        *         J
                        *
(01D5)  4D  50                  FCC   2,MP              JMP
(01D7)  4E  31                  FCB   $4E,$31
(01D9)  53  52                  FCC   2,SR              JSR
(01DB)  8D  31                  FCB   $8D,$31
                        *
                        *         (K), L
                        *
(01DD)  44  41                  FCC   2,DA              LDA
(01DF)  86  10                  FCB   $86,$10
(01E1)  44  53                  FCC   2,DS              LDS
(01E3)  8E  11                  FCB   $8E,$11
(01E5)  44  58                  FCC   2,DX              LDX
(01E7)  CE  11                  FCB   $CE,$11
(01E9)  53  52                  FCC   2,SR              LSR
(01EB)  44  30                  FCB   $44,$30
                        *
                        *         (M), N
                        *
(01ED)  45  47                  FCC   2,EG              NEG
(01EF)  40  30                  FCB   $40,$30
(01F1)  4F  50                  FCC   2,OP              NOP
(01F3)  01  50                  FCB   $01,$50
```

```
                        *
                        *         O
                        *
(01F5)  52  41          *       FCC  2,RA           ORA
(01F7)  8A  10          *       FCB  $8A,$10
(01F9)  52  47          *       FCC  2,RG           ORG
(01FB)  04  F0          *       FCB  $04,$F0
                        *
                        *         P
                        *
(01FD)  53  48          *       FCC  2,SH           PSH
(01FF)  36  00          *       FCB  $36,$00
(0201)  55  4C          *       FCC  2,UL           PUL
(0203)  32  00          *       FCB  $32,$00
                        *
                        *         (Q), R
                        *
(0205)  4D  42          *       FCC  2,MB           RMB
(0207)  05  F0          *       FCB  $05,$F0
(0209)  4F  4C          *       FCC  2,OL           ROL
(020B)  49  30          *       FCB  $49,$30
(020D)  4F  52          *       FCC  2,OR           ROR
(020F)  46  30          *       FCB  $46,$30
(0211)  54  49          *       FCC  2,TI           RTI
(0213)  3B  50          *       FCB  $3B,$50
(0215)  54  53          *       FCC  2,TS           RTS
(0217)  39  50          *       FCB  $39,$50
                        **SKIP
                        *
                        *         S
                        *
(0219)  42  41          *       FCC  2,BA           SBA
(021B)  10  50          *       FCB  $10,$50
(021D)  42  43          *       FCC  2,BC           SBC
(021F)  82  10          *       FCB  $82,$10
(0221)  45  43          *       FCC  2,EC           SEC
(0223)  0D  50          *       FCB  $0D,$50
(0225)  45  49          *       FCC  2,EI           SEI
(0227)  0F  50          *       FCB  $0F,$50
(0229)  45  56          *       FCC  2,EV           SEV
(022B)  0B  50          *       FCB  $0B,$50
(022D)  54  41          *       FCC  2,TA           STA
(022F)  87  20          *       FCB  $87,$20
(0231)  54  53          *       FCC  2,TS           STS
(0233)  8F  21          *       FCB  $8F,$21
(0235)  54  58          *       FCC  2,TX           STX
(0237)  CF  21          *       FCB  $CF,$21
(0239)  55  42          *       FCC  2,UB           SUB
(023B)  80  10          *       FCB  $80,$10
(023D)  57  49          *       FCC  2,WI           SWI
(023F)  3F  50          *       FCB  $3F,$50
                        *
                        *         T
                        *
(0241)  41  42          *       FCC  2,AB           TAB
(0243)  16  50          *       FCB  $16,$50
(0245)  41  50          *       FCC  2,AP           TAP
(0247)  06  50          *       FCB  $06,$50
(0249)  42  41          *       FCC  2,BA           TBA
(024B)  17  50          *       FCB  $17,$50
(024D)  50  41          *       FCC  2,PA           TPA
(024F)  07  50          *       FCB  $07,$50
(0251)  53  54          *       FCC  2,ST           TST
(0253)  4D  30          *       FCB  $4D,$30
(0255)  53  58          *       FCC  2,SX           TSX
(0257)  30  50          *       FCB  $30,$50
(0259)  58  53          *       FCC  2,XS           TXS
(025B)  35  50          *       FCB  $35,$50
                        *
                        *         (U, V), W
                        *
(025D)  41  49          *       FCC  2,AI           WAI
(025F)  3E  50          *       FCB  $3E,$50
                        *
                        *         (X, Y, Z)
                        *
```

```
                        **SKIP
                        *
                        *       *************** UNRESOLVED FWD REFERENCE TABLE
                        *
(0261)                  FTBL    RMB  0
                        *
                        *         ADDR OF SYMB IN SYMB TABLE  CHAR (2)
                        *         PGMCNT OF UNRESOLVED REF    CHAR (2)
                        *         TYPE INDICATOR              CHAR (1)   -1 = ONE BYTE OUT
                        *                                                 0 = TWO BYTES OUT
                        *                                                +1 = RELATIVE
                        *
(02DE)                  *       ORG  FTBL+125     SET FOR 25 FIVE BYTE ENTRIES
                        *
                        *       *************** SYMBOL TABLE
                        *
(02DE)  01              *       FCB  $01
(02DF)                  STBL    RMB  0
                        *
                        *         LABEL NAME  CHAR (4)
                        *         LABEL ADDR  CHAR (2)
                        *
(02DF)  01              *       NOP               SPACE FOR
(02E0)  01              *       NOP               USER DEFINED
(02E1)  01              *       NOP               ROUTINES
(02E2)  39              *       RTS
(078F)                  *       ORG  STBL+1200    SET FOR 200 SIX BYTE ENTRIES
(078F)  00              *       FCB  $00          TABLE STOP BYTE
                        *
                        *       *************** EXTERNAL ADDRESS TABLE
                        *
(0790)  7E  E1  D1      CHROUT  JMP  $E1D1
(0793)  7E  E1  AC      CHRIN   JMP  $E1AC
(0796)  36              HEAOUT  PSHA
(0797)  BD  E0  67              JSR  $E067
(079A)  32                      PULA
(079B)  7E  E0  6B              JMP  $E06B
(079E)  7E  E0  CC      BLKOUT  JMP  $E0CC
(07A1)  7E  E0  7E      STROUT  JMP  $E07E
                        **SKIP
                        *
                        *
                        *       ********************************************************************
                        *                       MAIN PROCEDURE LOOP
                        *
(07A4)  BD  02  DE      MAIN    JSR  STBL-1       EXECUTE USER ROUTINES
(0001)  07  A4          *
(07A7)  CE  02  61              LDX  #FTBL        CLEAR THE SYMBOL TABLE AND THE
(07AA)  6F  00          CLR1    CLR  X              FORWARD JUMP TABLE
(07AC)  08                      INX
(07AD)  8C  07  8F              CPX  #STBL+1200
(07B0)  26  F8                  BNE  CLR1
(07B2)  86  39                  LDAA #$39         BLOCK FURTHER
(07B4)  B7  02  DE              STAA STBL-1         EXECUTION OF USER PROCS
                        *
                        *         THIS IS THE CONTINUE ENTRY POINT
                        *
(07B7)  CE  00  B8      ENTRY   LDX  #BUF1        INITIALIZE
(07BA)  DF  2A                  STX  NXCHAR         INCOMING CHARACTER POINTER
(07BC)  7F  00  1C              CLR  EOJFL        CLEAR FLAGS
(07BF)  7F  00  31              CLR  P_POS
(07C2)  CE  07  B7              LDX  #ENTRY       LOAD SECONDARY ENTRY POINT FOR MIKBUG MONITOR
(07C5)  FF  A0  48              STX  $A048
(07C8)  8E  A0  7F              LDS  #$A07F       LOAD STACK POINTER FOR MIKBUG MONITOR
(07CB)  CE  00  59              LDX  #HEADR       WRITE OUT
(07CE)  BD  07  A1              JSR  STROUT         HEADING
(07D1)  BD  00  00              JSR  CMD          MAIN PROCEDURE LOOP - CALL PROC-COMMANDS
(07D4)  BD  00  00              JSR  EOJ          CALL END OF JOB
(07D7)  7E  E0  E3              JMP  $E0E3        JUMP TO MONITOR MAINLINE
```

```
                        **SKIP  ********************************************************************
                        *                                                                           *
                        *                          I/O SUBROUTINES                                  *
                        *                                                                           *
                        *****************************************************************************
                        *                                  LOAD BUFFER FROM TERMINAL
                        *
07DA  CE 00 88          LOUBUF LDX  #BUF1       SET INDEX TO BUFFER
07DD  86 3E                    LDAA #'>                                  *** PROMPT CHARACTER
07DF  BD 07 90                 JSR  CHROUT
07E2  BD 07 93          BUFLOD JSR  CHRIN       GET CHAR
07E5  81 21                    CMPA #'!         PROCESS BACKSPACE        *** BACKSPACE CHARACTER
07E7  26 00                    BNE  BUFGO
07E9  8C 00 88                 CPX  #BUF1
07EC  27 F4                    BEQ  BUFLOD
07EE  09                       DEX
07EF  20 F1                    BRA  BUFLOD
07F1  A7 00             BUFGO  STAA X           STORE CHARACTER
07E8  08
07F3  91 18                    CMPA CR          IF NOT CARRIAGE RETURN THEN
07F5  26 01                    BNE  **+3           CONTINUE
07F7  39                       RTS
07F8  8C 00 F0                 CPX  #BUF1+56    IF NOT AT END OF BUFFER
07FB  27 E5                    BEQ  BUFLOD
07FD  81 20                    CMPA #$20        AND IF CHARACTER
07FF  2B E1                    BMI  BUFLOD        IS BETWEEN HEX 20 AND 5F
0801  81 5F                    CMPA #$5F
0803  22 DD                    BHI  BUFLOD
0805  08                       INX              INCREMENT INDEX
0806  20 DA                    BRA  BUFLOD      CONTINUE
                        **SKIP
                        *
                        *      ***************************************************************
                        *                                  RETRIEVE NEXT CHAR
                        *
0808  DF 24             NX_CHR STX  IXS3        SAVE IX
080A  DE 2A                    LDX  NXCHAR      LOAD NEXT CHAR TO USE POSITION INTO IX
080C  96 17                    LDAA CC          IF LAST CHAR
080E  91 18                    CMPA CR            WAS NOT A CARRIAGE RETURN,
0810  26 00                    BNE  NXC9          THEN CONTINUE
0812  7D 00 31                 TST  P_POS       ELSE IF P_POS = NULL THEN
0815  27 03                    BEQ  **5           GO TO GET INPUT
0817  BD 00 00                 JSR  WRITE       ELSE WRITE OUT LINE OF PRINT
081A  86 01                    LDAA #$01          SET P-POS
081C  97 31                    STAA P_POS           TO 1
081E  BD 07 DA                 JSR  LODBUF        LOAD BUFFER
0821  CE 00 88                 LDX  #BUF1         SET CURRENT
0824  DF 2A                    STX  NXCHAR          CHAR POINTER
0826  A6 00             NXC9   LDAA X           LOAD CHAR INTO A
0811  16
0828  97 17                    STAA CC            AND CC
082A  08                       INX              INCREMENT NEXT CHAR PTR
082B  DF 2A                    STX  NXCHAR
082D  DE 24                    LDX  IXS3        RESTORE IX
082F  39                       RTS                AND RETURN
                        **SKIP
                        *
                        *      ***************************************************************
                        *                          FIND NEXT ITEM FROM INPUT STREAM
                        *
0830  96 17             NX_ITM LDAA CC          IF NEXT ITEM
0832  91 0A             NXI1   CMPA BLNK          BLANK THEN
0834  27 01                    BEQ  **+3          CONTINUE
0836  39                      RTS              RETURN
0837  BD 08 08                JSR  NX_CHR        TRY NEXT CHARACTER FOR BEING A NON BLANK
083A  20 F6                    BRA  NXI1
                        *
                        *
                        *      ***************************************************************
                        *                  LOAD ITEM FROM INPUT STREAM INTO 'NAME'
                        *
083C  DE 0A             LD_ITM LDX  BLNK        BLANK OUT NAME
083E  DF 05                    STX  NAME
0840  DF 07                    STX  NAME+2
0842  CE 00 05                 LDX  #NAME
                        *
0845  D6 17             LDI1   LDAB CC          LOAD CURRENT CHAR
0847  D1 0A                    CMPB BLNK
0849  27 00                    BEQ  LDI9        IF BLANK, RETURN
084B  D1 18                    CMPB CR          IF CARRIAGE RETURN,
084D  27 00                    BEQ  LDI9          THEN RETURN
084F  C1 2B                    CMPB #$2B        IF PLUS
0851  27 00                    BEQ  LDI9          THEN  RETURN
0853  C1 2D                    CMPB #$2D        IF MINUS
0855  27 00                    BEQ  LDI9          THEN  RETURN
0857  C1 2C                    CMPB #$2C        IF COMMA
0859  27 00                    BEQ  LDI9          THEN  RETURN
085B  E7 00                    STAB X           LOAD CHAR
085D  8C 00 08                 CPX  #NAME+3       IF NAME IS FULL
0860  27 01                    BEQ  **+3          DON'T
0862  08                       INX              INCREMENT INDEX
0863  BD 08 08                JSR  NX_CHR        GET NEXT CHAR
0866  20 DD                    BRA  LDI1
0868  39              LDI9   RTS
084A  1D
084E  19
0852  15
0856  11
085A  0D
                        **SKIP
                        *
                        *
                        *      ***************************************************************
                        *                                  ERROR ROUTINE
                        *
0869  97 53             ERRS   STAA ERRB        LOAD ERROR MESSAGE
086B  DF 20                    STX  IXS1        SAVE INDEX
086D  CE 00 44                 LDX  #ERRA       WRITE
0870  BD 07 A1                 JSR  STROUT        ERROR MSG
0873  DE 20                    LDX  IXS1        RESTORE INDEX
0875  7C 00 03                 INC  ERORSW      SET ERROR SWITCH (ON)
0878  39                       RTS              RETURN
                        *
                        *      ***************************************************************
                        *                                  CALCULATE PRINT LINE
                        *
0879  7D 00 03          CALCWR TST  ERORSW      IF ERROR CONDITION EXISTS,
087C  23 01                    BLS  **+3          RETURN
087E  39                       RTS
087F  96 16                    LDAA BYTOUT
0881  D6 31                    LDAB P_POS       IF PRINT_POS
0883  C1 01                    CMPB #$01          = 1 THEN
0885  27 00                    BEQ  CLC4          GO TO INITIALIZE (WITHOUT PRINT)
0887  C1 02                    CMPB #$02        IF PRINT_POS
0889  26 00                    BNE  CLC2          NOT = 2 THEN CONTINUE
088B  97 B6                    STAA BYT2        ELSE LOAD SECOND BYTE
088D  7C 00 31                 INC  P_POS         INCREASE PRINT_POS
0890  20 00                    BRA  CLC9          GO TO END
0892  C1 03             CLC2   CMPB #$03        IF PRINT_POS
088A  07
0894  26 00                    BNE  CLC3          NOT = 3 THEN CONTINUE
0896  97 B7                    STAA BYT3        ELSE LOAD THIRD BYTE
0898  7C 00 31                 INC  P_POS         INCREASE PRINT_POS
089B  20 00                    BRA  CLC9          GO TO END
089D  BD 00 00          CLC3   JSR  WRITE       WRITE OUT CURRENT LINE
08A0  07
08A0  DE 2F             CLC4   LDX  PGMCNT      LOAD
08B6  19
08A2  DF B3                    STX  OUTADR        CURRENT ADDRESS
08A4  96 16                    LDAA BYTOUT
08A6  97 B5                    STAA BYT1        LOAD BYTE1
08A8  86 02                    LDAA #$02        SET PRINT_POS
08AA  97 31                    STAA P_POS         = 2
08AC  DE 2F             CLC9   LDX  PGMCNT      INCREASE
08B1  1A
089C  0F
08AE  08                       INX                PGMCNT
08AF  DF 2F                    STX  PGMCNT
0881  39                       RTS              RETURN
```

```
                        **SKIP
                        *
                        *      ************************************************
                        *                        WRITE ROUTINE
                        *
(08B2)  DF  24          WRITE  STX  IXS3        SAVE INDEX
(08B4)  08  B2
(089E)  08  B2
(08B4)  CE  00  73              LDX  #NXLINE    START NEXT
(08B7)  BD  07  A1              JSR  STROUT       PRINT LINE
(08BA)  CE  75  30              LDX  #$7530     LOAD IX WITH 30,000
(08BD)  6D  00          WRT0    TST  X            EXECUTE TIMING LOOP
(08BF)  09                      DEX
(08C0)  26  FB                  BNE  WRT0                                   ***
(08C2)  86  01                  LDAA #$01       WRITE LOADER RECOGNITION  (1)
(08C4)  BD  07  90              JSR  CHROUT       CHARACTER               ***
(08C7)  96  B3                  LDAA OUTADR     WRITE
(08C9)  BD  07  96              JSR  HEXOUT       OUT FOUR
(08CC)  96  B4                  LDAA OUTADR+1     BYTES
(08CE)  BD  07  96              JSR  HEXOUT       OF ADDRESS
(08D1)  C6  01                  LDAB #$01
(08D3)  CE  00  B5              LDX  #BYT1
(08D6)  D1  31          WRT1    CMPB P_POS      LOOP THROUGH BYTES OUT
(08D8)  2B  00                  BMI  WRT2       IF P_POS DONE THEN
(08DA)  26  00                  BNE  WRT4
(08DC)  86  01                  LDAA #$01         WRITE OUT               ***
(08DE)  BD  07  90              JSR  CHROUT         LOADER STOP CHARACTER (1)
(08E1)  86  14                  LDAA #$14         WRITE OUT               ***
(08E3)  BD  07  90              JSR  CHROUT         TAPE OFF CHR
(08E6)  BD  07  9E      WRT4    JSR  BLKOUT
(08D8)  0A
(08E9)  BD  07  9E              JSR  BLKOUT
(08EC)  BD  07  9E              JSR  BLKOUT
(08EF)  20  00                  BRA  WRT3
(08F1)  BD  07  9E      WRT2    JSR  BLKOUT     WRITE OUT BLANK
(08D9)  17
(08F4)  A6  00                  LDAA X          GET NEXT BYTE OUT
(08F6)  BD  07  96              JSR  HEXOUT     PRINT IT IN HEX
(08F9)  08              WRT3    INX             GO TO NEXT BYTE
(08F0)  08
(08FA)  5C                      INCB
(08FB)  C1  04                  CMPB #$04       IF ALL NOT OUT THEN
(08FD)  26  D7                  BNE  WRT1         GO TO NEXT LOOP
(08FF)  96  31                  LDAA P_POS      IF REQUIRED,
(0901)  81  04                  CMPA #4
(0903)  26  00                  BNE  WRT5
(0905)  86  01                  LDAA #$01       WRITE OUT               ***
(0907)  BD  07  90              JSR  CHROUT       LOADER STOP CHARACTER (2)
(090A)  86  14                  LDAA #$14       WRITE OUT               ***
(090C)  BD  07  90              JSR  CHROUT       TAPE OFF CHARACTER
                        **SKIP
                        *
                        ******* WRITE CONTINUED
                        *
(090F)  BD  07  9E      WRT5    JSR  BLKOUT     WRITE OUT FINAL BLANK
(0904)  0A
(0912)  DE  2F                  LDX  PGMCNT     SET UP NEXT
(0914)  DF  B3                  STX  OUTADR       ADDRESS IN WRITE STRING
(0916)  DE  24                  LDX  IXS3       RESTORE INDEX
(0918)  86  0A                  LDAA #$0A       WRITE OUT
(091A)  BD  07  90              JSR  CHROUT       LINE FEED
(091D)  96  0A                  LDAA BLNK         BLANK OUT
(091F)  97  B5                  STAA BYT1         OUTPUT AREA
(0921)  97  B6                  STAA BYT2
(0923)  97  B7                  STAA BYT3
(0925)  86  01                  LDAA #$01       RESET OUTPUT
(0927)  97  31                  STAA P_POS        POINTER
(0929)  39                      RTS
```

```
                        **SKIP
                        ************************************************************
                        *                                                          *
                        *                PROCESS COMMAND PROCEDURE                 *
                        *                                                          *
                        ************************************************************
(092A)  CE  00  04      CMD     LDX  #MOD       GET STARTING ADDRESS
(0702)  09  2A
(092D)  D6  0A                  LDAB BLNK       SET CHARACTER TO LOAD WITH
(092F)  7F  00  03              CLR  ERORSW     CLEAR ERROR SWITCH
(0932)  7F  00  3F              CLR  UNRESL     CLEAR UNRESOLVED SWITCH
(0935)  E7  00          MN01    STAB X          BLANK CURRENT BYTE
(0937)  08                      INX             GO TO NEXT BYTE
(0938)  8C  00  0A              CPX  #BLNK      LAST TO BE BLANKED?
(0938)  26  F8                  BNE  MN01         IF NOT, CONTINUE
(093D)  96  18                  LDAA CR
(093F)  97  17                  STAA CC
(0941)  BD  08  08              JSR  NX_CHR     LOAD FIRST CHAR
(0944)  91  18                  CMPA CR
(0946)  27  E2                  BEQ  CMD        BLANK LINE - FIRST CHAR IS A CARRAIGE RETURN
(0948)  81  2A                  CMPA #$2A       COMMENT ENTRY - IF FIRST CHAR IS AN '*'
(094A)  27  DE                  BEQ  CMD          THEN GO TO NEXT COMMAND
(094C)  91  0A                  CMPA BLNK       TEST FOR LABEL AND IF ABSENT
(094E)  27  00                  BEQ  MN04         GO TO START-VERB
(0950)  BD  08  3C              JSR  LD_ITM     ELSE, PROCESS LABEL
(0953)  7C  00  26              INC  LAB_SW
(0956)  BD  00  00              JSR  SYMB_S
(0959)  BD  08  30      MN04    JSR  NX_ITM     LOOK FOR START OF VERB
(094F)  09
(095C)  91  18                  CMPA CR
(095E)  27  00                  BEQ  MN08       BLANK LINE WITH A SPACE OR LABEL IN CC-1
(0960)  BD  08  3C              JSR  LD_ITM     LOAD VERB
(0963)  86  02                  LDAA #2
(0965)  97  19                  STAA CMPLEN
(0967)  BD  00  00              JSR  TRNS       LOOK UP VERB
(096A)  86  04                  LDAA #4
(096C)  97  19                  STAA CMPLEN
(096E)  7D  00  03              TST  ERORSW     BRANCH IF ERROR
(0971)  22  B7                  BHI  CMD          TO START OF NEXT COMMAND
(0973)  96  18                  LDAA CLASS
(0975)  81  0E                  CMPA #$0E
(0977)  26  00                  BNE  MN05       CHECK FOR END-OF-PROGRAM-VERB
(0979)  BD  0A  B2              JSR  WRITE        WRITE OUT LAST LINE
(097C)  39                      RTS               AND RETURN
(097D)  D6  2E          MN05    LDAB OPCODE     LOAD POSSIBLE MACRO NUMBER INTO B
(0978)  04
(097F)  81  0F                  CMPA #$0F
(0981)  26  00                  BNE  MN06       CHECK FOR MACRO
(0983)  BD  00  00              JSR  PSEUDO       AND PROCESS IF FOUND
(0986)  20  00                  BRA  MN08         THEN GO TO NEXT COMMAND
                        **SKIP
                        *
                        ******* CMD CONTINUED
                        *
(0988)  BD  08  30      MN06    JSR  NX_ITM     SEARCH FOR THE START OF THE OPERANDS
(0982)  05
(0988)  81  23                  CMPA #'#          IF IT STARTS
(098D)  26  00                  BNE  MN07         WITH '#' THEN
(098F)  97  04                  STAA MOD          LOAD MOD
(0991)  BD  08  08              JSR  NX_CHR       WITH '#' AND CONTINUE
(0994)  BD  08  30      MN07    JSR  NX_ITM     SEARCH REST OF OPERAND
(098E)  05
(0997)  BD  00  00              JSR  M_SRCH       FOR VERB MODIFIER
(099A)  BD  00  00              JSR  VERB       PROCESS VERB
(099D)  BD  00  00      MN08    JSR  R_FWD      RESOLVE FORWARD REFERENCES
(095F)  3D
(0987)  15
(09A0)  7E  09  2A              JMP  CMD        GO TO NEXT COMMAND
```

```
                        **SKIP
                        *
                        *       ********************************************************************
                        *                              SEARCH OPERAND FOR X OR *X
                        *
(09A3)  DE  2A          M_SRCH LDX  NXCHAR      SET INDEX TO CURRENT PLACE IN BUFFER
(0998)  09  A3
(09A5)  96  17                 LDAA CC          CHECK NEXT CHAR FOR
(09A7)  81  58                 CMPA #'X          'X'
(09A9)  26  00                 BNE  SCH5        IF NOT THEN GO TO 'CHECK FOR *X'
(09AB)  A6  00                 LDAA X           IF NEXT CHAR IS
(09AD)  91  0A                 CMPA BLNK          BLANK
(09AF)  27  00                 BEQ  SCH3          OR
(09B1)  91  1B                 CMPA CR            CR
(09B3)  27  00                 BEQ  SCH3          OR
(09B5)  81  2B                 CMPA #$2B          +
(09B7)  27  00                 BEQ  SCH3          OR
(09B9)  81  2D                 CMPA #$2D          -
(09BB)  27  00                 BEQ  SCH3          THEN GO TO LOAD MOD
(09BD)  20  00                 BRA  SCH5        ELSE GO TO 'CHECK FOR *X'
(09BF)  86  58          SCH3   LDAA #'X         LOAD 'X'
(09B0)  0E
(09B4)  0A
(09B8)  06
(09BC)  02
(09C1)  97  04                 STAA MOD           INTO MOD
(09C3)  86  30                 LDAA #'0
(09C5)  97  17                 STAA CC
(09C7)  20  00                 BRA  SCH9        GO TO END
(09C9)  A6  00          SCH5   LDAA X           CHECK FOR *X
(09AA)  1E
(09BE)  0A
(09C8)  08                     INX              IF NEXT CHAR IS
(09CC)  91  0A                 CMPA BLNK          BLANK
(09CE)  27  00                 BEQ  SCH9          OR
(09D0)  91  1B                 CMPA CR            CR
(09D2)  27  00                 BEQ  SCH9          THEN GO TO END
(09D4)  81  2C                 CMPA #$2C        IF IT IS A COMMA THEN CONTINUE
(09D6)  26  F1                 BNE  SCH5          ELSE
(09D8)  A6  00                 LDAA X               LOAD CHAR
(09DA)  97  04                 STAA MOD             INTO MOD
(09DC)  39              SCH9   RTS              RETURN
(09CF)  0C
(09C8)  13
(09D3)  08
                        **SKIP
                        *
                        *       ********************************************************************
                        *                          FIND VERB IN TRANSLATION TABLE
                        *
(09DD)  96  05          TRNS   LDAA NAME        VERB TRANSLATION
(0968)  09  DD
(09DF)  81  41                 CMPA #'A         IF VERB STARTS ABOVE A
(09E1)  2C  06                 BGE  *+8           CONTINUE
(09E3)  86  41                 LDAA #'A         ELSE LOAD ERROR WITH 'A'
(09E5)  BD  08  69             JSR  ERRS          SIGNAL ERROR
(09E8)  39                     RTS                AND RETURN
(09E9)  81  5A                 CMPA #'Z         IF VERB STARTS BELOW Z
(09EB)  2F  06                 BLE  *+8           CONTINUE
(09ED)  86  41                 LDAA #'A         ELSE LOAD ERROR WITH 'A'
(09EF)  BD  08  69             JSR  ERRS          SIGNAL ERROR
(09F2)  39                     RTS                AND RETURN
(09F3)  84  1F                 ANDA #$1F        STRIP OFF HIGH BITS
(09F5)  4A                     DECA
(09F6)  48                     ASLA             DOUBLE A-1 GIVING ENTRY INTO TBL1
(09F7)  CE  00  F1             LDX  #TBL1
(09FA)  4D              TR01   TSTA             INCREMENT
(09FB)  27  00                 BEQ  TR02          INDEX
(09FD)  08                     INX                TO
(09FE)  4A                     DECA               LETTER OFFSET
(09FF)  20  F9                 BRA  TR01
(0A01)  E6  01          TR02   LDAB 1,X         LOAD
(09FC)  04
(0A03)  D7  29                 STAB LTRL          LETTER LENGTH
(0A05)  E6  00                 LDAB X           MULTIPLY
(0A07)  4F                     CLRA               VERB OFFSET
(0A08)  58                     ASLB               BY 4 TO OBTAIN
(0A09)  49                     ROLA               TABLE OFFSET (TTBL)
(0A0A)  58                     ASLB
(0A0B)  49                     ROLA
(0A0C)  CE  01  25             LDX  #TTBL       ADD
(0A0F)  DF  3D                 STX  TRNOFF        TABLE OFFSET
(0A11)  DB  3E                 ADDB TRNOFF+1      TO
(0A13)  99  3D                 ADCA TRNOFF        TABLE STARTING POS
(0A15)  D7  3E                 STAB TRNOFF+1
(0A17)  97  3D                 STAA TRNOFF
(0A19)  CE  00  06             LDX  #NAME+1     LOAD NAME
(0A1C)  DF  32                 STX  STR1          INTO STRING1
(0A1E)  7D  00  29      TR03   TST  LTRL        END OF LETTER LENGTH?
(0A21)  26  06                 BNE  *+8           IF NOT TRY NEXT ONE
(0A23)  86  42                 LDAA #'B         ELSE LOAD ERROR 'B'
(0A25)  BD  08  69             JSR  ERRS          SIGNAL ERROR - VERB NOT FOUND
(0A28)  39                     RTS                AND RETURN
(0A29)  DE  3D                 LDX  TRNOFF
(0A2B)  DF  34                 STX  STR2        LOAD VERB INTO STRING 2
(0A2D)  BD  00  00             JSR  STR_CM      COMPARE VERB WITH NAME
(0A30)  7D  00  11             TST  ANSW        END IF
(0A33)  23  00                 BLS  TR08          NOT EQUAL, CONTINUE
(0A35)  BD  00  00             JSR  OP_GEN
(0A38)  7D  00  11             TST  ANSW
(0A38)  23  00                 BLS  TR08
(0A3D)  39                     RTS                ELSE RETURN
(0A3E)  08              TR08   INX              ADD 4 TO
(0A34)  09
(0A3C)  01
(0A3F)  08                     INX                ADDR OF
(0A40)  08                     INX                VERB
(0A41)  08                     INX                IN TABLE
(0A42)  DF  3D                 STX  TRNOFF
(0A44)  CE  00  06             LDX  #NAME+1
(0A47)  DF  32                 STX  STR1
(0A49)  7A  00  29             DEC  LTRL
(0A4C)  20  D0                 BRA  TR03
                        **SKIP
                        *
                        *       ********************************************************************
                        *                          CHECK OP CODE SPECIFICS
                        *
(0A4E)  A6  02          OP_GEN LDAA 2,X
(0A36)  0A  4E
(0A50)  97  2E                 STAA OPCODE      LOAD OPCODE
(0A52)  A6  03                 LDAA 3,X
(0A54)  16                     TAB
(0A55)  C4  0F                 ANDB #$0F        LOAD FLAGS
(0A57)  D7  1D                 STAB FLAGS
(0A59)  44                     LSRA             SHIFT
(0A5A)  44                     LSRA               CLASS
(0A5B)  44                     LSRA               TO RIGHT
(0A5C)  44                     LSRA
(0A5D)  97  18                 STAA CLASS       LOAD CLASS
(0A5F)  D6  08                 LDAB NAME+3      B = OPERAND
(0A61)  D1  0A                 CMPB BLNK        IF NOT BLANK,
(0A63)  26  00                 BNE  OP03          CONTINUE
(0A65)  7D  00  1D             TST  FLAGS       ELSE IF FLAG > 0
(0A68)  22  00                 BHI  OP09          GO TO END
(0A6A)  81  02                 CMPA #2          IF CLASS <= 2 THEN
(0A6C)  23  00                 BLS  OP08          NO MATCH
(0A6E)  20  00                 BRA  OP09        ELSE GO TO END
(0A70)  81  03          OP03   CMPA #3          FOR OPERAND PRESENT, IF CLASS > 3 THEN
(0A64)  08
(0A72)  22  00                 BHI  OP08          NO MATCH
(0A74)  7D  00  1D             TST  FLAGS       IF FLAG ~= 0 THEN
(0A77)  22  00                 BHI  OP08          NO MATCH
(0A79)  C1  41                 CMPB #'A         IF OPERAND = 'A' THEN
(0A7B)  27  00                 BEQ  OP06          GO TO ADJUST CLASS
(0A7D)  C1  42                 CMPB #'B         IF OPERAND ~= 'B'
(0A7F)  26  00                 BNE  OP08          NO MATCH
(0A81)  D6  2E                 LDAB OPCODE      ADJUST OPCODE
(0A83)  4D                     TSTA
(0A84)  26  00                 BNE  OP04        FOR CLASS 0, ADD $01 FOR B
(0A86)  CB  01                 ADDB #$01            (PSH, PUL)
(0A88)  D7  2E                 STAB OPCODE
(0A8A)  20  00                 BRA  OP06          GO TO ADJUST CLASS
(0A8C)  81  03          OP04   CMPA #3          ADJUST CLASS 3 AND 1,2
(0A85)  06
(0A8E)  27  02                 BEQ  *+4
(0A90)  CB  30                 ADDB #$30
(0A92)  CB  10                 ADDB #$10
(0A94)  D7  2E                 STAB OPCODE
```

```
                        **SKIP
                        *
                        *****  OP_GEN CONT.
                        *
40A96)  4C              OP06   INCA            ADJUST CLASS
40A7C)  19
40A8B)  0A
40A97)  85  02                 BITA #02        SET CLASS 0 AND 3 TO 5
40A99)  26  00                 BNE  OP09
40A9B)  86  05                 LDAA #5
40A9D)  97  18                 STAA CLASS
40A9F)  20  00                 BRA  OP09
40AA1)  7F  00  11      OP08   CLR  ANSW        NO MATCH
40A73)  2D
40A6D)  33
40A78)  28
40A80)  20
40AA4)  39              OP09   RTS             RETURN
40A69)  3A
40A6F)  34
40A9A)  09
40AA0)  03
                        *
                        *
                        *      ********************************************************************
                        *                      COMPARE TWO STRINGS
                        *
40AA5)  36              STR_CM PSHA            SAVE A
40A2E)  0A  A5
40AA6)  DF  20                 STX  IXS1         IX
40AA8)  9F  38                 STS  SVST         STACK PTR
40AAA)  7F  00  11             CLR  ANSW       CLEAR ANSWER (NO MATCH)
40AAD)  96  19                 LDAA CMPLEN     SET COMPARE LENGTH
40AAF)  97  3C                 STAA TALY
40AB1)  DE  32                 LDX  STR1       LOAD
40AB3)  9E  34                 LDS  STR2         STRINGS
40AB5)  34                     DES             DECREMENT STACK SO PULL WILL WORK
40AB6)  32              ST01   PULA            GET NEXT CHARACTER
40AB7)  A1  00                 CMPA X          COMPARE
40AB9)  26  00                 BNE  ST99       IF NOT EQUAL, GO TO NOMATCH
40ABB)  08                     INX               ELSE
40ABC)  7A  00  3C             DEC  TALY         GO TO NEXT CHAR IF NOT DONE
40ABF)  26  F5                 BNE  ST01
40AC1)  7C  00  11             INC  ANSW       ELSE SET ANSWER (MATCH)
40AC4)  DE  20          ST99   LDX  IXS1       RESTORE IX,
40ABA)  09
40AC6)  9E  38                 LDS  SVST         STACK PTR,
40AC8)  32                     PULA              AND A
40AC9)  39                     RTS             RETURN
                        **SKIP
                        *
                        *      ********************************************************************
                        *                      PROCESS VERB AND OPERANDS
                        *
40ACA)  96  18          VERB   LDAA CLASS        A = VERB CLASS
40998)  0A  CA
40ACC)  D6  2E                 LDAB OPCODE       B = OP CODE
40ACE)  81  03                 CMPA #3           IF NOT CLASS 4 OR 5 THEN
40AD0)  23  00                 BLS  VB02           BRANCH TO MEMORY REQUIRED ITEMS
40AD2)  D7  16                 STAB BYTOUT       BYTE OUT = OP CODE
40AD4)  BD  08  79             JSR  CALCWR       WRITE OP CODE
40AD7)  96  18                 LDAA CLASS
40AD9)  81  04                 CMPA #64          IF CLASS 4 VERB,
40ADB)  26  03                 BNE  *+5            GET
40ADD)  BD  00  00             JSR  RELATV         RELATIVE ADDRESS
40AE0)  39                     RTS               END OF CLASS 4 AND 5 VERBS
40AE1)  96  04          VB02   LDAA  MOD         MEMORY REQUIRED, A = VERB MODIFIER
40AD1)  0F
40AE3)  81  23                 CMPA #'#          IF MOD = '#,
40AE5)  26  00                 BNE  VB03           CALL
40AE7)  BD  00  00             JSR  IMED_M         IMEDIATE PROCESSING
40AEA)  39                     RTS
40AE8)  81  58          VB03   CMPA #'X          IF MOD = 'X,
40AE6)  04
40AED)  26  00                 BNE  VB20
40AEF)  CB  20                 ADDB #$20           ADJUST OP CODE
40AF1)  BD  00  00             JSR  INDXED         CALL INDEXED PROCESSING
40AF4)  39                     RTS
40AF5)  91  0A          VB20   CMPA BLNK         IF MOD = ' ,
40AEE)  06
40AF7)  26  00                 BNE  VB99
40AF9)  CB  10                 ADDB #$10           ADJUST OP CODE
40AFB)  BD  00  00             JSR  DFLT           CALL DEFAULT ADDRESSING PROCEDURE
40AFE)  39                     RTS
40AFF)  86  44          VB99   LDAA #'D          IF MOD WAS NOT FOUND ABOVE,
40AF8)  06
40B01)  BD  08  69             JSR  ERRS           SIGNAL ERROR,
40B04)  39                     RTS                 AND RETURN - END OF PROCESS VERB
                        **SKIP
                        *
                        *      ********************************************************************
                        *                      GET A RELATIVE ADDRESS SHIFT
                        *
40B05)  BD  00  00      RELATV JSR  GENADR      GET OPERAND
40ADE)  0B  05
40B08)  7D  00  3F             TST  UNRESL      IF NOT UNRESOLVED THEN
40B0B)  23  00                 BLS  RL10          CONTINUE
40B0D)  96  0D                 LDAA AD01+1        ELSE WRITE
40B0F)  97  16                 STAA BYTOUT        OUT A DUMMY
40B11)  BD  08  79             JSR  CALCWR        ARGUMENT
40B14)  39                     RTS              RETURN
40B15)  CE  00  2F      RL10   LDX  #PGMCNT       RELATIVE FROM OLD DATA NAME,
40B0C)  08
40B18)  96  0C                 LDAA AD01        LOAD RELATIVE VALUE
40B1A)  D6  0D                 LDAB AD01+1        WITH
40B1C)  E0  01                 SUBB 1,X           HOLD ADDR
40B1E)  A2  00                 SBCA X             MINUS
40B20)  C0  01                 SUBB #$01          PGM COUNT
40B22)  92  2C                 SBCA NULL          MINUS
40B24)  D7  16                 STAB BYTOUT        ONE
40B26)  BD  00  00             JSR  BRTST       CHECK FOR ANSWER > 127, OR <=128
40B29)  01                     NOP
40B2A)  01                     NOP
40B2B)  BD  08  79             JSR  CALCWR
40B2E)  39                     RTS              RETURN
                        *
                        *
                        *      ********************************************************************
                        *                      CHECK RELATIVE ADDRESS RANGE
                        *
40B2F)  91  2C          BRTST  CMPA NULL         IF HIGH BYTE
40B27)  08  2F
40B31)  27  06                 BEQ  *+8            NOT = 00 OR FF THEN
40B33)  81  FF                 CMPA #$FF           GO TO ERROR
40B35)  27  02                 BEQ  *+4
40B37)  20  00                 BRA  RLER
40B39)  98  16                 EORA BYTOUT      IF SIGNS OF BYTES ARE THE SAME THEN
40B3B)  2A  00                 BPL  BRT9          CONTINUE
40B3D)  86  47          RLER   LDAA #'G
40B3B)  04
40B3F)  BD  08  69             JSR  ERRS        WRITE OUT ERROR MSG
40B42)  DE  2F                 LDX  PGMCNT
40B44)  7D  00  1E             TST  FWD
40B47)  22  01                 BHI  *+3
40B49)  09                     DEX              CORRECT PGMCNT FOR BRANCH INSTRUCTIONS
40B4A)  DF  2F                 STX  PGMCNT
40B4C)  DF  83                 STX  OUTADR
40B4E)  39              BRT9   RTS              RETURN
40B3C)  11
```

```
                        **SKIP
                        *
                        *       ***************************************************************
                        *                   PROCESS IMMEDIATE ADDRESSING ITEMS (M = #)
                        *
40B4F)  96  18  IMED_M LDAA CLASS      CHECK CLASS
40AE8)  08  4F
40B51)  81  01         CMPA #$01         FOR 1
40B53)  27  06         BEQ  **8          BEING OK
40B55)  86  43         LDAA #'C          IF NOT,
40B57)  BD  08  69     JSR  ERRS         SIGNAL ERROR
40B5A)  39             RTS              AND RETURN
40B5B)  37             PSHB             SAVE B (OP CODE)
40B5C)  BD  00  00     JSR  GENADR      GENERATE ADDESS
40B5F)  33             PULB             RESTORE B
40B60)  D7  16         STAB BYTOUT      WRITE OUT
40B62)  BD  08  79     JSR  CALCWR       OP CODE
40B65)  96  05         LDAA NAME        IF OPERAND
40B67)  81  27         CMPA #''          DOES NOT START WITH ' THEN
40B69)  27  00         BEQ  IMD1         CONTINUE
40B6B)  BD  00  00     JSR  IMD2        ELSE PUT OUT BYTE(S)
40B6E)  39             RTS
40B6F)  96  06  IMD1   LDAA NAME+1      ELSE WRITE OUT
40B6A)  04
40B71)  97  16         STAA BYTOUT       LITERAL CHARACTER
40B73)  BD  08  79     JSR  CALCWR       AND
40B76)  39             RTS               RETURN
                        *
                        *
                        *       ***************************************************************
                        *                        WRITE OUT ONE OR TWO BYTE OPERANDS
                        *
40B77)  7D  00  1D  IMD2   TST  FLAGS      IF NOT TWO BYTE REGISTER INSTRUCTION THEN
40B6C)  08  77
40B7A)  23  00         BLS  IMD4          CONTINUE
40B7C)  96  0C         LDAA AD01         ELSE WRITE OUT
40B7E)  97  16         STAA BYTOUT        AD01
40B80)  BD  08  79     JSR  CALCWR
40B83)  96  0D  IMD4   LDAA AD01+1       WRITE OUT
40B7B)  07
40B85)  97  16         STAA BYTOUT        AD01+1
40B87)  BD  08  79     JSR  CALCWR
40B8A)  39             RTS               RETURN
                        **SKIP
                        *
                        *       ***************************************************************
                        *                        PROCESS INDEXED ADDRESSING (M = X)
                        *
40B8B)  37             INDXED PSHB        SAVE B (OP CODE)
40AF2)  0B  8B
40B8C)  BD  00  00     JSR  GENADR       GENERATE OFFSET
40B8F)  33             PULB              RESTORE B
40B90)  D7  16         STAB BYTOUT
40B92)  BD  08  79     JSR  CALCWR       WRITE OUT OP CODE
40B95)  96  0D         LDAA AD01+1       WRITE OUT OFFSET
40B97)  97  16         STAA BYTOUT
40B99)  BD  08  79     JSR  CALCWR
40B9C)  39             RTS
                        *
                        *
                        *       ***************************************************************
                        *
40B9D)  37             *                  PROCESS DEFAULT ADDRESSING FROM SYMB TABL (M = * )
                        *
40AFC)  0B  9D         DFLT   PSHB              SAVE B (OP CODE)
40B9E)  BD  00  00             JSR  GENADR       GET NAME AND ADDRESS IN OPERAND
40BA1)  33                     PULB             RESTORE B
40BA2)  7F  00  3C             CLR  TALY
40BA5)  7D  00  0C             TST  AD01        IF EXTENDED ADDR REQUIRED
40BA8)  26  00                 BNE  DFL0
40BAA)  7D  00  3F             TST  UNRESL       UNRESOLVED
40BAD)  22  00                 BHI  DFL0
40BAF)  96  18                 LDAA CLASS
40BB1)  81  03                 CMPA #3           OR CLASS THREE VARB THEN
40BB3)  27  00                 BEQ  DFL0
40BB5)  20  00                 BRA  DFL1
40BB7)  CB  20         DFL0    ADDB #$20         ADJUST OP CODE FOR EXTENDED
40BA9)  0D
40BAE)  08
40BB4)  02
40BB9)  7C  00  3C             INC  TALY
40BBC)  D7  16         DFL1    STAB BYTOUT       WRITE OUT
40BB6)  05
40BBE)  BD  08  79             JSR  CALCWR       OP CODE
40BC1)  7D  00  3C             TST  TALY
40BC4)  27  00                 BEQ  DFL4
40BC6)  D6  0C                 LDAB AD01         WRITE
40BC8)  D7  16                 STAB BYTOUT        OUT
40BCA)  BD  08  79             JSR  CALCWR        HIGH ORDER ADDRESS BYTE
40BCD)  D6  0D         DFL4    LDAB AD01+1       WRITE OUT
40BC5)  07
40BCF)  D7  16                 STAB BYTOUT        LOW ORDER
40BD1)  BD  08  79             JSR  CALCWR        ADDRESS BYTE
40BD4)  39                     RTS              END       ********
                        **SKIP
                        *
                        *       ***************************************************************
                        *               GENERATE AN ADDRESS FROM A NAME FROM THE INPUT STRING
                        *
40BD5)  7F  00  26  GENADR CLR  LAB_SW      CLEAR LABEL SWITCH
40B06)  0B  D5
40B5D)  08  D5
40B8D)  0B  D5
40B9F)  0B  D5
40BD8)  BD  00  00             JSR  GETADR       GET OPERAND ITEM
40BDB)  BD  00  00             JSR  GEN8         RESOLVE SYMBOLS
40BDE)  96  17         GEN1    LDAA CC           IF NEXT CHAR
40BE0)  81  28                 CMPA #$28          IS A '(' THEN
40BE2)  27  00                 BEQ  GEN2          GO TO OFFSET ADJUST
40BE4)  81  2D                 CMPA #$2D         IF NEXT CHAR IS A '-' THEN
40BE6)  27  00                 BEQ  GEN2          GO TO OFFSET ADJUST
40BE8)  39                     RTS               ELSE RETURN
40BE9)  97  09         GEN2    STAA SIGN         SAVE SIGN OF OFFSET
40BE3)  05
40BE7)  01
40BEB)  DE  0C                 LDX  AD01         SAVE
40BED)  DF  0E                 STX  SAVADR        ADDR-0?
40BEF)  BD  08  08             JSR  NX_CHR
40BF2)  BD  00  00             JSR  GETADR       GET OPERAND ITEM
40BF5)  BD  00  00             JSR  GEN8         RESOLVE SYMBOLS
40BF8)  96  09                 LDAA SIGN         IF SIGN
40BFA)  81  2B                 CMPA #$2B          WAS A '+' THEN
40BFC)  27  00                 BEQ  GEN3          GO TO ADDITION CORRECTION
40BFE)  96  0F                 LDAA SAVADR+1     SUBTRACTION CORRECTION:
40C00)  90  0D                 SUBA AD01+1        SUBTRACT OFFSET FROM SAVED ADDR
40C02)  97  0D                 STAA AD01+1
40C04)  96  0E                 LDAA SAVADR
40C06)  92  0C                 SBCA AD01
40C08)  97  0C                 STAA AD01          *
40C0A)  20  D2                 BRA  GEN1         TRY ANOTHER TERM
40C0C)  96  0F         GEN3    LDAA SAVADR+1     ADDITION CORRECTION:
40BFD)  0E
40C0E)  9B  0D                 ADDA AD01+1        ADD OFFSET TO SAVED ADDR
40C10)  97  0D                 STAA AD01+1
40C12)  96  0E                 LDAA SAVADR
40C14)  99  0C                 ADCA AD01
40C16)  97  0C                 STAA AD01
40C18)  20  C4                 BRA  GE .         TRY ANOTHER TERM
```

```
                        **SKIP
                        *
                        *       ************************************************************
                        *                          RESOLVE SYMBOL ADDRESSES
                        *
(0C1A)  7D 00 03        GEN8    TST  ERORSW     IF NO ERRORS,
(08BC)  0C 1A
(08F6)  0C 1A
(0C1D)  22 00                   BHI  GEN9
(0C1F)  7D 00 1F                TST  HEX          NOT HEX,
(0C22)  22 00                   BHI  GEN9
(0C24)  7D 00 10                TST  ALPHA        AND ALPHA THEN
(0C27)  23 00                   BLS  GEN9
(0C29)  BD 00 00                JSR  SYMB_S       CALL SYMBOL RESOLVE
(0C2C)  39              GEN9    RTS
(0C1E)  0D
(0C23)  08
(0C28)  03
                        **SKIP
                        *
                        *       ************************************************************
                        *                          GET ADDRESS FOR ITEM IN OPERAND
                        *
(0C2D)  7C 00 10        GETADR  INC  ALPHA      SET ALPHA AND HEX
(08D9)  0C 2D
(08F3)  0C 2D
(0C30)  7C 00 1F                INC  HEX          SWITCHES ON
(0C33)  7F 00 0C                CLR  AD01       CLEAR FINAL NUMBER
(0C36)  7F 00 0D                CLR  AD01+1
(0C39)  96 17                   LDAA CC
(0C3B)  81 24                   CMPA #'$        IF ADDR NOT HEX
(0C3D)  26 00                   BNE  GAD1         THEN CONTINUE
(0C3F)  BD 08 08                JSR  NX_CHR     ELSE DROP $ CHAR
(0C42)  BD 08 3C                JSR  LD_ITM       LOAD ADDR
(0C45)  BD 00 00                JSR  CVTHEX       CONVERT TO HEX
(0C48)  39                      RTS               AND RETURN
(0C49)  7F 00 1F        GAD1    CLR  HEX        TURN HEX OFF
(0C3E)  0A
(0C4C)  81 26                   CMPA #'&        TEST NEXT CHAR FOR &
(0C4E)  26 03                   BNE  *+5        IF NOT CONTINUE
(0C50)  BD 08 08                JSR  NX_CHR     ELSE DROP IT (NUMERIC IS ASSUMED DECIMAL)
(0C53)  BD 08 3C                JSR  LD_ITM     LOAD NEXT ITEM INTO NAME
(0C56)  CE 00 05                LDX  #NAME      TEST FIRST CHAR
(0C59)  A6 00                   LDAA X            IF NOT NUMERIC
(0C5B)  91 0A                   CMPA BLNK         BUT NOT NULL OPERAND
(0C5D)  27 00                   BEQ  GAD2
(0C5F)  81 30                   CMPA #'0
(0C61)  2B 00                   BMI  GAD5
(0C63)  81 39                   CMPA #'9
(0C65)  22 00                   BHI  GAD5         THEN GO TO NON NUMERIC CODE
                        *  NUMERIC OPERAND
(0C67)  91 0A           GAD2    CMPA BLNK       IF AT END OF ITEM
(0C5E)  08
(0C69)  27 00                   BEQ  GAD4         GO TO END OF NUMERIC
(0C68)  8C 00 09                CPX  #NAME+4
(0C6E)  27 00                   BEQ  GAD4
(0C70)  85 30                   BITA #$30       CHECK FOR VALID
(0C72)  26 06                   BNE  *+8          CHARACTERS
(0C74)  86 4A                   LDAA #'J
(0C76)  BD 08 69                JSR  ERRS
(0C79)  39                      RTS
(0C7A)  84 0F                   ANDA #$0F       DROP INPUT'S LEFT 4 BITS
(0C7C)  A7 00                   STAA X            AND REPLACE INTO NAME
(0C7E)  C6 09                   LDAB #9         MULTIPLY ADD1 BY 10
(0C80)  D7 3C                   STAB TALY
(0C82)  D6 0D                   LDAB AD01+1
(0C84)  96 0C                   LDAA AD01
(0C86)  DB 0D           GAD3    ADDB AD01+1
(0C88)  99 0C                   ADCA AD01
(0C8A)  7A 00 3C                DEC  TALY
(0C8D)  26 F7                   BNE  GAD3
                        **SKIP
                        *  GETADR CONT
                        *
(0CAF)  EB 00                   ADDB X          ADD NEW NUMBER TO AD01
(0C91)  99 2C                   ADCA NULL
(0C93)  D7 0D                   STAB AD01+1
(0C95)  97 0C                   STAA AD01
(0C97)  08                      INX             GO TO
(0C98)  A6 00                   LDAA X            NEXT
(0C9A)  20 CB                   BRA  GAD2         CHAR
(0C9C)  7F 00 10        GAD4    CLR  ALPHA      TURN OFF ALPHA
(0C9F)  39                      RTS             RETURN
(0CA0)  CE 00 40        GAD5    LDX  #XUPND     COMPARE NAME WITH 'X   '
(0C62)  3D
(0C66)  39
(0CA3)  DF 32                   STX  STR1
(0CA5)  CE 00 05                LDX  #NAME
(0CA8)  DF 34                   STX  STR2
(0CAA)  BD 0A A5                JSR  STR_CM
(0CAD)  7D 00 11                TST  ANSW       IF NOT EQUAL THEN
(0CB0)  23 00                   BLS  GAD6         CONTINUE
(0CB2)  DE 2C                   LDX  NULL       ELSE
(0CB4)  DF 05                   STX  NAME         CLEAR NAME
(0CB6)  DF 07                   STX  NAME+2
(0CB8)  7F 00 10                CLR  ALPHA      TURN OFF ALPHA SWITCH
(0CBB)  39                      RTS             RETURN
(0CBC)  CE 00 12        GAD6    LDX  #ASTK      COMPARE NAME WITH '*   '
(0CBF)  0A
(0CC1)  DF 32                   STX  STR1
(0CC3)  CE 00 05                LDX  #NAME
(0CC4)  DF 34                   STX  STR2
(0CC6)  BD 0A A5                JSR  STR_CM
(0CC9)  7D 00 11                TST  ANSW       IF NOT EQUAL THEN
(0CCC)  23 00                   BLS  GAD9         CONTINUE
(0CCE)  DE 2F                   LDX  PGMCNT     ELSE GET PGMCNT
(0CD0)  96 18                   LDAA CLASS        IF VERB CLASS
(0CD2)  81 04                   CMPA #$4            NOT = 4 THEN
(0CD4)  26 01                   BNE  *+3            CONTINUE
(0CD6)  09                      DEX               ELSE DECREMENT PGMCNT
(0CD7)  DF 0C                   STX  AD01         OUTPUT ADDRESS
(0CD9)  7F 00 10                CLR  ALPHA        SET ALPHA OFF
(0CDC)  39              GAD9    RTS             RETURN
(0CDD)  0E
                        **SKIP
                        *
                        *
                        *       ************************************************************
                        *                  CONVERT NAME TO HEX IN AD01, AD01+1
                        *
(0CDD)  CE 00 05        CVTHEX  LDX  #NAME      START AT NAME
(0C46)  0C DD
(0CE0)  A6 00           CVT1    LDAA X
(0CE2)  91 0A                   CMPA BLNK       IF CHAR = BLANK THEN
(0CE4)  27 00                   BEQ  CVT9         GO TO END
(0CE6)  8C 00 09                CPX  #NAME+4    IF AT END OF INPUT THEN
(0CE9)  27 00                   BEQ  CVT9         GO TO END
(0CEB)  81 30                   CMPA #'0        IF CHAR < 0
(0CED)  2B 00                   BMI  CVT3         OR
(0CEF)  81 46                   CMPA #'F          > F
(0CF1)  22 00                   BHI  CVT3         THEN SIGNAL ERROR
(0CF3)  20 06                   BRA  *+8        ELSE CONTINUE
(0CF5)  86 4A           CVT3    LDAA #'J
(0CEE)  06
(0CF2)  02
(0CF7)  BD 08 69                JSR  ERRS
(0CFA)  39                      RTS
(0CFB)  81 41                   CMPA #'A        IF CHAR < A THEN
(0CFD)  2B 02                   BMI  *+4          CONTINUE
(0CFF)  8B 09                   ADDA #$09       ELSE ADD 9 TO LOW NIBBLE
(0001)  84 0F                   ANDA #$0F       DROP HIGH NIBBLE
(0003)  A7 00                   STAA X          RESTORE CHARACTER TO NAME
(0005)  96 0C                   LDAA AD01       MULTIPLY
(0007)  D6 0D                   LDAB AD01+1       CURRENT VALUE
(0009)  58                      ASLB              BY 16
(000A)  49                      ROLA
(000B)  58                      ASLB
(000C)  49                      ROLA
(000D)  58                      ASLB
(000E)  49                      ROLA
(000F)  58                      ASLB
(0010)  49                      ROLA
(0011)  EB 00                   ADDB X          ADD VALUE OF THIS CHARACTER
(0013)  D7 0D                   STAB AD01+1     RESTORE TOTAL VALUE
(0015)  97 0C                   STAA AD01
(0017)  08                      INX             GO TO NEXT CHAR
(0018)  20 C6                   BRA  CVT1
(001A)  39              CVT9    RTS             RETURN
(0CE5)  34
(0CEA)  2F
```

```
                        **SKIP
                        *
                        *       ********************************************************************
                        *                       PROCESS A NAME AGAINST THE SYMBOL TABLE
                        *                       IF NOT THERE, SET IT UP.  RETURN ADDRESS
                        *
(0D1A) 96 05            SYMB_S LDAA NAMF        IF LABEL STARTS WITH A ' (QUOTE) THEN
(0957) 0D 1A
(0C2A) 0D 14
(0D1D) 81 27                    CMPA #''          RETURN
(0D1F) 26 01                    BNE  **3
(0D21) 39                       RTS
(0D22) 81 40                    CMPA #$40       IF LABEL STARTS WITH A CODE LESS THAN ALPHA THEN
(0D24) 22 06                    BHI  **8          SIGNAL ERROR
(0D26) 86 4B                    LDAA #'K
(0D28) BD 0B 69                 JSR  ERRS
(0D2B) 39                       RTS
(0D2C) CE 02 DF                 LDX  #STBL      SET INDEX TO TABLE ENTRY POINT
(0D2F) C6 01                    LDAB #$01       INITIALIZE B (LOOP COUNTER)
(0D31) C1 C9            SM01    CMPB #201       MAIN LOOP - COMPARE B WITH TABLE SIZE + 1
(0D33) 26 06                    BNE  **8          AND IF < , CONTINUE
(0D35) 86 45                    LDAA #'E        ELSE
(0D37) BD 0B 69                 JSR  ERRS       PUT OUT TABLE FULL ERROR (E)
(0D3A) 39                       RTS               AND RETURN
(0D3B) DF 34                    STX  STR2       LOAD SYMBOL INTO STRING 2
(0D3D) DF 3A                    STX  TBLS       SAVE SYMBOL
(0D3F) CE 00 05                 LDX  #NAME      LOAD NAME
(0D42) DF 32                    STX  STR1         INTO STRING 1
(0D44) BD 0A A5                 JSR  STR_CM     CALL STRING COMPARE
(0D47) 7D 00 11                 TST  ANSW       IF NO HIT THEN
(0D4A) 23 00                    BLS  NOHIT        GO TO NO-HIT LOOP
(0D4C) DE 3A                    LDX  TBLS       RESTORE INDEX TO SYMB
(0D4E) 7D 00 26                 TST  LAB_SW     IF LABEL,
(0D51) 22 00                    BHI  SM10         BRANCH TO LABEL PROC
(0D53) EE 04                    LDX  4,X        LOAD ADDRESS -- (NOT NEW, NOT LABEL)
(0D55) DF 0C                    STX  ADD1         INTO HOLD ADDRESS
(0D57) 9C 2C                    CPX  NULL       IF ADDER = 0 SET
(0D59) 26 03                    BNE  **5          UNRESOLVED
(0D5B) BD 00 00                 JSR  S_FWD        ADDR,
(0D5E) 39                       RTS             RETURN
                        **SKIP
                        *
                        ***** SYMB_S CONT
                        *
(0D5F) EE 04            SM10    LDX  4,X        LABEL PROC
(0D52) 0C
(0D61) 9C 2C                    CPX  NULL       IF ADDR NOT = 0 THEN
(0D63) 27 06                    BEQ  **8          GO TO ERROR ROUTINE
(0D65) 86 46                    LDAA #'F
(0D67) BD 0B 69                 JSR  ERRS
(0D6A) 39                       RTS
(0D6B) BD 00 00                 JSR  SMBLOD     LOAD SYMBOL TABLE
(0D6E) 7C 00 1E                 INC  FWD        SIGNAL ADDRESSES TO RESOLVE
(0D71) DE 3A                    LDX  TBLS
(0D73) EE 04                    LDX  4,X        LOAD ADDRESS
(0D75) DF 0C                    STX  ADD1         INTO HOLD ADDRESS
(0D77) 39                       RTS             RETURN
(0D78) DE 3A            NOHIT   LDX  TBLS       TEST FOR EMPTY SLOT IN TABLE
(0D4B) 2C
(0D7A) 6D 00                    TST  X
(0D7C) 26 00                    BNE  SM90
(0D7E) BD 00 00                 JSR  SMBLOD     ELSE LOAD ITEM
(0D81) 7D 00 26                 TST  LAB_SW     IF NOT
(0D84) 22 03                    BHI  **5          LABEL
(0D86) BD 00 00                 JSR  S_FWD        THEN SET UNRESOLVED RWD ADDRESS
(0D89) 39                       RTS             RETURN
(0D8A) DE 3A            SM90    LDX  TBLS       TRY NEXT
(0D7D) 0C
(0D8C) 08                       INX               ITEM,
(0D8D) 08                       INX               INCR IX BY 6
(0D8E) 08                       INX
(0D8F) 08                       INX
(0D90) 08                       INX
(0D91) 08                       INX
(0D92) 5C                       INCB              AND B BY 1
(0D93) 20 9C                    BRA  SM01       GO TO MAIN SYMB LOOP

                        **SKIP
                        *
                        *       ********************************************************************
                        *                       LOAD A NAME INTO THE SYMBOL TABLE
                        *
(0D95) DE 3A            SMBLOD  LDX  TBLS
(0D6C) 0D 95
(0D7F) 0D 95
(0D97) 9F 3A                    STS  SVST       SAVE STACK PTR
(0D99) 9E 05                    LDS  NAME       LOAD
(0D9B) AF 00                    STS  X            NAME
(0D9D) 9E 07                    LDS  NAME+2
(0D9F) AF 02                    STS  2,X
(0DA1) 7D 00 26                 TST  LAB_SW     IF NOT LABEL,
(0DA4) 23 00                    BLS  SMHS         BRANCH TO ENDING
(0DA6) 9E 2F                    LDS  PGMCNT     LOAD
(0DA8) AF 04                    STS  4,X          ADDR
(0DAA) DF 27                    STX  LSTLAB     SAVE LABEL ADDRESS
(0DAC) EE 04            SMHS    LDX  4,X        STORE ADDR
(0DA5) 06
(0DAE) DF 0C                    STX  ADD1         IN HOLD ADDR
(0DB0) 9E 3A                    LDS  SVST       RESTORE STACK PTR
(0DB2) 39                       RTS             RETURN
                        **SKIP
                        *
                        *       ********************************************************************
                        *                       SET FOREWARD UNRESOLVED ADDRESS
                        *
(0DB3) 86 01            S_FWD   LDAA #$01         SET A (COUNTER) TO 1
(0D5C) 0D B3
(0DB7) 0D B3
(0DB5) 7F 00 3C                 CLR  TALY         AND TALLY TO 0
(0DB8) CE 02 61                 LDX  #FTBL        SET IX TO FWD BRANCH TABLE
(0DBB) 81 19            SFW1    CMPA #&25       IF A < 25
(0DBD) 2F 06                    BLE  **8          THEN CONTINUE
(0DBF) 86 49                    LDAA #'I        ELSE
(0DC1) BD 0B 69                 JSR  ERRS         WRITE OUT TABLE ERROR AND
(0DC4) 39                       RTS               RETURN
(0DC5) 81 19                    CMPA #&25       IF A < 25
(0DC7) 26 00                    BNE  SFW3         THEN CONTINUE
(0DC9) 86 48                    LDAA #'H        ELSE
(0DCB) BD 0B 69                 JSR  ERRS         WARN OF NEAR TABLE ERROR,
(0DCE) 7F 00 03                 CLR  ERORSW         CLEAR ERROR SWITCH
(0DD1) E6 00            SFW3    LDAB X          TEST FOR EMPTY SLOT
(0DC8) 08
(0DD3) 26 00                    BNE  SFW5         AND IF NOT, CONTINUE
(0DD5) E6 01                    LDAB 1,X        ELSE
(0DD7) 26 00                    BNE  SFW5         GO TO SET ADDRESS
(0DD9) 20 00                    BRA  SFW7
(0DDB) 4C               SFW5    INCA              ELSE
(0DD4) 06
(0DD8) 02
(0DDC) 08                       INX               INCTREMENT
(0DDD) 08                       INX               A BY 1,
(0DDE) 08                       INX               IX AND
(0DDF) 08                       INX               TALY
(0DE0) 08                       INX               BY
(0DE1) D6 3C                    LDAB TALY         5
(0DE3) CB 05                    ADDB #$05         AND
(0DE5) D7 3C                    STAB TALY         GO TO
(0DE7) 20 D2                    BRA  SFW1         MAIN LOOP
```

```
                        **SKIP
                        *
                        ***** S_FWD CONT.
                        *
(0DE9) 96 04            SFW7    LDAA MOD        IF INDEXED ANDRESSING THEN
(0DDA) 0E               
(0DEB) 81 58            
(0DED) 26 02            
(0DEF) 6A 04            
(0DF1) 81 23            
(0DF3) 26 00            
(0DF5) 7D 00 1D         
(0DF8) 22 00            
(0DFA) 6A 04            
(0DFC) 96 18            SFW8    LDAA CLASS      IF RELATIVE ADDRESSING THEN
(0DF4) 07               
(0DF9) 02               
(0DFE) 81 04                    CMPA #4           SET TYPE AS +1
(0E00) 26 02                    BNE  *+4
(0E02) 6C 04                    INC  4,X
(0E04) 9F 3A                    STS  SVST       SAVE STACK POINTER
(0E06) 9F 3A                    LDS  TBLS       LOAD ADDRESS OF SYMB IN SYMB TABL
(0E08) AF 00                    STS  X            INTO FWD ANDRESS TABLE
(0E0A) 9E 2F                    LDS  PGMCNT       INTO
(0E0C) 31                       INS
(0E0D) AF 02                    STS  2,X          ADDR TABL
(0E0F) 9E 3A                    LDS  SVST       RESTORE STACK POINTER
(0E11) 7C 00 3F                 INC  UNRESL
(0E14) 39                       RTS             RETURN
                        **SKIP
                        *  ************************************************************
                        *                  RESOLVE FOREWARD ADDRESS REFERENCES
                        *
(0E15) 7D 00 1E         R_FWD   TST  FWD        IF NO FORWARD REFERENCES TO RESOLVE
(099E) 0E 15            
(0E18) 22 01                    BHI  *+3          RETURN, ELSE
(0E1A) 39                       RTS
(0E1B) 86 01                    LDAA #$01       INITIALIZE A(COUNT)
(0E1D) CE 02 61                 LDX  #FTBL        AND IX (TABLE POSITION)
(0E20) DF 22                    STX  IXS2
(0E22) 7D 00 03         RF01    TST  ERORSW     MAIN LOOP
(0E25) 22 00                    BHI  RF02       IF ERROR, RETURN
(0E27) 81 19                    CMPA #$25
(0E29) 2F 00                    BLE  RF03       IF COUNTER < OR = 25, CONTINUE
(0E2B) 7F 00 1E                 CLR  FWD        CLEAR FWD SWITCH
(0E2E) 39               RF02    RTS             ELSE RETURN - END OF SEARCH
(0E26) 07               
(0E2F) 36               RF03    PSHA            SAVE A
(0E2A) 04               
(0E30) 7D 00 1C                 TST  EOJFL      IF EOJ,
(0E33) 23 00                    BLS  RF05
(0E35) BD 00 00                 JSR  RF70       CALL DUMP-UNRESOLVED
(0E38) 20 00                    BRA  RF80
(0E3A) DE 22            RF05    LDX  IXS2       COMPARE FWD ADDR OFFSET
(0E34) 05               
(0E3C) A6 00                    LDAA X          IF EITHER ONES NOT MATCH
(0E3E) E6 01                    LDAB 1,X          THEN GO TO INCREMENT LOOP
(0E40) 91 27                    CMPA LSTLAB
(0E42) 26 00                    BNE  RF80
(0E44) D1 28                    CMPB LSTLAB+1
(0E46) 26 00                    BNE  RF80
(0E48) DE 2F                    LDX  PGMCNT     ELSE FOUND HIT TO RESOLVE
(0E4A) DF 36                    STX  SVPC       SAVE PGM_CNT
(0E4C) BD 08 82                 JSR  WRITE      WRITE OUT CURRENT OUTPUT
(0E4F) DE 22                    LDX  IXS2       LOAD PGM-CNT
(0E51) EE 02                    LDX  2,X          WITH
(0E53) DF 2F                    STX  PGMCNT       PGM-CNT IN SYMBOL TABLE
(0E55) DE 22                    LDX  IXS2       TEST TYPE INDICATOR
(0E57) 6D 04                    TST  4,X
(0E59) 2B 00                    BMI  RF10         IF = -1 HANDLE ONE BYTE OUT
(0E5B) 26 00                    BNE  RF20         IF = +1 HANDLE RELATIVE ITEMS
(0E5D) DE 27                    LDX  LSTLAB     WRITE OUT
(0E5F) E6 04                    LDAB 4,X          ADDRESS
(0E61) D7 16                    STAB BYTOUT       IN SYMBOL TABLE
(0E63) BD 08 79                 JSR  CALCWR       FOR THIS SYMBOL
(0E66) DE 27            RF10    LDX  LSTLAB
(0E5A) 08               
(0E68) E6 05                    LDAB 5,X
(0E6A) D7 16                    STAB BYTOUT
(0E6C) BD 08 79                 JSR  CALCWR
(0E6F) 20 00                    BRA  RF30       GO TO RESTORE
```

```
                        **SKIP
                        *
                        ***** R_FWD CONT
                        *
(0E71) DE 27            RF20    LDX  LSTLAB     HANDLE RELATIVE ITEMS
(0E5C) 14               
(0E73) A6 04                    LDAA 4,X        LOAD RELATIVE ADDR #ITM
(0E75) E6 05                    LDAB 5,X          SYMBOL TABLE ADDR
(0E77) CE 00 2F                 LDX  #PGMCNT      MINUS
(0E7A) E0 01                    SUBB 1,X          PGM_CNT AT PREVIOUS REFERENCE
(0E7C) A2 00                    SBCA X
(0E7E) DE 2F                    LDX  PGMCNT     ADJUST PGM_CNT
(0E80) 09                       DEX               FOR RELATIVE
(0E81) DF 2F                    STX  PGMCNT       ITEMS
(0E83) D7 16                    STAB BYTOUT
(0E85) BD 0A 2F                 JSR  BRTST      TEST RANGE
(0E88) BD 0A 79                 JSR  CALCWR       RELATIVE ADDR
(0E8B) DE 36            RF30    LDX  SVPC       RESTORE
(0E70) 1A               
(0E8D) DF 2F                    STX  PGMCNT       PGM-CNT
(0E8F) DE 22                    LDX  IXS2       CLEAR
(0E91) 6F 00                    CLR  X            ITEMS
(0E93) 6F 01                    CLR  1,X          IN FWD  (POSITION IN SYMBOL TABLE)
(0E95) 6F 02                    CLR  2,X          ADDR    (PGM
(0E97) 6F 03                    CLR  3,X          TABL     CNT)
(0E99) 6F 04                    CLR  4,X                   (TYPE INDICATOR)
(0E9B) DE 22            RF80    LDX  IXS2       INCREMENT LOOP
(0E43) 57               
(0E39) 61               
(0E47) 53               
(0E9D) 08                       INX               INCREMENT
(0E9E) 08                       INX               IX
(0E9F) 08                       INX               BY
(0EA0) 08                       INX               5
(0EA1) 08                       INX               AND
(0EA2) DF 22                    STX  IXS2         SAVE IT
(0EA4) 32                       PULA            RESTORE A (COUNTER)
(0EA5) 4C                       INCA            INCREMENT A
(0EA6) 7E 0E 22                 JMP  RF01       GO TO MAIN LOOP
                        **SKIP
                        *
                        *  ************************************************************
                        *                        PRINT OUT UNRESOLVED ITEM
                        *
(0EA9) DE 22            RF70    LDX  IXS2
(0EA6) 0E A9            
(0EAB) E6 00                    LDAB X          IF POSITION NOT EMPTY
(0EAD) 26 00                    BNE  RF75         THEN
(0EAF) E6 01                    LDAB 1,X          GO TO DUMP ITEM FOR
(0EB1) 26 00                    BNE  RF75         BOTH BYTES
(0EB3) 39                       RTS             ELSE RETURN
(0EB4) EE 00            RF75    LDX  X          DUMP ITEM TO
(0EAE) 05               
(0EB2) 01               
(0EB6) 86 04                    LDAA #4           TERMINAL LOOP
(0EB8) 97 3C                    STAA TALY
(0EBA) A6 00            RF76    LDAA X
(0EBC) BD 07 90                 JSR  CHROUT
(0EBF) 08                       INX
(0EC0) 7A 00 3C                 DEC  TALY
(0EC3) 26 F5                    BNE  RF76
(0EC5) BD 07 9E                 JSR  BLKOUT     WRITE OUT BLANK
(0EC8) DE 22                    LDX  IXS2         AND UNRESOLVED ADDRESS
(0ECA) A6 02                    LDAA 2,X
(0ECC) BD 07 96                 JSR  HEXOUT
(0ECF) A6 03                    LDAA 3,X
(0ED1) BD 07 96                 JSR  HEXOUT
(0ED4) CE 00 93                 LDX  #CRLF
(0ED7) BD 07 A1                 JSR  STROUT     GO TO NEXT LINE
(0EDA) 39                       RTS
```

```
                        **SKIP
                        *
                        *       ************************************************************
                        *                       START PSEUDO OP PROCESSING
                        *
(0ED8)  CE 00 00        PSEUDO  LDX  #PSTBL     LOCATE PSEUDO OP
(0984)  0E 08
(0EDE)  5D              PSU1    TSTB              USING OP CODE (B) AND
(0EDF)  27 00                   BEQ  PSU3         JUMP TABLE (PSTBL)
(0EE1)  08                      INX
(0EE2)  08                      INX
(0EE3)  5A                      DECB
(0EE4)  20 F8                   BRA  PSU1
(0EE6)  BD 0A 30        PSU3    JSR  NX_ITM      LOAD START OF OPERAND
(0EE9)  05
(0EEA)  AD 00                   JSR  X           CALL OPERAND PROCESSING
(0EEC)  39                      RTS              RETURN
                        *
                        *   ASSEMBLER DIRECTIVES
                        *   EQU
                        *
(0EED)  BD 0A 05        EQU     JSR  GENADR      GET OPERAND
(0EEF)  9F 3A                   STS  SVST        SAVE STACK PTR
(0EF1)  9E 0C                   LDS  ADD1        MOVE OPERAND
(0EF3)  DE 27                   LDX  LSTLAB        INTO
(0EF5)  AF 04                   STX  4,X           SYMBOL TABLE
(0EF7)  9F B3                   STS  OUTADR
(0EF9)  9E 3A                   LDS  SVST        RESTORE STACK PTR
(0EFB)  39                      RTS              RETURN
                        *
                        *   FCH = FDB
                        *
(0EFC)  7F 00 1D        FCH     CLR  FLAGS       FOR SINGLE BYTE
(0EFF)  20 00                   BRA  FB01          CLEAR FLAGS
(0F01)  7F 00 1D        FDB     CLR  FLAGS       FOR DOUBLE BYTE
(0F04)  7C 00 1D                INC  FLAGS         SET FLAGS
(0F07)  BD 0A 05        FB01    JSR  GENADR      OBTAIN OPERAND AND USE OUTPUT FOR
(0F0A)  BD 0A 77                JSR  IMD2          IMEDIATE ADDRESSING TO PUT IT OUT
(0F0D)  96 17                   LDAA CC          IF NEXT CHATR
(0F0F)  81 2C                   CMPA #$2C          IS A COMMA
(0F11)  27 01                   BEQ  *+3           CONTINUE
(0F13)  39                      RTS              ELSE RETURN
(0F14)  BD 0A 08                JSR  NX_CHR      GET NEXT CHAR
(0F17)  20 EE                   BRA  FB01        PUT OUT NEXT CHAR
                        **SKIP
                        *
                        *   FCC
                        *
(0F19)  BD 0A 05        FCC     JSR  GENADR      GET CHAR
(0F1C)  96 0D                   LDAA ADD1+1        COUNT
(0F1E)  97 3C                   STAA TALY
(0F20)  26 01           FCC1    BNE  *+3           CONTINUE
(0F22)  39                      RTS              ELSE RETURN
(0F23)  BD 0A 08                JSR  NX_CHR      PUT OUT
(0F26)  97 16                   STAA BYTOUT        NEXT CHAR
(0F28)  BD 0A 79                JSR  CALCWR
(0F2B)  7A 00 3C                DEC  TALY        DECREMENT COUNT
(0F2E)  20 F0                   BRA  FCC1        GO TO NEXT CHAR
                        *
                        *   ORG
                        *
(0F30)  BD 0A 05        ORG     JSR  GENADR      PUT OPERAND
(0F33)  DE 0C                   LDX  ADD1          INTO
(0F35)  DF 2F                   STX  PGMCNT        PGMCNT
(0F37)  DF B3                   STX  OUTADR
(0F39)  39                      RTS
                        *
                        *   RMB
                        *
(0F3A)  DE 2F           RMB     LDX  PGMCNT
(0F3C)  DF B3                   STX  OUTADR
(0F3E)  BD 0A 05                JSR  GENADR      ADD OPRAND
(0F41)  96 30                   LDAA PGMCNT+1      TO
(0F43)  D6 2F                   LDAB PGMCNT        PGMCNT
(0F45)  9B 00                   ADDA ADD1+1
(0F47)  09 0C                   ADCB ADD1
(0F49)  97 30                   STAA PGMCNT+1
(0F4B)  D7 2F                   STAB PGMCNT
(0F4D)  39                      RTS
                        *
                        *   END OF ASSEMBLER DIRECTIVES
                        *
(0F4E)  20 9C           PSTBL   BRA  EQU         TABLE OF STARTING ADDRESSES FOR PSEUDO ROUTINES
(0EDC)  0F 4E
(0F50)  20 AA                   BRA  FCB
(0F52)  20 C5                   BRA  FCC
(0F54)  20 AB                   BRA  FDB
(0F56)  20 DA                   BRA  ORG
(0F58)  20 E0                   BRA  RMB
                        **SKIP
                        *
                        *
                        *       ************************************************************
                        *                       END OF ASSEMBLY
                        *
(0F5A)  CE 00 78        EOJ     LDX  #MSG1       WRITE OUT
(07D5)  0F 5A
(0F5D)  BD 07 A1                JSR  STROUT        UNRESOLVED TITLE
(0F60)  CF 00 6E                LDX  #LINFED     WRITE OUT
(0F63)  BD 07 A1                JSR  STROUT        BLANK LINE
(0F66)  7C 00 1C                INC  EOJFL       SET END OF JOB FLAG
(0F69)  86 01                   LDAA #1            AND FWD FLAG
(0F6B)  97 1E                   STAA FWD
(0F6D)  BD 0F 15                JSR  R_FWD       SEARCH FOR UNRESOLVED ITEMS
(0F70)  BD 00 00                JSR  SMDP
(0F73)  86 0A                   LDAA #$0A
(0F75)  BD 07 90                JSR  CHROUT
(0F78)  39                      RTS              RETURN
                        *
                        *
                        *       ************************************************************
                        *                       DUMP SYMBOL TABLE
                        *
(0F79)  CE 00 98        SMDP    LDX  #MSG2       WRITE OUT
(0F7B)  0F 79
(0F7C)  BD 07 A1                JSR  STROUT        TABLE DUMP HEADING
(0F7F)  BD 07 93                JSR  CHRIN       RECIEVE INSTRUCTION FOR SORT
(0F82)  81 59                   CMPA #'Y         IF NOT TO SORT THEN
(0F84)  26 03                   BNE  *+5           CONTINUE, ELSE
(0F86)  BD 00 00                JSR  SORT        SORT SYMBOL TABLE
(0F89)  CE 02 0F                LDX  #STBL       POSITION TO START OF TABLE
(0F8C)  86 05           SMD2    LDAA #5          LOAD TALY WITH 5
(0F8E)  97 3C                   STAA TALY
(0F90)  0F 20                   STX  IAS1        ELSE
(0F92)  CE 00 93                LDX  #CRLF         PUT OUT CARRIAGE RETURN/LINE FEED
(0F95)  BD 07 A1                JSR  STROUT
(0F98)  0E 20                   LDX  IAS1
(0F9A)  86 04           SMD4    LDAA #4          AND COUNT WITH 4
(0F9C)  97 1A                   STAA COUNT
(0F9E)  A6 00           SMD6    LDAA X           WRITE OUT FIRST 4 CHARACTERS
(0FA0)  27 00                   BEQ  SMD9          IF NULL CHARACTER, GO TO END
(0FA2)  BD 07 90                JSR  CHROUT
(0FA5)  08                      INX
(0FA6)  7A 00 1A                DEC  COUNT
(0FA9)  26 F3                   BNE  SMD6
(0FAB)  BD 07 9E                JSR  BLKOUT      WRITE OUT A BLANK
(0FAE)  A6 00                   LDAA X
(0FB0)  BD 07 96                JSR  HEXOUT      WRITE OUT FIRST HALF OF ADDRESS
(0FB3)  08                      INX
(0FB4)  A6 00                   LDAA X
(0FB6)  BD 07 96                JSR  HEXOUT      WRITE OUT SECOND HALF OF ADDRESS
(0FB9)  08                      INX
(0FBA)  BD 07 9E                JSR  BLKOUT      WRITE OUT TWO BLANKS
(0FBD)  BD 07 9E                JSR  BLKOUT
(0FC0)  7A 00 3C                DEC  TALY        IF 5 SYMBOLS HAVE NOT BEEN PRINTED,
(0FC3)  26 05                   BNE  SMD4          CONTINUE
(0FC5)  20 C5                   BRA  SMD2          GO TO NEXT GROUP OF 5 SYMBOLS
(0FC7)  39              SMD9    RTS              AT END, RETURN
(0FA1)  25
```

```
                        **SKIP
                        *
                        *       ********************************************************************
                        *                                SORT SYMBOL TABLE
                        *
(0FCA)  CE  02  0F      SORT    LDX  #STBL      GET START OF TABLE
(0FCD)  0F  CA          
(0FCE)  7F  00  11              CLR  ANSW       CLEAR SWAP SWITCH
(0FD0)  20  00                  BRA  SORT4
(0FD2)  C6  06          SORT1   LDAB #6         LOAD LENGTH OF ENTRY
(0FD4)  A6  00          SORT2   LDAA X          COMPARE CHAR OF ENTRY WITH NEXT ENTRY
(0FD6)  A1  06                  CMPA 6,X
(0FD8)  2D  00                  BLT  SORT3        IF < GO TO CHECK NEXT SYM
(0FDA)  26  08                  BNE  SWAP         IF > GO TO SWAP SYMBOLS
(0FDC)  08                      INX             ELSE GO TO NEXT CHAR (LAST CANT BE EQUAL)
(0FDD)  5A                      DECB            B = PORTION OF SYMBOLS NOT EQUAL
(0FDE)  20  F4                  BRA  SORT2      GO TO NEXT CHARACTER
(0FE0)  08              SORT3   INX             SKIP TO
(0FE1)  06              
(0FE2)  5A                      DECB              NEXT SYMBOL
(0FE3)  26  FC                  BNE  SORT3
(0FE5)  6D  06          SORT4   TST  6,X        IF NOT AT END OF TABLE
(0FE7)  12              
(0FE8)  26  EA                  BNE  SORT1        KEEP GOING
(0FEA)  7D  00  11              TST  ANSW       IF SWAP THEN
(0FED)  26  DD                  BNE  SORT         KEEP GOING
(0FEF)  39                      RTS             ELSE RETURN
(0FF0)  D7  1A          SWAP    STAB COUNT      STORE NUMBER OF CHAR'S TO SWAP
(0FF2)  12              
(0FF3)  D7  11                  STAB ANSW       SET SWAP INDICATOR
(0FF5)  A6  00          SWAP1   LDAA X          SWAP LOOP
(0FF7)  E6  06                  LDAB 6,X
(0FF9)  A7  06                  STAA 6,X
(0FFB)  E7  00                  STAB X
(0FFD)  08                      INX
(0FFE)  7A  00  1A              DEC  COUNT
(1001)  26  F2                  BNE  SWAP1
(1003)  20  E2                  BRA  SORT4      CONTINUE
                        *
                        *
                        *       ********************************************************************
                        *       ********************************************************************
                        *                                FINAL END OF PROGRAM
                        *
```

```
END OF ASSEMBLY, UNRESOLVED ITEMS ARE:


SYMBOL TABLE FOLLOWS:

MAIN  1956    EROW     3    MOD      4    NAME     5    SIGN     9
BLNK    10    ADD1    12    SAVR    14    ALPA    16    ANSW    17
ASTK    18    BYTT    22    CC      23    CLAS    24    CMPN    25
COUT    26    CR      27    EOJL    28    FLAS    29    FWD     30
HEX     31    IXS1    32    IXS3    34    IXS3    36    LABW    38
LSTB    39    LTRL    41    NXCR    42    NULL    44    OPCE    46
PGMT    47    P_PS    49    STR1    50    STR2    52    SVPC    54
SVST    56    TBLS    58    TALY    60    TRNF    61    UNRL    63
XOPD    64    ERRA    68    ERRB    83    HEAH    89    LIND    110
NXLE   115    MSG1   120    CRLF   147    MSG2   152    OUTR    179
BYT1   181    BYT2   182    BYT3   183    BUF1   184    TBL1    241
TTBL   293    FTBL   609    STBL   735    CHRT  1936    CHRN   1939
HEXT  1942    BLKT  1950    STRT  1953    CLR1  1962    ENTY   1975
CMD   2346    EOJ   3930    LODF  2010    BUFD  2018    BUFO   2033
NXER  2056    NXC9  2086    WRIE  2226    NX_M  2096    NXI1   2098
LDEM  2108    LDI1  2117    LDI9  2152    ERRS  2153    CALR   2169
CLC4  2208    CLC2  2194    CLC9  2220    CLC3  2205    WRT0   2237
WRT1  2262    WRT2  2289    WRT4  2278    WRT3  2297    WRT5   2319
MN01  2357    MN04  2393    SYMS  3355    MN08  2461    TRNS   2525
MN05  2429    MN06  2440    PSEO  3803    MN07  2452    M_SM   2467
VERB  2762    A_FD  3605    SCH5  2505    SCH3  2495    SCH9   2524
TR01  2554    TR02  2561    TR03  2590    STRM  2725    TR08   2622
OPEN  2638    OP03  2672    OP09  2724    OP08  2721    OP06   2710
OP04  2700    ST01  2742    ST99  2756    VB02  2785    RELV   2821
VB03  2795    IMEM  2895    VB20  2805    INDU  2955    VB99   2815
DFLT  2973    GENR  3029    RL10  2837    BRIT  2863    RLER   2877
BRT9  2894    IM01  2927    IMD2  2935    IMD4  2947    DFL0   2999
DFL1  3004    DFL4  3021    GETR  3117    GEN8  3098    GEN1   3038
GEN2  3049    GEN3  3084    GEN9  3116    GAD1  3145    CVTX   3293
GAD2  3175    GAD5  3232    GAD4  3228    GAD3  3206    GAD6   3260
GAD9  3292    CVT1  3296    CVT9  3254    CVT3  3317    SM01   3377
NOHT  3448    SM10  3423    S_FD  3507    SMBD  3477    SM90   3466
SMB5  3500    SFW1  3515    SFW3  3537    SFW5  3547    SFW7   3561
SFW8  3580    RF01  3618    RF02  3630    RF03  3631    RF05   3642
RF70  3753    RF80  3739    RF10  3686    RF20  3697    RF30   3723
RF75  3764    RF76  3770    PSTL  3918    PSU1  3806    PSU3   3814
EQU   3820    FCB   3836    FB01  3847    FDB   3841    FCC    3865
FCC1  3872    ORG   3888    RMB   3898    SMDP  3961    SORT   4040
SMD2  3980    SMD4  3994    SMD6  3998    SMD9  4039    SOR4   4066
SOR1  4048    SOR2  4050    SOR3  4062    SWAP  4076    SWA1   4080

END OF NORMAL EXECUTION


PRINT REPEATED  BY OPERATOR
```

```
                    ********************************************************************
                    *                                                                  *
                    *                    M 6 8 0 0   A S S E M B L E R                 *
                    *                                                                  *
                    *                  J. F. EMMERICHS  -  MARCH, 1977                 *
                    *                          VERSION 3.1                             *
                    *                                                                  *
                    *                                                                  *
                    *                  COPYRIGHT JOHN F. EMMERICHS, 1977               *
                    *                                                                  *
                    *        ANY ORGANIZATION OR INDIVIDUAL INTERESTED IN REPRODUCING   *
                    *        THIS PROGRAM IN WHOLE OR PART OR IN USING IDEAS TAKEN      *
                    *        FROM THIS PROGRAM IS FREE TO DO SO.  HOWEVER, ALL SUCH     *
                    *        ORGANIZATIONS AND INDIVIDUALS ARE REQUESTED TO REPRODUCE   *
                    *        THIS COPYRIGHT MESSAGE AS PART OF THE INTRODUCTION TO A    *
                    *        WRITTEN WORK, OR AT THE HEAD OF PROGRAM LISTINGS.          *
                    *                                                                  *
                    ********************************************************************
                    *                                                                  *
                    *    ERROR CODE MEANINGS                                           *
                    *                                                                  *
                    *        A = VERB DOES NOT START WITH AN ALPHA CHARACTER           *
                    *        B = VERB NOT IN TABLE                                     *
                    *        C = VERB DOES NOT ALLOW IMMEDIATE (#) ADDRESSING          *
                    *        D = INVALID VERB MODIFIER                                 *
                    *        E = SYMBOL TABLE OVERFLOW                                 *
                    *        F = LABEL ALREADY IN USE                                  *
                    *        G = RELATIVE BRANCH > 128 AWAY                            *
                    *        H = FWD JUMP TABLE FULL ***** NOT AN ERROR BUT A WARNING *** *
                    *        I = FWD JUMP TABLE OVERFLOW                               *
                    *        J = INVALID CHARACTER IN HEX OR DECIMAL NUMBER            *
                    *        K = INVALID SYMBOL                                        *
                    *                                                                  *
                    ********************************************************************
                    *                                                                  *
                    *    VERB CLASS MEANINGS                                           *
                    *                                                                  *
                    *        0 = PSH AND PUL.  BOTH ARE CHANGED TO CLASS 5 FOR PROCESSING *
                    *        1 = OP CODES WHERE IMMEDIATE ADDRESSING IS VALID (EG. LDA) *
                    *        2 = OP CODES WHERE DIRECT ADDRESSING IS VALID BUT         *
                    *              IMMEDIATE IS NOT (EG. STA)                          *
                    *        3 = OP CODES WHICH CAN ADDRESS MEMORY (NOT IMMED OR DIRECT) *
                    *              OR AN ACCUMULATOR ALONE.  IF AN ACCUMULATOR IS BEING *
                    *              USED, CLASS IS CHANGED TO A 5 FOR PROCESSING (EG. CLR) *
                    *        4 = OP CODES WHICH USE RELATIVE ADDRESSING (EG. BRA)      *
                    *        5 = OP CODES WHICH USE INHERENT ADDRESSING (EG. INX)      *
                    *        E = END STATEMENT (END)                                   *
                    *        F = ASSEMBLER DIRECTIVE (EG. ORG)                         *
                    *                                                                  *
                    *    FLAGS (LOW NIBBLE OF CLASS) ARE FOR 16 BIT REGISTER OP CODES  *
                    *                                                                  *
                    *    OP CODE IN TABLE IS ACTUAL CODE USED FOR CLASSES 0, 4, 5, E, AND *
                    *       F.  FOR 1, 2, AND 3, IT IS THE IMMEDIATE CODE (EVEN IF NOT *
                    *       A VALID ADDRESSING MODE FOR A GIVEN INSTRUCTION)           *
                    *                                                                  *
                    ********************************************************************
                    **SKIP
                    ********************************************************************
                    *                                                                  *
                    *                    S T A R T   O F   L I S T I N G               *
                    *                                                                  *
                    ********************************************************************
(0000)              *         ORG   0000
(0000) 7E 00 00     *         JMP   MAIN        PGM ENTRY POINT IS AT 0000
(0020)              *         ORG   $0020
(000D)              * CR      EQU   $D
(000A)              * LF      EQU   $A
(0004)              * EOT     EQU   4
(0020)              * SPACE   EQU   $20
(0020)              * ERORSW  RMB   1           ERROR SWITCH
(0021)              * MOD     RMB   1                                    ITEMS      *
(0022)              * NAME    RMB   4                                    BLANKED FOR *
(0026)              * SIGN    RMB   1                                    EACH       *
(0027)              * BLNK    RMB   2                                    COMMAND    *
(0029)              * AD01    RMB   2           GENERAL ADDRESS VARIABLE
(002B)              * SAVADR  RMB   2           SAVE ADDRESS
(002D)              * ALPHA   RMB   1           ALPHA OPERAND SWITCH
(002E)              * ANSW    RMB   1           STRING COMPARE ANSWER
(002F)              * BYTOUT  RMB   1           BYTE OUT
(0030)              * CC      RMB   1           CURRENT CHARACTER
(0031)              * CLASS   RMB   1           VERB CLASS
(0032)              * CMPLEN  RMB   1           COMPARE LENGTH
(0033)              * COUNT   RMB   1           GENERAL COUNTER - 1
(0034)              * DELETE  RMB   1           DELETE TABLE ENTRIES FLAG
(0035)              * EJECT   RMB   1           EJECT FLAG
(0036)              * ENDFL   RMB   1           END OF BLOCK FLAG
(0037)              * FLAGS   RMB   1           VERB FLAGS
(0038)              * FWD     RMB   1           RESOLVE ADDRESS FLAG
(0039)              * HEX     RMB   1           HEXADECIMAL OPERAND SWITCH
(003A)              * IXS1    RMB   2           IX-SAVE-1
(003C)              * IXS2    RMB   2           IX-SAVE-2
(003E)              * IXS3    RMB   2           IX-SAVE-3
(0040)              * LEVEL   RMB   1           BEGIN NESTING LEVEL
(0041)              * LAB_SW  RMB   1           LABEL-SWITCH
(0042)              * LSTLAB  RMB   2           ADDRESS OF LAST LABEL ENCOUNTERED
(0044)              * LTRL    RMB   1           LETTER LENGTH
(0045)              * NOWPOS  RMB   2           CURRENT POSITION IN SYMBOL TABLE
(0047)              * NOWSTK  RMB   2           CURRENT LOCATION OF BEGIN STACK
(0049)              * NXCHAR  RMB   2           NEXT CHARACTER TO BE USED FROM BUFFER
(004B)              * NULL    RMB   2           ZERO VALUE CONSTANT
(004D)              * OPCODE  RMB   1           OP CODE
(004E)              * PGMCNT  RMB   2           PROGRAM COUNT
(0050)              * P_POS   RMB   1           NEXT BYTE POSITION TO PRINT
(0051)              * STR1    RMB   2           ADDRESS OF STRING 1
(0053)              * STR2    RMB   2           ADDRESS OF STRING 2
(0055)              * SVPC    RMB   2           SAVE PROGRAM COUNTER
(0057)              * SVST    RMB   2           SAVE STACK POINTER
(0059)              * TBLS    RMB   2           SYMBOL TABLE OFFSET
(0058)              * TALY    RMB   1           GENERAL COUNTER - 2
(005C)              * TERM    RMB   1           END OF PROGRAM FLAG
(005D)              * TMPBTM  RMB   2           CURRENT BOTTOM OF BEGIN NEST IN SYMBOL TABLE
(005F)              * TRNOFF  RMB   2           TRANSLATION TABLE OFFSET
(0061)              * UNRESL  RMB   1           NEW ITEM IN SYMBOL TABLE INDICATOR
                    **SKIP
                    *
                    *  LINEOUT
                    *
(0062)              * OUTADR  RMB   2
(0064)              * BYT1    RMB   1
(0065)              * BYT2    RMB   1
(0066)              * BYT3    RMB   1
                    *
                    *       ************************  TABLE OF TABLES
                    *
(0067)              * FTBL    RMB   2         START OF FWD REFERENCE TABLE
(0069)              * TB2WNG  RMB   2         WARNING POINT IN FWD REFERENCE TABLE
(0068)              * TB2TOP  RMB   2         END OF FWD REFERENCE TABLE
(006D)              * STBL    RMB   2         START OF SYMBOL TABLE
                    *
                    *
                    *
(006F)              * BUF1    RMB   56        INPUT BUFFER 1 - NOT INTERRUPT FED
(00A7)              * BUFEND  RMB   1
                    *
                    *
                    *       ***************  UNRESOLVED FWD REFERENCE TABLE
                    *
(00A8)              * TB01    RMB   100         SET FOR 20 FIVE BYTE ENTRIES
                    *
                    *         ADDR OF SYMB IN SYMB TABLE  CHAR (2)
                    *         PGMCNT AT UNRESOLVED REF    CHAR (2)
                    *         TYPE INDICATOR              CHAR (1)   -1 = ONE BYTE OUT
                    *                                                 0 = TWO BYTES OUT
                    *                                                +1 = RELATIVE
                    *
                    *
                    *       ***************  SYMBOL TABLE
                    *
(010C)              * TB02    RMB   1           ENTRY POINT FOR USER WRITTEN SUBROUTINES
(010D)              * TB03    RMB   900         SET FOR 150 SIX BYTE ENTRIES
                    *
                    *         SYMBOL NAME  CHAR (4)
                    *         SYMBOL ADDR  CHAR (2)
                    *
(0491)              * TB04    RMB   1           TABLE STOP BYTE
(0492)              *         RMB   2           START OF FIRST BEGIN BLOCK IN STBL
                    **SKIP
                    *
                    *       ***************  EXTERNAL ADDRESS TABLE
                    *
(0494) 7E E1 D1     * CHROUT  JMP   $E1D1       MIKBUG PRINT CHARACTER
(0497) 7E E1 AC     * CHRIN   JMP   $E1AC       MIKBUG READ CHARACTER
(049A) 36           * HEXOUT  PSHA              PRINT A BYTE FROM THE A ACCUMULATOR
(049B) BD E0 67     *         JSR   $E067         AS TWO HEXIDECIMAL CHARACTERS
(049E) 32           *         PULA
(049F) 7E E0 68     *         JMP   $E068
(04A2) 7E E0 CC     * BLKOUT  JMP   $E0CC       MIKBUG PRINT A BLANK
(04A5) 7E E0 7E     * STROUT  JMP   $E07E       MIKBUG PRINT A STRING ENDING WITH '04'
(04A8) 7E E0 E3     * MIKBUG  JMP   $E0E3       MIKBUG MAINLINE
```

```
                        *
                        *       *************** TRANSLATIN TABLE OFFSET TABLE
                        *
(04AB)                  TBL1    EQU  *
(04AB)  00  06                  FCB  0,6        A
(04AD)  06  12                  FCB  6,18       B
(04AF)  18  08                  FCB  24,8       C
(04B1)  20  04                  FCB  32,4       D
(04B3)  24  03                  FCB  36,3       E
(04B5)  27  03                  FCB  39,3       F
(04B7)  00  00                  FCB  0,0        G
(04B9)  00  00                  FCB  0,0        H
(04BB)  2A  03                  FCB  42,3       I
(04BD)  2D  02                  FCB  45,2       J
(04BF)  00  00                  FCB  0,0        K
(04C1)  2F  04                  FCB  47,4       L
(04C3)  00  00                  FCB  0,0        M
(04C5)  33  02                  FCB  51,2       N
(04C7)  35  02                  FCB  53,2       O
(04C9)  37  02                  FCB  55,2       P
(04CB)  00  00                  FCB  0,0        Q
(04CD)  39  05                  FCB  57,5       R
(04CF)  3E  0A                  FCB  62,10      S
(04D1)  48  07                  FCB  72,7       T
(04D3)  00  00                  FCB  0,0        U
(04D5)  00  00                  FCB  0,0        V
(04D7)  4F  01                  FCB  79,1       W
(04D9)  00  00                  FCB  0,0        X
(04DB)  00  00                  FCB  0,0        Y
(04DD)  00  00                  FCB  0,0        Z
                        **SKIP
                        *       *************** MNEMONIC LOOKUP TABLE
(04DF)                  TTBL    RMB  0
                        *                       MNEMONIC OPERATOR KEY  CHAR (2)
                        *                       OP CODE                CHAR (1)
                        *                       CLASS AND FLAG         CHAR (1)
                        *
                        *         A
                        *
(04DF)  42  41                  FCC  2,BA               ABA
(04E1)  1B  50                  FCB  $1B,$50
(04E3)  44  43                  FCC  2,DC               ADC
(04E5)  89  10                  FCB  $89,$10
(04E7)  44  44                  FCC  2,DD               ADD
(04E9)  8B  10                  FCB  $8B,$10
(04EB)  4E  44                  FCC  2,ND               AND
(04ED)  84  10                  FCB  $84,$10
(04EF)  53  4C                  FCC  2,SL               ASL
(04F1)  48  30                  FCB  $48,$30
(04F3)  53  52                  FCC  2,SR               ASR
(04F5)  47  30                  FCB  $47,$30
                        *
                        *         B
                        *
(04F7)  43  43                  FCC  2,CC               BCC
(04F9)  24  40                  FCB  $24,$40
(04FB)  43  53                  FCC  2,CS               BCS
(04FD)  25  40                  FCB  $25,$40
(04FF)  45  51                  FCC  2,EQ               BEQ
(0501)  27  40                  FCB  $27,$40
(0503)  47  45                  FCC  2,GE               BGE
(0505)  2C  40                  FCB  $2C,$40
(0507)  47  4E                  FCC  2,GN               BGN   (BEGIN)
(0509)  06  F0                  FCB  $06,$F0
(050B)  47  54                  FCC  2,GT               BGT
(050D)  2E  40                  FCB  $2E,$40
(050F)  48  49                  FCC  2,HI               BHI
(0511)  22  40                  FCB  $22,$40
(0513)  49  54                  FCC  2,IT               BIT
(0515)  85  10                  FCB  $85,$10
(0517)  4C  45                  FCC  2,LE               BLE
(0519)  2F  40                  FCB  $2F,$40
(051B)  4C  53                  FCC  2,LS               BLS
(051D)  23  40                  FCB  $23,$40
(051F)  4C  54                  FCC  2,LT               BLT
(0521)  2D  40                  FCB  $2D,$40
(0523)  4D  49                  FCC  2,MI               BMI
(0525)  28  40                  FCB  $28,$40
(0527)  4E  45                  FCC  2,NE               BNE
(0529)  26  40                  FCB  $26,$40
(052B)  50  4C                  FCC  2,PL               BPL
(052D)  2A  40                  FCB  $2A,$40
(052F)  52  41                  FCC  2,RA               BRA
(0531)  20  40                  FCB  $20,$40
(0533)  53  52                  FCC  2,SR               BSR
(0535)  8D  40                  FCB  $8D,$40
(0537)  56  43                  FCC  2,VC               BVC
(0539)  28  40                  FCB  $28,$40
(053B)  56  53                  FCC  2,VS               BVS
(053D)  29  40                  FCB  $29,$40
                        **SKIP
                        *
                        *         C
                        *
(053F)  42  41                  FCC  2,BA               CBA
(0541)  11  50                  FCB  $11,$50
(0543)  4C  43                  FCC  2,LC               CLC
(0545)  0C  50                  FCB  $0C,$50
(0547)  4C  49                  FCC  2,LI               CLI
(0549)  0E  50                  FCB  $0E,$50
(054B)  4C  52                  FCC  2,LR               CLR
(054D)  4F  30                  FCB  $4F,$30
(054F)  4C  56                  FCC  2,LV               CLV
(0551)  0A  50                  FCB  $0A,$50
(0553)  4D  50                  FCC  2,MP               CMP
(0555)  81  10                  FCB  $81,$10
(0557)  4F  4D                  FCC  2,OM               COM
(0559)  43  30                  FCB  $43,$30
(055B)  50  58                  FCC  2,PX               CPX
(055D)  8C  11                  FCB  $8C,$11
                        *
                        *         D
                        *
(055F)  41  41                  FCC  2,AA               DAA
(0561)  19  50                  FCB  $19,$50
(0563)  45  43                  FCC  2,EC               DEC
(0565)  4A  30                  FCB  $4A,$30
(0567)  45  53                  FCC  2,ES               DES
(0569)  34  50                  FCB  $34,$50
(056B)  45  58                  FCC  2,EX               DEX
(056D)  09  50                  FCB  $09,$50
                        *
                        *         E
                        *
(056F)  4E  44                  FCC  2,ND               END
(0571)  00  E0                  FCB  $00,$E0
(0573)  4F  52                  FCC  2,OR               EOR
(0575)  88  10                  FCB  $88,$10
(0577)  51  55                  FCC  2,QU               EQU
(0579)  00  F0                  FCB  $00,$F0
                        *
                        *         F
                        *
(057B)  43  42                  FCC  2,CB               FCB
(057D)  01  F0                  FCB  $01,$F0
(057F)  43  43                  FCC  2,CC               FCC
(0581)  02  F0                  FCB  $02,$F0
(0583)  44  42                  FCC  2,DB               FDB
(0585)  03  F0                  FCB  $03,$F0
                        *
                        *         (G, H), I
                        *
(0587)  4E  43                  FCC  2,NC               INC
(0589)  4C  30                  FCB  $4C,$30
(058B)  4E  53                  FCC  2,NS               INS
(058D)  31  50                  FCB  $31,$50
(058F)  4E  58                  FCC  2,NX               INX
(0591)  08  50                  FCB  $08,$50
                        **SKIP
                        *
                        *         J
                        *
(0593)  4D  50                  FCC  2,MP               JMP
(0595)  4E  31                  FCB  $4E,$31
(0597)  53  52                  FCC  2,SR               JSR
(0599)  8D  31                  FCB  $8D,$31
                        *
                        *         (K), L
                        *
(059B)  44  41                  FCC  2,DA               LDA
(059D)  86  10                  FCB  $86,$10
(059F)  44  53                  FCC  2,DS               LDS
(05A1)  8E  11                  FCB  $8E,$11
(05A3)  44  58                  FCC  2,DX               LDX
(05A5)  CE  11                  FCB  $CE,$11
(05A7)  53  52                  FCC  2,SR               LSR
(05A9)  44  30                  FCB  $44,$30
                        *
                        *         (M), N
                        *
(05AB)  45  47                  FCC  2,EG               NEG
(05AD)  40  30                  FCB  $40,$30
```

```
(05AF) 4F 50    |        FCC  2,OP            NOP
(05B1) 01 50    |        FCB  $01,$50
                | *
                | *         O
                | *
(05B3) 52 41    |        FCC  2,RA            ORA
(05B5) 8A 10    |        FCB  $8A,$10
(05B7) 52 47    |        FCC  2,RG            ORG
(05B9) 04 F0    |        FCB  $04,$F0
                | *
                | *         P
                | *
(05BB) 53 48    |        FCC  2,SH            PSH
(05BD) 36 00    |        FCB  $36,$00
(05BF) 55 4C    |        FCC  2,UL            PUL
(05C1) 32 00    |        FCB  $32,$00
                | *
                | *         (Q), R
                | *
(05C3) 4D 42    |        FCC  2,MB            RMB
(05C5) 05 F0    |        FCB  $05,$F0
(05C7) 4F 4C    |        FCC  2,OL            ROL
(05C9) 49 30    |        FCB  $49,$30
(05CB) 4F 52    |        FCC  2,OR            ROR
(05CD) 46 30    |        FCB  $46,$30
(05CF) 54 49    |        FCC  2,TI            RTI
(05D1) 38 50    |        FCB  $38,$50
(05D3) 54 53    |        FCC  2,TS            RTS
(05D5) 39 50    |        FCB  $39,$50
                | **SKIP
                | *
                | *         S
                | *
(05D7) 42 41    |        FCC  2,BA            SBA
(05D9) 10 50    |        FCB  $10,$50
(05DB) 42 43    |        FCC  2,BC            SBC
(05DD) 82 10    |        FCB  $82,$10
(05DF) 45 43    |        FCC  2,EC            SEC
(05E1) 0D 50    |        FCB  $0D,$50
(05E3) 45 49    |        FCC  2,EI            SEI
(05E5) 0F 50    |        FCB  $0F,$50
(05E7) 45 56    |        FCC  2,EV            SEV
(05E9) 0B 50    |        FCB  $0B,$50
(05EB) 54 41    |        FCC  2,TA            STA
(05ED) 87 20    |        FCB  $87,$20
(05EF) 54 53    |        FCC  2,TS            STS
(05F1) 8F 21    |        FCB  $8F,$21
(05F3) 54 58    |        FCC  2,TX            STX
(05F5) CF 21    |        FCB  $CF,$21
(05F7) 55 42    |        FCC  2,UB            SUB
(05F9) 80 10    |        FCB  $80,$10
(05FB) 57 49    |        FCC  2,WI            SWI
(05FD) 3F 50    |        FCB  $3F,$50
                | *
                | *         T
                | *
(05FF) 41 42    |        FCC  2,AB            TAB
(0601) 16 50    |        FCB  $16,$50
(0603) 41 50    |        FCC  2,AP            TAP
(0605) 06 50    |        FCB  $06,$50
(0607) 42 41    |        FCC  2,BA            TBA
(0609) 17 50    |        FCB  $17,$50
(0608) 50 41    |        FCC  2,PA            TPA
(060D) 07 50    |        FCB  $07,$50
(060F) 53 54    |        FCC  2,ST            TST
(0611) 4D 30    |        FCB  $4D,$30
(0613) 53 58    |        FCC  2,SX            TSX
(0615) 30 50    |        FCB  $30,$50
(0617) 58 53    |        FCC  2,XS            TXS
(0619) 35 50    |        FCB  $35,$50
                | *
                | *         (U, V), W
                | *
(061B) 41 49    |        FCC  2,AI            WAI
(061D) 3E 50    |        FCB  $3E,$50
                | *
                | *         (X, Y, Z)
                | *
                | **SKIP
                | *         ************************   MESSAGES FOLLOW
(061F) 0D 00 00 | ERRA   FCB  CR,,                   ERROR MESSAGE
(0622) 2A 2A 2A |        FCC  /*** ERROR - /
(0625) 20 45 52 |
(0628) 52 4F 52 |
(062B) 20 2D 20 |
```

```
(062E)          | ERRB   RMB  1
(062F) 0A 20 04 |        FCB  LF,SPACE,EOT
(0632) 0D 0A 0A | HEADR  FCB  CR,LF,LF,LF
(0635) 0A       |
(0636) 4C 4F 43 |        FCC  /LOCN B1 B2 B3 /
(0639) 4E 20 42 |
(063C) 31 20 42 |
(063F) 32 20 42 |
(0642) 33 20    |
(0644) 0A 0A 00 | LINFED FCB  LF,LF,,EOT
(0647) 04       |
(0648) 0D 00 00 | NXLINE FCB  CR,,,$12,EOT
(064B) 12 04    |
                | *                  NOTE: TAPE ON CHARACTER IS THE LAST
                | *                        CHARACTER  PRINTED IN THE ABOVE LITERAL
                | *
(064D) 0D 00    | MSG1   FCB  CR,
(064F) 2A 2A 2A |        FCC  /*** UNRESOLVED:/
(0652) 20 55 4E |
(0655) 52 45 53 |
(0658) 4F 4C 56 |
(065B) 45 44 3A |
(065E) 0D 0A 00 |        FCB  CR,LF,,EOT
(0661) 04       |
(0662) 0D 00    | MSG2   FCB  CR,
(0664) 2A 2A 2A |        FCC  /*** SYMBOLS?/
(0667) 20 53 59 |
(066A) 4D 42 4F |
(066D) 4C 53 3F |
(0670) 0D 0A 00 | CRLF   FCB  CR,LF,,EOT
(0673) 04       |
(0674) 2A 20 20 | ASTK   FCC  /*   /     FOR COMPARE WITH *
(0677) 20       |
(0678) 58 20 20 | XOPND  FCC  /X   /     FOR COMPARE WITH X
(067B) 20       |
(067C) 0D       | EOL    FCB  CR         VALUE FOR END OF INPUT LINE
                | **SKIP
                | *
                | *
                | *       ****************************************************************
                | *                        MAIN PROCEDURE LOOP
                | *
(067D) 86 01    | MAIN   LDAA #1             INITIALIZE FLAGS
(0001) 06 7D    |
(067F) 97 35    |        STAA EJECT
(0681) 97 40    |        STAA LEVEL
(0683) 86 39    |        LDAA #$39           CHANGE THIS TO $01 IF USER ROUTINES
(0685) CE 01 0C |        LDX  #TB02            ARE USED
(0688) DF 6B    |        STX  TB2TOP         EXECUTE USER ROUTINES
(068A) A7 00    |        STAA X
(068C) AD 00    |        JSR  X
(068E) 09       |        DEX                 LOAD FWD TABLE WARNING LEVEL
(068F) 09       |        DEX
(0690) 09       |        DEX
(0691) 09       |        DEX
(0692) 09       |        DEX
(0693) DF 69    |        STX  TB2WNG
(0695) CE 00 00 |        LDX  #0             INITIALIZE ADDRESSES
(0698) DF 4E    |        STX  PGMCNT
(069A) DF 62    |        STX  OUTADDR        INITIALIZE OUTPUT LINE
(069C) 86 20    |        LDAA #SPACE
(069E) 97 64    |        STAA BYT1
(06A0) 97 65    |        STAA BYT2
(06A2) 97 66    |        STAA BYT3
(06A4) CE 00 A8 |        LDX  #TB01          CLEAR THE SYMBOL TABLE AND THE
(06A7) DF 67    |        STX  FTBL
(06A9) 6F 00    | CLR1   CLR  X                FORWARD JUMP TABLE
(06AB) 08       |        INX
(06AC) 8C 04 91 |        CPX  #TB04
(06AF) 26 F8    |        BNE  CLR1
(06B1) DF 47    |        STX  NOWSTK
(06B3) 8E 01 0D |        LDS  #TB03          INITIALIZE TOP OF TABLE STACK
(06B6) 9F 45    |        STS  NOWPOS
(06B8) AF 01    |        STS  1,X
(06BA) 9F 6D    |        STS  STBL
(06BC) 86 39    |        LDAA #$39           BLOCK FURTHER
(06BE) DE 6B    |        LDX  TB2TOP
(06C0) A7 00    |        STAA X                EXECUTION OF USER PROCS
                | **SKIP
                | *         THIS IS THE CONTINUE ENTRY POINT
                | *
(06C2) 7D 00 40 | ENTRY  TST  LEVEL          IF NEW PROGRAM,
(06C5) 27 B6    |        BEQ  MAIN             GO TO COLD START
(06C7) CE 00 6F |        LDX  #BUF1          INITIALIZE
(06CA) DF 49    |        STX  NXCHAR           INCOMING CHARACTER POINTER
```

```
(06CC) CE 06 C2 |        LDX  #ENTRY      LOAD SECONDARY ENTRY POINT FOR MIKBUG MONITOR
(06CF) FF A0 48 |        STX  $A048         (WARM START)
(06D2) 8E A0 7F |        LDS  #$A07F       LOAD STACK POINTER FOR MIKBUG MONITOR
(06D5) CE 00 00 |        LDX  #0           INITIALIZE ZERO VALUES
(06D8) DF 29    |        STX  AD01
(06DA) DF 48    |        STX  NULL
(06DC) 4F       |        CLRA
(06DD) 97 36    |        STAA ENDFL
(06DF) 97 5C    |        STAA TERM
(06E1) 97 38    |        STAA FWD
(06E3) 97 50    |        STAA P_POS
(06E5) 97 58    |        STAA TALY
(06E7) 97 61    |        STAA UNRESL
(06E9) 86 04    |        LDAA #4           INITIALIZE COMPARE LENGTH
(06EB) 97 32    |        STAA CMPLEN
(06ED) BD 00 00 |        JSR  CMD          MAIN PROCEDURE LOOP - CALL PROC-COMMANDS
(06F0) 7E 04 A8 |        JMP  MIKBUG       JUMP TO MONITOR MAINLINE
                |**SKIP  ***************************************************************
                | *                                                                     *
                | *                          I/O SUBROUTINES                            *
                | *                                                                     *
                | ***********************************************************************
                | *                                  LOAD BUFFER FROM TERMINAL
                | *
(06F3) 7D 00 35 | LODBUF TST  EJECT        IF EJECT IS SET
(06F6) 23 00    |        BLS  LOB1           PRINT HEADINGS
(06F8) CE 06 32 |        LDX  #HEADR
(06FB) BD 04 A5 |        JSR  STROUT
(06FE) 7F 00 35 |        CLR  EJECT
(0701) CE 00 6F | LOB1   LDX  #BUF1        SET INDEX TO BUFFER
(06F7) 09       |
(0704) 86 3E    |        LDAA #'>          PRINT PROMPT CHAR          *** PROMPT CHARACTER
(0706) BD 04 94 |        JSR  CHROUT
(0709) BD 04 97 | LDB5   JSR  CHRIN        GET CHAR
(070C) 81 21    |        CMPA #'!          PROCESS BACKSPACE          *** BACKSPACE CHARACTER
(070E) 26 00    |        BNE  BUFGO
(0710) 8C 00 6F |        CPX  #BUF1
(0713) 27 F4    |        BEQ  LDB5
(0715) 09       |        DEX
(0716) 20 F1    |        BRA  LDB5
(0718) A7 00    | BUFGO  STAA X            STORE CHARACTER
(070F) 08       |
(071A) B1 06 7C |        CMPA EOL          IF NOT END OF LINE THEN
(071D) 26 01    |        BNE  *+3            CONTINUE
(071F) 39       |        RTS
(0720) 8C 00 A7 |        CPX  #BUFEND      IF NOT AT END OF BUFFER
(0723) 27 E4    |        BEQ  LDB5
(0725) 81 20    |        CMPA #$20         AND IF CHARACTER
(0727) 2B E0    |        BMI  LDB5           IS BETWEEN HEX 20 AND 5F
(0729) 81 5F    |        CMPA #$5F
(0728) 22 DC    |        BHI  LDB5
(072D) 08       |        INX               INCREMENT INDEX
(072E) 20 D9    |        BRA  LDB5         CONTINUE
                |**SKIP
                | *
                | *      ***************************************************************
                | *                              RETRIEVE NEXT CHAR
                | *
(0730) DF 3E    | NX_CHR STX  IXS3         SAVE IX
(0732) DE 49    |        LDX  NXCHAR       LOAD NEXT CHAR TO USE POSITION INTO IX
(0734) 96 30    |        LDAA CC           IF LAST CHAR
(0736) B1 06 7C |        CMPA EOL            WAS NOT A CARRAIGE RETURN,
(0739) 26 00    |        BNE  NXC9           THEN CONTINUE
(073B) 7D 00 50 |        TST  P_POS        ELSE IF P_POS = NULL THEN
(073E) 27 03    |        BEQ  *+5            GO TO GET INPUT
(0740) BD 00 00 |        JSR  WRITE        ELSE WRITE OUT LINE OF PRINT
(0743) 86 01    |        LDAA #$01           SET P-POS
(0745) 97 50    |        STAA P_POS            TO 1
(0747) BD 06 F3 |        JSR  LODBUF         LOAD BUFFER
(074A) CE 00 6F |        LDX  #BUF1          SET CURRENT
(074D) DF 49    |        STX  NXCHAR           CHAR POINTER
(074F) A6 00    | NXC9   LDAA X            LOAD CHAR INTO A
(073A) 14       |
(0751) 97 30    |        STAA CC             AND CC
(0753) 08       |        INX               INCREMENT NEXT CHAR PTR
(0754) DF 49    |        STX  NXCHAR
(0756) DE 3E    |        LDX  IXS3         RESTORE IX
(0758) 39       |        RTS                 AND RETURN
                |**SKIP
                | *
                | *      ***************************************************************
                | *                      FIND NEXT ITEM FROM INPUT STREAM
                | *
(0759) 96 30    | NX_ITM LDAA CC         IF NEXT ITEM
(075B) 81 20    | NXI1   CMPA #SPACE       BLANK THEN
(075D) 27 01    |        BEQ  *+3            CONTINUE
(075F) 39       |        RTS               RETURN
(0760) BD 07 30 |        JSR  NX_CHR         TRY NEXT CHARACTER FOR BEING A NON BLANK
(0763) 20 F6    |        BRA  NXI1
                | *
                | *
                | *      ***************************************************************
                | *                  LOAD ITEM FROM INPUT STREAM INTO 'NAME'
                | *
(0765) DE 27    | LD_ITM LDX  BLNK         BLANK OUT NAME
(0767) DF 22    |        STX  NAME
(0769) DF 24    |        STX  NAME+2
(076B) CE 00 22 |        LDX  #NAME
                | *
(076E) D6 30    | LDI1   LDAB CC           LOAD CURRENT CHAR
(0770) D1 27    |        CMPB BLNK
(0772) 27 00    |        BEQ  LDI9         IF BLANK,
(0774) F1 06 7C |        CMPB EOL            END OF INPUT LINE,
(0777) 27 00    |        BEQ  LDI9
(0779) C1 2B    |        CMPB #$2B           PLUS,
(077B) 27 00    |        BEQ  LDI9
(077D) C1 2D    |        CMPB #$2D           MINUS,
(077F) 27 00    |        BEQ  LDI9
(0781) C1 2C    |        CMPB #$2C           OR COMMA,
(0783) 27 00    |        BEQ  LDI9             THEN CHECK FOR LITERAL INPUT
(0785) E7 00    | LDI5   STAB X            LOAD CHAR
(0787) 8C 00 25 |        CPX  #NAME+3        IF NAME IS FULL
(078A) 27 01    |        BEQ  *+3            DON'T
(078C) 08       |        INX               INCREMENT INDEX
(078D) BD 07 30 |        JSR  NX_CHR         GET NEXT CHAR
(0790) 20 DC    |        BRA  LDI1
(0792) 09       | LDI9   DEX               IF PREVIOUS CHAR
(0773) 1E       |
(0778) 19       |
(077C) 15       |
(0780) 11       |
(0784) 0D       |
(0793) A6 00    |        LDAA X              WAS NOT AN APOSTROPHE
(0795) 81 27    |        CMPA #''
(0797) 26 00    |        BNE  LD99           GO TO END
(0799) 09       |        DEX
(079A) A6 00    |        LDAA X            IF TWO CHARACTERS BACK
(079C) 81 27    |        CMPA #''            WAS AN APOSTROPHE
(079E) 27 00    |        BEQ  LD98           GO TO END
(07A0) 08       |        INX               ELSE LOAD DELIMITER AS INPUT CHARACTER
(07A1) 08       |        INX
(07A2) 20 E1    |        BRA LDI5
(07A4) 08       | LD98   INX               RESTORE INDEX
(079F) 04       |
(07A5) 08       | LD99   INX
(0798) 0C       |
(07A6) 39       |        RTS               RETURN
                |**SKIP
                | *
                | *
                | *      ***************************************************************
                | *                                 ERROR ROUTINE
                | *
(07A7) B7 06 2E | ERRS   STAA ERRB         LOAD ERROR MESSAGE
(07AA) DF 3A    |        STX  IXS1         SAVE INDEX
(07AC) CE 06 1F |        LDX  #ERRA        WRITE
(07AF) BD 04 A5 |        JSR  STROUT         ERROR MSG
(07B2) DE 3A    |        LDX  IXS1         RESTORE INDEX
(07B4) 7C 00 20 |        INC  ERORSW       SET ERROR SWITCH (ON)
(07B7) 39       |        RTS               RETURN
                | *
                | *      ***************************************************************
                | *                            CALCULATE PRINT LINE
                | *
(07B8) 7D 00 20 | CALCWR TST  ERORSW       IF ERROR CONDITION EXISTS,
(07BB) 23 01    |        BLS  *+3            RETURN
(07BD) 39       |        RTS
(07BE) 96 2F    |        LDAA BYTOUT
(07C0) D6 50    |        LDAB P_POS        IF PRINT_POS
(07C2) C1 01    |        CMPB #$01           = 1 THEN
(07C4) 27 00    |        BEQ  CLC4           GO TO INITIALIZE (WITHOUT PRINT)
(07C6) C1 02    |        CMPB #$02         IF PRINT_POS
(07C8) 26 00    |        BNE  CLC2           NOT = 2 THEN CONTINUE
(07CA) 97 65    |        STAA BYT2         ELSE LOAD SECOND BYTE
(07CC) 7C 00 50 |        INC  P_POS          INCREASE PRINT_POS
(07CF) 20 00    |        BRA  CLC9           GO TO END
(07D1) C1 03    | CLC2   CMPB #$03         IF PRINT_POS
(07C9) 07       |
(07D3) 26 00    |        BNE  CLC3           NOT = 3 THEN CONTINUE
(07D5) 97 66    |        STAA BYT3         ELSE LOAD THIRD BYTE
(07D7) 7C 00 50 |        INC  P_POS          INCREASE PRINT_POS
```

```
(07DA) 20 00      |        BRA  CLC9         GO TO END
(07DC) BD 00 00   | CLC3   JSR  WRITE        WRITE OUT CURRENT LINE
(07D4) 07         |
(07DF) DE 4E      | CLC4   LDX  PGMCNT       LOAD
(07C5) 19         |
(07E1) DF 62      |        STX  OUTADR         CURRENT ADDRESS
(07E3) 96 2F      |        LDAA BYTOUT
(07E5) 97 64      |        STAA BYT1         LOAD BYTE1
(07E7) 86 02      |        LDAA #$02         SET PRINT_POS
(07E9) 97 50      |        STAA P_POS          = 2
(07EB) DE 4E      | CLC9   LDX  PGMCNT       INCREASE
(07D0) 1A         |
(07DB) 0F         |
(07ED) 08         |        INX                 PGMCNT
(07EE) DF 4E      |        STX  PGMCNT
(07F0) 39         |        RTS               RETURN
                  | **SKIP
                  | *
                  | *     ***************************************************************
                  | *                                        WRITE ROUTINE
                  | *
(07F1) DF 3E      | WRITE  STX  IXS3         SAVE INDEX
(0741) 07 F1      |
(07DD) 07 F1      |
(07F3) CE 06 48   |        LDX  #NXLINE      START NEXT
(07F6) BD 04 A5   |        JSR  STROUT         PRINT LINE
(07F9) CE 75 30   |        LDX  #$7530       LOAD IX WITH 30,000
(07FC) 6D 00      | WRT0   TST  X              EXECUTE TIMING LOOP
(07FE) 09         |        DEX                 TO PAUSE HALF A SECOND
(07FF) 26 FB      |        BNE  WRT0                                  ***
(0801) 86 01      |        LDAA #$01         WRITE LOADER RECOGNITION  (1)
(0803) BD 04 94   |        JSR  CHROUT         CHARACTER               ***
(0806) 96 62      |        LDAA OUTADR       WRITE
(0808) BD 04 9A   |        JSR  HEXOUT         OUT FOUR
(080B) 96 63      |        LDAA OUTADR+1       BYTES
(080D) BD 04 9A   |        JSR  HEXOUT         OF ADDRESS
(0810) C6 01      |        LDAB #$01
(0812) CE 00 64   |        LDX  #BYT1
(0815) D1 50      | WRT1   CMPB P_POS        LOOP THROUGH BYTES OUT
(0817) 2B 00      |        BMI  WRT2         IF P_POS DONE THEN
(0819) 26 00      |        BNE  WRT4
(081B) 86 01      |        LDAA #$01           WRITE OUT                 ***
(081D) BD 04 94   |        JSR  CHROUT           LOADER STOP CHARACTER (1)
(0820) 86 14      |        LDAA #$14           WRITE OUT                 ***
(0822) BD 04 94   |        JSR  CHROUT           TAPE OFF CHAR
(0825) BD 04 A2   | WRT4   JSR  BLKOUT
(081A) 0A         |
(0828) BD 04 A2   |        JSR  BLKOUT
(082B) BD 04 A2   |        JSR  BLKOUT
(082E) 20 00      |        BRA  WRT3
(0830) BD 04 A2   | WRT2   JSR  BLKOUT       WRITE OUT BLANK
(0818) 17         |
(0833) A6 00      |        LDAA X            GET NEXT BYTE OUT
(0835) BD 04 9A   |        JSR  HEXOUT       PRINT IT IN HEX
(0838) 08         | WRT3   INX               GO TO NEXT BYTE
(082F) 08         |
(0839) 5C         |        INCB
(083A) C1 04      |        CMPB #$04         IF ALL NOT OUT THEN
(083C) 26 D7      |        BNE  WRT1           GO TO NEXT LOOP
(083E) 96 50      |        LDAA P_POS        IF REQUIRED,
(0840) 81 04      |        CMPA #4
(0842) 26 00      |        BNE  WRT5
(0844) 86 01      |        LDAA #$01         WRITE OUT                 ***
(0846) BD 04 94   |        JSR  CHROUT         LOADER STOP CHARACTER (2)
(0849) 86 14      |        LDAA #$14         WRITE OUT                 ***
(084B) BD 04 94   |        JSR  CHROUT         TAPE OFF CHARACTER
                  | **SKIP
                  | *
                  | ******* WRITE CONTINUED
                  | *
(084E) BD 04 A2   | WRT5   JSR  BLKOUT       WRITE OUT FINAL BLANK
(0843) 0A         |
(0851) DE 4E      |        LDX  PGMCNT       SET UP NEXT
(0853) DF 62      |        STX  OUTADR         ADDRESS IN WRITE STRING
(0855) DE 3E      |        LDX  IXS3         RESTORE INDEX
(0857) 86 0A      |        LDAA #$0A         WRITE OUT
(0859) BD 04 94   |        JSR  CHROUT         LINE FEED
(085C) 96 27      |        LDAA BLNK           BLANK OUT
(085E) 97 64      |        STAA BYT1           OUTPUT AREA
(0860) 97 65      |        STAA BYT2
(0862) 97 66      |        STAA BYT3
(0864) 86 01      |        LDAA #$01         RESET OUTPUT
(0866) 97 50      |        STAA P_POS          POINTER
(0868) 39         |        RTS
```

```
                  | **SKIP
                  | *****************************************************************************
                  | *                                                                           *
                  | *                        PROCESS COMMAND PROCEDURE                          *
                  | *                                                                           *
                  | *****************************************************************************
(0869) CE 00 21   | CMD    LDX  #MOD         GET STARTING ADDRESS
(06EE) 08 69      |
(086C) C6 20      |        LDAB #SPACE       SET CHARACTER TO BLANK WITH
(086E) 7F 00 20   |        CLR  ERORSW       CLEAR ERROR SWITCH
(0871) 7F 00 61   |        CLR  UNRESL       CLEAR UNRESOLVED SWITCH
(0874) E7 00      | MN01   STAB X            BLANK CURRENT BYTE
(0876) 08         |        INX               GO TO NEXT BYTE
(0877) 8C 00 29   |        CPX  #AD01        LAST TO BE BLANKED?
(087A) 26 F8      |        BNE  MN01           IF NOT, CONTINUE
(087C) B6 06 7C   |        LDAA EOL
(087F) 97 30      |        STAA CC
(0881) BD 07 30   |        JSR  NX_CHR       LOAD FIRST CHAR
(0884) B1 06 7C   |        CMPA EOL
(0887) 27 E0      |        BEQ  CMD          BLANK LINE - FIRST CHAR IS A CARRAIGE RETURN
(0889) 81 2A      |        CMPA #$2A         COMMENT ENTRY - IF FIRST CHAR IS AN '*'
(088B) 26 00      |        BNE  MN02
(088D) BD 00 00   |        JSR  CMNT         CHECK FOR SPECIAL COMMENT LINES
(0890) 20 D7      |        BRA  CMD
(0892) 91 27      | MN02   CMPA BLNK         TEST FOR LABEL AND IF ABSENT
(088C) 05         |
(0894) 27 00      |        BEQ  MN04           GO TO START-VERB
(0896) BD 07 65   |        JSR  LD_ITM       ELSE, PROCESS LABEL
(0899) 7C 00 41   |        INC  LAB_SW
(089C) BD 00 00   |        JSR  SYMB_S
(089F) BD 07 59   | MN04   JSR  NX_ITM       LOOK FOR START OF VERB
(0895) 09         |
(08A2) B1 06 7C   |        CMPA EOL
(08A5) 27 00      |        BEQ  MN08         BLANK LINE WITH A SPACE OR LABEL IN CC-1
(08A7) BD 07 65   |        JSR  LD_ITM       LOAD VERB
(08AA) 86 02      |        LDAA #2
(08AC) 97 32      |        STAA CMPLEN
(08AE) BD 00 00   |        JSR  TRNS         LOOK UP VERB
(08B1) 86 04      |        LDAA #4
(08B3) 97 32      |        STAA CMPLEN
(08B5) 7D 00 20   |        TST  ERORSW       BRANCH IF ERROR
(08B8) 22 AF      |        BHI  CMD            TO START OF NEXT COMMAND
(08BA) 96 31      |        LDAA CLASS
(08BC) 81 0E      |        CMPA #$0E
(08BE) 26 00      |        BNE  MN05         CHECK FOR END-OF-BLOCK-VERB
(08C0) BD 00 00   |        JSR  END            WRITE OUT LAST LINE
(08C3) 7D 00 5C   |        TST  TERM         TEST FOR END OF PROGRAM
(08C6) 23 A1      |        BLS  CMD          IF NOT, GO TO NEXT COMMAND
(08C8) 39         |        RTS               IF SO, RETURN
(08C9) D6 4D      | MN05   LDAB OPCODE       LOAD POSSIBLE MACRO NUMBER INTO B
(08BF) 09         |
(08CB) 81 0F      |        CMPA #$0F
(08CD) 26 00      |        BNE  MN06         CHECK FOR MACRO
(08CF) BD 00 00   |        JSR  PSEUDO         AND PROCESS IF FOUND
(08D2) 20 00      |        BRA  MN08           THEN GO TO NEXT COMMAND
                  | **SKIP
                  | *
                  | ******* CMD CONTINUED
                  | *
(08D4) BD 07 59   | MN06   JSR  NX_ITM       SEARCH FOR THE START OF THE OPERANDS
(08CE) 05         |
(08D7) 81 23      |        CMPA #'#            IF IT STARTS
(08D9) 26 00      |        BNE  MN07           WITH '#' THEN
(08DB) 97 21      |        STAA MOD            LOAD MOD
(08DD) BD 07 30   |        JSR  NX_CHR         WITH '#' AND CONTINUE
(08E0) BD 07 59   | MN07   JSR  NX_ITM       SEARCH REST OF OPERAND
(08DA) 05         |
(08E3) BD 00 00   |        JSR  M_SRCH         FOR VERB MODIFIER
(08E6) BD 00 00   |        JSR  VERB         PROCESS VERB
(08E9) BD 00 00   | MN08   JSR  R_FWD        RESOLVE FORWARD REFERENCES
(08A6) 42         |
(08D3) 15         |
(08EC) 7E 08 69   |        JMP  CMD          GO TO NEXT COMMAND
                  | *
                  | *
                  | *     ***************************************************************
                  | *                  LOOK FOR SPECIAL HANDLING OF COMMENTS
                  | *
(08EF) BD 07 30   | CMNT   JSR  NX_CHR       LOOK FOR HEADING COMMAND (*P CARD)
(08EE) 08 EF      |
(08F2) 81 50      |        CMPA #'P
(08F4) 26 00      |        BNE  CMN9
(08F6) 7C 00 35   |        INC  EJECT        IF FOUND, SET EJECT
(08F9) 39         | CMN9   RTS
(08F5) 03         |
```

```
                        **SKIP
                        *
                        *       ***************************************************************
                        *                              SEARCH OPERAND FOR X OR ,X
                        *
(08FA) DE 49            M_SRCH LDX  NXCHAR      SET INDEX TO CURRENT PLACE IN BUFFER
(08E4) 08 FA
(08FC) 96 30                   LDAA CC          CHECK NEXT CHAR FOR
(08FE) 81 58                   CMPA #'X           'X'
(0900) 26 00                   BNE  SCH5        IF NOT THEN GO TO 'CHECK FOR ,X'
(0902) A6 00                   LDAA X           IF NEXT CHAR IS
(0904) 91 27                   CMPA BLNK          BLANK
(0906) 27 00                   BEQ  SCH3          OR
(0908) 81 06 7C                CMPA EOL           END OF LINE
(090B) 27 00                   BEQ  SCH3          OR
(090D) 81 2B                   CMPA #$2B          +
(090F) 27 00                   BEQ  SCH3          OR
(0911) 81 2D                   CMPA #$2D          -
(0913) 27 00                   BEQ  SCH3          THEN GO TO LOAD MOD
(0915) 20 00                   BRA  SCH5        ELSE GO TO 'CHECK FOR ,X'
(0917) 86 58            SCH3   LDAA #'X         LOAD 'X'
(0907) 0F
(090C) 0A
(0910) 06
(0914) 02
(0919) 97 21                   STAA MOD           INTO MOD
(091B) 86 30                   LDAA #'0         LOAD DEFAULT ZERO OFFSET
(091D) 97 30                   STAA CC            INTO CURRENT CHARACTER
(091F) 20 00                   BRA  SCH9        GO TO END
(0921) A6 00            SCH5   LDAA X           CHECK FOR ,X
(0901) 1F
(0916) 0A
(0923) 08                      INX              IF NEXT CHAR IS
(0924) 91 27                   CMPA BLNK          BLANK
(0926) 27 00                   BEQ  SCH9          OR
(0928) 81 06 7C                CMPA EOL           END OF LINE
(092B) 27 00                   BEQ  SCH9          THEN GO TO END
(092D) 81 2C                   CMPA #$2C        IF IT IS A COMMA THEN CONTINUE
(092F) 26 F0                   BNE  SCH5          ELSE
(0931) A6 00                   LDAA X               LOAD CHAR
(0933) 97 21                   STAA MOD             INTO MOD
(0935) 39               SCH9   RTS               RETURN
(0927) 0D
(0920) 14
(092C) 08
                        **SKIP
                        *
                        *       ***************************************************************
                        *                        FIND VERB IN TRANSLATION TABLE
                        *
(0936) 96 22            TRNS   LDAA NAME        VERB TRANSLATION
(08AF) 09 36
(0938) 81 41                   CMPA #'A         IF VERB STARTS ABOVE A
(093A) 2C 06                   BGE  *+8           CONTINUE
(093C) 86 41                   LDAA #'A         ELSE LOAD ERROR WITH 'A'
(093E) BD 07 A7                JSR  ERRS          SIGNAL ERROR
(0941) 39                      RTS                AND RETURN
(0942) 81 5A                   CMPA #'Z         IF VERB STARTS BELOW Z
(0944) 2F 06                   BLE  *+8           CONTINUE
(0946) 86 41                   LDAA #'A         ELSE LOAD ERROR WITH 'A'
(0948) BD 07 A7                JSR  ERRS          SIGNAL ERROR
(0948) 39                      RTS                AND RETURN
(094C) 84 1F                   ANDA #$1F        STRIP OFF HIGH BITS
(094E) 4A                      DECA
(094F) 48                      ASLA             DOUBLE A-1 GIVING ENTRY INTO TBL1
(0950) CE 04 AB                LDX  #TBL1
(0953) 4D               TR01   TSTA             INCREMENT
(0954) 27 00                   BEQ  TR02          INDEX
(0956) 08                      INX                TO
(0957) 4A                      DECA               LETTER OFFSET
(0958) 20 F9                   BRA  TR01
(095A) E6 01            TR02   LDAB 1,X         LOAD
(0955) 04
(095C) D7 44                   STAB LTRL          LETTER LENGTH
(095E) E6 00                   LDAB X           MULTIPLY
(0960) 4F                      CLRA               VERB OFFSET
(0961) 58                      ASLB               BY 4 TO OBTAIN
(0962) 49                      ROLA               TABLE OFFSET (TTBL)
(0963) 58                      ASLB
(0964) 49                      ROLA
(0965) CE 04 DF                LDX  #TTBL       ADD
(0968) DF 5F                   STX  TRNOFF        TABLE OFFSET
(096A) DB 60                   ADDB TRNOFF+1      TO
(096C) 99 5F                   ADCA TRNOFF        TABLE STARTING POS
(096E) D7 60                   STAB TRNOFF+1

(0970) 97 5F                   STAA TRNOFF
(0972) CE 00 23                LDX  #NAME+1     LOAD NAME
(0975) DF 51                   STX  STR1          INTO STRING 1
                        **SKIP
                        *
                        ****   TRNS CONT.
                        *
(0977) 7D 00 44         TR03   TST  LTRL        END OF LETTER LENGTH?
(097A) 26 06                   BNE  *+8           IF NOT TRY NEXT ONE
(097C) 86 42                   LDAA #'B         ELSE LOAD ERROR 'B'
(097E) BD 07 A7                JSR  ERRS          SIGNAL ERROR - VERB NOT FOUND
(0981) 39                      RTS                AND RETURN
(0982) DE 5F                   LDX  TRNOFF
(0984) DF 53                   STX  STR2        LOAD VERB INTO STRING 2
(0986) BD 00 00                JSR  STR_CM      COMPARE VERB WITH NAME
(0989) 7D 00 2E                TST  ANSW        END IF
(098C) 23 00                   BLS  TR08          NOT EQUAL, CONTINUE
(098E) BD 00 00                JSR  OP_GEN
(0991) 7D 00 2E                TST  ANSW
(0994) 23 00                   BLS  TR08
(0996) 39                      RTS                ELSE RETURN
(0997) 08               TR08   INX              ADD 4 TO
(0980) 09
(0995) 01
(0998) 08                      INX                ADDR OF
(0999) 08                      INX                VERB
(099A) 08                      INX                IN TABLE
(099B) DF 5F                   STX  TRNOFF
(099D) CE 00 23                LDX  #NAME+1
(09A0) DF 51                   STX  STR1
(09A2) 7A 00 44                DEC  LTRL
(09A5) 20 D0                   BRA  TR03        TRY NEXT ITEM IN TABLE
                        **SKIP
                        *
                        *       ***************************************************************
                        *                        CHECK OP CODE SPECIFICS
                        *
(09A7) A6 02            OP_GEN LDAA 2,X
(098F) 09 A7
(09A9) 97 4D                   STAA OPCODE      LOAD OPCODE
(09AB) A6 03                   LDAA 3,X
(09AD) 16                      TAB
(09AE) C4 0F                   ANDB #$0F        LOAD FLAGS
(09B0) D7 37                   STAB FLAGS
(09B2) 44                      LSRA             SHIFT
(09B3) 44                      LSRA               CLASS
(09B4) 44                      LSRA               TO RIGHT
(09B5) 44                      LSRA
(09B6) 97 31                   STAA CLASS       LOAD CLASS
(09B8) D6 25                   LDAB NAME+3      B = OPERAND
(09BA) D1 27                   CMPB BLNK        IF NOT BLANK,
(09BC) 26 00                   BNE  OP03          CONTINUE
(09BE) 7D 00 37                TST  FLAGS       ELSE IF FLAG > 0
(09C1) 22 00                   BHI  OP09          GO TO END
(09C3) 81 02                   CMPA #2          IF CLASS <= 2 THEN
(09C5) 23 00                   BLS  OP08          NO MATCH
(09C7) 20 00                   BRA  OP09        ELSE GO TO END
(09C9) 81 03            OP03   CMPA #3          FOR OPERAND PRESENT, IF CLASS > 3 THEN
(09BD) 08
(09CB) 22 00                   BHI  OP08          NO MATCH
(09CD) 7D 00 37                TST  FLAGS       IF FLAG -= 0 THEN
(09D0) 22 00                   BHI  OP08          NO MATCH
(09D2) C1 41                   CMPB #'A         IF OPERAND = 'A' THEN
(09D4) 27 00                   BEQ  OP06          GO TO ADJUST CLASS
(09D6) C1 42                   CMPB #'B         IF OPERAND -= 'B'
(09D8) 26 00                   BNE  OP08          NO MATCH
(09DA) D6 4D                   LDAB OPCODE      ADJUST OPCODE
(09DC) 4D                      TSTA
(09DD) 26 00                   BNE  OP04        FOR CLASS 0, ADD $01 FOR B
(09DF) CB 01                   ADDB #$01              (PSH, PUL)
(09E1) D7 4D                   STAB OPCODE
(09E3) 20 00                   BRA  OP06          GO TO ADJUST CLASS
(09E5) 81 03            OP04   CMPA #3          ADJUST CLASS 3 AND 1,2
(09DE) 06
(09E7) 27 02                   BEQ  *+4
(09E9) C0 30                   ADDB #$30
(09EB) CB 10                   ADDB #$10
(09ED) D7 4D                   STAB OPCODE      STORE ADJUSTED OP CODE
                        **SKIP
                        *
                        ****   OP_GEN CONT.
                        *
(09EF) 4C               OP06   INCA             ADJUST CLASS
(09D5) 19
(09E4) 0A
```

```
(09F0) 85 02      |         BITA #02        SET CLASS 0 AND 3 TO 5
(09F2) 26 00      |         BNE  OP09
(09F4) 86 05      |         LDAA #5
(09F6) 97 31      |         STAA CLASS
(09F8) 20 00      |         BRA  OP09
(09FA) 7F 00 2E   | OP08    CLR  ANSW       NO MATCH
(09CC) 2D         |
(09C6) 33         |
(09D1) 28         |
(09D9) 20         |
(09FD) 39         | OP09    RTS             RETURN
(09C2) 3A         |
(09C8) 34         |
(09F3) 09         |
(09F9) 03         |
                  | *
                  | *
                  | *       ***************************************************************
                  | *                       COMPARE TWO STRINGS
                  | *
(09FE) 36         | STR_CM PSHA             SAVE A
(0987) 09 FE      |
(09FF) DF 3A      |         STX  IXS1         IX
(0A01) 9F 57      |         STS  SVST         STACK PTR
(0A03) 7F 00 2E   |         CLR  ANSW       CLEAR ANSWER (NO MATCH)
(0A06) 96 32      |         LDAA CMPLEN     SET COMPARE LENGTH
(0A08) 97 58      |         STAA TALY
(0A0A) DE 51      |         LDX  STR1       LOAD
(0A0C) 9E 53      |         LDS  STR2         STRINGS
(0A0E) 34         |         DES             DECREMENT STACK SO PULL WILL WORK
(0A0F) 32         | ST01    PULA            GET NEXT CHARACTER
(0A10) A1 00      |         CMPA X          COMPARE
(0A12) 26 00      |         BNE  ST99       IF NOT EQUAL, GO TO NOMATCH
(0A14) 08         |         INX               ELSE
(0A15) 7A 00 58   |         DEC  TALY         GO TO NEXT CHAR IF NOT DONE
(0A18) 26 F5      |         BNE  ST01
(0A1A) 7C 00 2E   |         INC  ANSW       ELSE SET ANSWER (MATCH)
(0A1D) DE 3A      | ST99    LDX  IXS1       RESTORE IX,
(0A13) 09         |
(0A1F) 9E 57      |         LDS  SVST         STACK PTR,
(0A21) 32         |         PULA              AND A
(0A22) 39         |         RTS             RETURN
                  | **SKIP
                  | *
                  | *       ***************************************************************
                  | *                       PROCESS VERB AND OPERANDS
                  | *
(0A23) 96 31      | VERB    LDAA CLASS      A = VERB CLASS
(08E7) 0A 23      |
(0A25) D6 4D      |         LDAB OPCODE     B = OP CODE
(0A27) 81 03      |         CMPA #3         IF NOT CLASS 4 OR 5 THEN
(0A29) 23 00      |         BLS  VB02         BRANCH TO MEMORY REQUIRED ITEMS
(0A2B) D7 2F      |         STAB BYTOUT     BYTE OUT = OP CODE
(0A2D) BD 07 B8   |         JSR  CALCWR     WRITE OP CODE
(0A30) 96 31      |         LDAA CLASS
(0A32) 81 04      |         CMPA #&4        IF CLASS 4 VERB,
(0A34) 26 03      |         BNE  *+5          GET
(0A36) BD 00 00   |         JSR  RELATV       RELATIVE ADDRESS
(0A39) 39         |         RTS             END OF CLASS 4 AND 5 VERBS
(0A3A) 96 21      | VB02    LDAA  MOD       MEMORY REQUIRED, A = VERB MODIFIER
(0A2A) 0F         |
(0A3C) 81 23      |         CMPA #'#        IF MOD = '#'
(0A3E) 26 00      |         BNE  VB03         CALL
(0A40) BD 00 00   |         JSR  IMED_M       IMEDIATE PROCESSING
(0A43) 39         |         RTS
(0A44) 81 58      | VB03    CMPA #'X        IF MOD = 'X'
(0A3F) 04         |
(0A46) 26 00      |         BNE  VB20
(0A48) CB 20      |         ADDB #$20         ADJUST OP CODE
(0A4A) BD 00 00   |         JSR  INDXED       CALL INDEXED PROCESSING
(0A4D) 39         |         RTS
(0A4E) 91 27      | VB20    CMPA BLNK       IF MOD = ' '
(0A47) 06         |
(0A50) 26 00      |         BNE  VB99
(0A52) CB 10      |         ADDB #$10         ADJUST OP CODE
(0A54) BD 00 00   |         JSR  DFLT         CALL DEFAULT ADDRESSING PROCEDURE
(0A57) 39         |         RTS
(0A58) 86 44      | VB99    LDAA #'D        IF MOD WAS NOT FOUND ABOVE,
(0A51) 06         |
(0A5A) BD 07 A7   |         JSR  ERRS         SIGNAL ERROR,
(0A5D) 39         |         RTS               AND RETURN - END OF PROCESS VERB
                  | **SKIP
                  | *
                  | *       ***************************************************************
                  | *                       GET A RELATIVE ADDRESS SHIFT
                  | *
(0A5E) BD 00 00   | RELATV JSR  GENADR      GET OPERAND
(0A37) 0A 5E      |
(0A61) 7D 00 61   |         TST  UNRESL     IF NOT UNRESOLVED THEN
(0A64) 23 00      |         BLS  RL10         CONTINUE
(0A66) 96 2A      |         LDAA AD01+1       ELSE WRITE
(0A68) 97 2F      |         STAA BYTOUT       OUT A DUMMY
(0A6A) BD 07 B8   |         JSR  CALCWR       ARGUMENT
(0A6D) 39         |         RTS             RETURN
(0A6E) CE 00 4E   | RL10    LDX  #PGMCNT      RELATIVE FROM OLD DATA NAME,
(0A65) 08         |
(0A71) 96 29      |         LDAA AD01       LOAD RELATIVE VALUE
(0A73) D6 2A      |         LDAB AD01+1       WITH
(0A75) E0 01      |         SUBB 1,X          HOLD ADDR
(0A77) A2 00      |         SBCA X            MINUS
(0A79) C0 01      |         SUBB #$01         PGM COUNT
(0A7B) 92 48      |         SBCA NULL         MINUS
(0A7D) D7 2F      |         STAB BYTOUT       ONE
(0A7F) BD 00 00   |         JSR  BRTST      CHECK FOR ANSWER > 127, OR <-128
(0A82) 01         |         NOP
(0A83) 01         |         NOP
(0A84) BD 07 B8   |         JSR  CALCWR
(0A87) 39         |         RTS             RETURN
                  | *
                  | *
                  | *       ***************************************************************
                  | *                       CHECK RELATIVE ADDRESS RANGE
                  | *
(0A88) 91 48      | BRTST   CMPA NULL       IF HIGH BYTE
(0A80) 0A 88      |
(0A8A) 27 06      |         BEQ  *+8          NOT = 00 OR FF THEN
(0A8C) 81 FF      |         CMPA #$FF         GO TO ERROR
(0A8E) 27 02      |         BEQ  *+4
(0A90) 20 00      |         BRA  RLER
(0A92) 98 2F      |         EORA BYTOUT     IF SIGNS OF BYTES ARE THE SAME THEN
(0A94) 2A 00      |         BPL  BRT9         CONTINUE
(0A96) 86 47      | RLER    LDAA #'G
(0A91) 04         |
(0A98) BD 07 A7   |         JSR  ERRS       WRITE OUT ERROR MSG
(0A98) DE 4E      |         LDX  PGMCNT
(0A9D) 7D 00 38   |         TST  FWD
(0AA0) 22 01      |         BHI  *+3
(0AA2) 09         |         DEX             CORRECT PGMCNT FOR BRANCH INSTRUCTIONS
(0AA3) DF 4E      |         STX  PGMCNT
(0AA5) DF 62      |         STX  OUTADR
(0AA7) 39         | BRT9    RTS             RETURN
(0A95) 11         |
                  | **SKIP
                  | *
                  | *       ***************************************************************
                  | *                       PROCESS IMMEDIATE ADDRESSING ITEMS (M = #)
                  | *
(0AA8) 96 31      | IMED_M LDAA CLASS       CHECK CLASS
(0A41) 0A A8      |
(0AAA) 81 01      |         CMPA #$01         FOR 1
(0AAC) 27 06      |         BEQ  *+8          BEING OK
(0AAE) 86 43      |         LDAA #'C          IF NOT,
(0AB0) BD 07 A7   |         JSR  ERRS         SIGNAL ERROR
(0AB3) 39         |         RTS             AND RETURN
(0AB4) 37         |         PSHB            SAVE B (OP CODE)
(0AB5) BD 00 00   |         JSR  GENADR     GENERATE ADDESS
(0AB8) 33         |         PULB            RESTORE B
(0AB9) D7 2F      |         STAB BYTOUT     WRITE OUT
(0ABB) BD 07 B8   |         JSR  CALCWR       OP CODE
(0ABE) BD 00 00   |         JSR  IMD2       ELSE PUT OUT BYTE(S) OF OPERAND
(0AC1) 39         |         RTS               RETURN
                  | *
                  | *
                  | *       ***************************************************************
                  | *                       WRITE OUT ONE OR TWO BYTE OPERANDS
                  | *
(0AC2) 96 22      | IMD2    LDAA NAME       IF OPERAND
(0ABF) 0A C2      |
(0AC4) 81 27      |         CMPA #''          DOES NOT START WITH ' THEN
(0AC6) 26 00      |         BNE  IMD3         CONTINUE
(0AC8) 96 23      |         LDAA NAME+1     ELSE WRITE OUT LITERAL BYTE
(0ACA) 20 00      |         BRA  IMD5
(0ACC) 7D 00 37   | IMD3    TST  FLAGS      IF NOT TWO BYTE REGISTER INSTRUCTION THEN
(0AC7) 04         |
```

```
(0ACF) 23 00      |        BLS  IMD4          CONTINUE
(0AD1) 96 29      |        LDAA AD01        ELSE WRITE OUT
(0AD3) 97 2F      |        STAA BYTOUT        AD01
(0AD5) BD 07 B8   |        JSR  CALCWR
(0AD8) 96 2A      | IMD4   LDAA AD01+1      WRITE OUT
(0ADO) 07         |
(0ADA) 97 2F      | IMD5   STAA BYTOUT        AD01+1
(0ACB) 0E         |
(0ADC) BD 07 B8   |        JSR  CALCWR
(0ADF) 39         |        RTS                RETURN
                  | **SKIP
                  | *
                  | *       ***************************************************************
                  | *                          PROCESS INDEXED ADDRESSING (M = X)
                  | *
(0AE0) 37         | INDXED PSHB             SAVE B (OP CODE)
(0A4B) 0A E0      |
(0AE1) BD 00 00   |        JSR  GENADR      GENERATE OFFSET
(0AE4) 33         |        PULB             RESTORE B
(0AE5) D7 2F      |        STAB BYTOUT
(0AE7) BD 07 B8   |        JSR  CALCWR      WRITE OUT OP CODE
(0AEA) 96 2A      |        LDAA AD01+1      WRITE OUT OFFSET
(0AEC) 97 2F      |        STAA BYTOUT
(0AEE) BD 07 B8   |        JSR  CALCWR
(0AF1) 39         |        RTS
                  | *
                  | *
                  | *       ***************************************************************
                  | *                  PROCESS DEFAULT ADDRESSING FROM SYMB TABL (M = ' ')
                  | *
(0AF2) 37         | DFLT   PSHB             SAVE B (OP CODE)
(0A55) 0A F2      |
(0AF3) BD 00 00   |        JSR  GENADR        GET NAME AND ADDRESS IN OPERAND
(0AF6) 33         |        PULB             RESTORE B
(0AF7) 7F 00 5B   |        CLR  TALY
(0AFA) 7D 00 29   |        TST  AD01        IF EXTENDED ADDR REQUIRED
(0AFD) 26 00      |        BNE  DFL0
(0AFF) 7D 00 61   |        TST  UNRESL        UNRESOLVED
(0B02) 22 00      |        BHI  DFL0
(0B04) 96 31      |        LDAA CLASS
(0B06) 81 03      |        CMPA #3            OR CLASS THREE VERB THEN
(0B08) 27 00      |        BEQ  DFL0
(0B0A) 20 00      |        BRA  DFL1
(0B0C) CB 20      | DFL0   ADDB #$20          ADJUST OP CODE FOR EXTENDED
(0AFE) 0D         |
(0B03) 08         |
(0B09) 02         |
(0B0E) 7C 00 5B   |        INC  TALY
(0B11) D7 2F      | DFL1   STAB BYTOUT      WRITE OUT
(0B08) 05         |
(0B13) BD 07 B8   |        JSR  CALCWR        OP CODE
(0B16) 7D 00 5B   |        TST  TALY
(0B19) 27 00      |        BEQ  DFL4
(0B1B) D6 29      |        LDAB AD01        WRITE
(0B1D) D7 2F      |        STAB BYTOUT        OUT
(0B1F) BD 07 B8   |        JSR  CALCWR        HIGH ORDER ADDRESS BYTE
(0B22) D6 2A      | DFL4   LDAB AD01+1      WRITE OUT
(0B1A) 07         |
(0B24) D7 2F      |        STAB BYTOUT        LOW ORDER
(0B26) BD 07 B8   |        JSR  CALCWR        ADDRESS BYTE
(0B29) 39         |        RTS              END       **********
                  | **SKIP
                  | *
                  | *       ***************************************************************
                  | *              GENERATE AN ADDRESS FROM A NAME FROM THE INPUT STRING
                  | *
(0B2A) 7F 00 41   | GENADR CLR  LAB_SW        CLEAR LABEL SWITCH
(0A5F) 0B 2A      |
(0AB6) 0B 2A      |
(0AE2) 0B 2A      |
(0AF4) 0B 2A      |
(0B2D) BD 00 00   |        JSR  GETADR        GET OPERAND ITEM
(0B30) BD 00 00   |        JSR  GEN8          RESOLVE SYMBOLS
(0B33) 96 30      | GEN1   LDAA CC            IF NEXT CHAR
(0B35) 81 2B      |        CMPA #$2B            IS A '+' THEN
(0B37) 27 00      |        BEQ  GEN2            GO TO OFFSET ADJUST
(0B39) 81 2D      |        CMPA #$2D          IF NEXT CHAR IS A '-' THEN
(0B3B) 27 00      |        BEQ  GEN2            GO TO OFFSET ADJUST
(0B3D) 39         |        RTS                ELSE RETURN
(0B3E) 97 26      | GEN2   STAA SIGN          SAVE SIGN OF OFFSET
(0B38) 05         |
(0B3C) 01         |
(0B40) DE 29      |        LDX  AD01          SAVE
(0B42) DF 2B      |        STX  SAVADR          ADDR-01
(0B44) BD 07 30   |        JSR  NX_CHR

(0B47) BD 00 00   |        JSR  GETADR        GET OPERAND ITEM
(0B4A) BD 00 00   |        JSR  GEN8          RESOLVE SYMBOLS
(0B4D) 96 26      |        LDAA SIGN          IF SIGN
(0B4F) 81 2B      |        CMPA #$2B            WAS A '+' THEN
(0B51) 27 00      |        BEQ  GEN3            GO TO ADDITION CORRECTION
(0B53) 96 2C      |        LDAA SAVADR+1    SUBTRACTION CORRECTION:
(0B55) 90 2A      |        SUBA AD01+1        SUBTRACT OFFSET FROM SAVED ADDR
(0B57) 97 2A      |        STAA AD01+1
(0B59) 96 2B      |        LDAA SAVADR
(0B5B) 92 29      |        SBCA AD01
(0B5D) 97 29      |        STAA AD01
(0B5F) 20 D2      |        BRA  GEN1          TRY ANOTHER TERM
(0B61) 96 2C      | GEN3   LDAA SAVADR+1    ADDITION CORRECTION:
(0B52) 0E         |
(0B63) 9B 2A      |        ADDA AD01+1        ADD OFFSET TO SAVED ADDR
(0B65) 97 2A      |        STAA AD01+1
(0B67) 96 2B      |        LDAA SAVADR
(0B69) 99 29      |        ADCA AD01
(0B6B) 97 29      |        STAA AD01
(0B6D) 20 C4      |        BRA  GEN1          TRY ANOTHER TERM
                  | **SKIP
                  | *
                  | *       ***************************************************************
                  | *                              RESOLVE SYMBOL ADDRESSES
                  | *
(0B6F) 7D 00 20   | GEN8   TST  ERORSW        IF NO ERRORS,
(0B31) 0B 6F      |
(0B4B) 0B 6F      |
(0B72) 22 00      |        BHI  GEN9
(0B74) 7D 00 39   |        TST  HEX             NOT HEX,
(0B77) 22 00      |        BHI  GEN9
(0B79) 7D 00 2D   |        TST  ALPHA           AND ALPHA THEN
(0B7C) 23 00      |        BLS  GEN9
(0B7E) BD 00 00   |        JSR  SYMB_S          CALL SYMBOL RESOLVE
(0B81) 39         | GEN9   RTS
(0B73) 0D         |
(0B78) 08         |
(0B7D) 03         |
                  | **SKIP
                  | *
                  | *       ***************************************************************
                  | *                              GET ADDRESS FOR ITEM IN OPERAND
                  | *
(0B82) 7C 00 2D   | GETADR INC  ALPHA         SET ALPHA AND HEX
(0B2E) 0B 82      |
(0B48) 0B 82      |
(0B85) 7C 00 39   |        INC  HEX             SWITCHES ON
(0B88) 7F 00 29   |        CLR  AD01          CLEAR FINAL ADDRESS
(0B8B) 7F 00 2A   |        CLR  AD01+1
(0B8E) 96 30      |        LDAA CC
(0B90) 81 24      |        CMPA #'$           IF ADDR NOT HEX
(0B92) 26 00      |        BNE  GAD1            THEN CONTINUE
(0B94) BD 07 30   |        JSR  NX_CHR        ELSE DROP $ CHAR
(0B97) BD 07 65   |        JSR  LD_ITM          LOAD ADDR
(0B9A) BD 00 00   |        JSR  CVTHEX          CONVERT TO HEX
(0B9D) 39         |        RTS                  AND RETURN
(0B9E) 7F 00 39   | GAD1   CLR  HEX           TURN HEX OFF
(0B93) 0A         |
(0BA1) 81 26      |        CMPA #'&           TEST NEXT CHAR FOR &
(0BA3) 26 03      |        BNE  *+5           IF NOT CONTINUE
(0BA5) BD 07 30   |        JSR  NX_CHR        ELSE DROP IT (NUMERIC IS ASSUMED DECIMAL)
(0BA8) BD 07 65   |        JSR  LD_ITM        LOAD NEXT ITEM INTO NAME
(0BAB) CE 00 22   |        LDX  #NAME         TEST FIRST CHAR
(0BAE) A6 00      |        LDAA X               IF NOT NUMERIC
(0BB0) 91 27      |        CMPA BLNK            BUT NOT NULL OPERAND
(0BB2) 27 00      |        BEQ  GAD2
(0BB4) 81 30      |        CMPA #'0
(0BB6) 2B 00      |        BMI  GAD5
(0BB8) 81 39      |        CMPA #'9
(0BBA) 22 00      |        BHI  GAD5            THEN GO TO NON NUMERIC CODE
                  | *  NUMERIC OPERAND
(0BBC) 91 27      | GAD2   CMPA BLNK          IF AT END OF ITEM
(0BB3) 08         |
(0BBE) 27 00      |        BEQ  GAD4            GO TO END OF NUMERIC
(0BC0) 8C 00 26   |        CPX  #NAME+4
(0BC3) 27 00      |        BEQ  GAD4
(0BC5) 85 30      |        BITA #$30          CHECK FOR VALID
(0BC7) 26 06      |        BNE  *+8             CHARACTERS
(0BC9) 86 4A      |        LDAA #'J
(0BCB) BD 07 A7   |        JSR  ERRS
(0BCE) 39         |        RTS
(0BCF) 84 0F      |        ANDA #$0F          DROP INPUT'S LEFT 4 BITS
(0BD1) A7 00      |        STAA X               AND REPLACE INTO NAME
(0BD3) C6 09      |        LDAB #9            MULTIPLY AD01 BY 10
```

```
(08D5)  D7 5B      |        STAB TALY
(08D7)  D6 2A      |        LDAB AD01+1
(08D9)  96 29      |        LDAA AD01
(08DB)  DB 2A      | GAD3   ADDB AD01+1
(08DD)  99 29      |        ADCA AD01
(08DF)  7A 00 5B   |        DEC  TALY
(08E2)  26 F7      |        BNE  GAD3
                   | **SKIP
                   | *  GETADR CONT
                   | *
(08E4)  EB 00      |        ADDB X          ADD NEW NUMBER TO AD01
(08E6)  99 48      |        ADCA NULL
(08E8)  D7 2A      |        STAB AD01+1
(08EA)  97 29      |        STAA AD01
(08EC)  08         |        INX             GO TO
(08ED)  A6 00      |        LDAA X            NEXT
(08EF)  20 CB      |        BRA  GAD2         CHAR
(08F1)  7F 00 2D   | GAD4   CLR  ALPHA      TURN OFF ALPHA
(08BF)  31         |
(08C4)  2C         |
(08F4)  39         |        RTS             RETURN
(08F5)  CE 06 78   | GAD5   LDX  #XOPND     COMPARE NAME WITH *X   *
(08B7)  3D         |
(08BB)  39         |
(08F8)  DF 51      |        STX  STR1
(08FA)  CE 00 22   |        LDX  #NAME
(08FD)  DF 53      |        STX  STR2
(08FF)  BD 09 FE   |        JSR  STR_CM
(0C02)  7D 00 2E   |        TST  ANSW       IF NOT EQUAL THEN
(0C05)  23 00      |        BLS  GAD6         CONTINUE
(0C07)  DE 48      |        LDX  NULL       ELSE
(0C09)  DF 22      |        STX  NAME         CLEAR NAME
(0C08)  DF 24      |        STX  NAME+2
(0C0D)  7F 00 2D   |        CLR  ALPHA      TURN OFF ALPHA SWITCH
(0C10)  39         |        RTS             RETURN
(0C11)  CE 06 74   | GAD6   LDX  #ASTK      COMPARE NAME WITH **   *
(0C06)  0A         |
(0C14)  DF 51      |        STX  STR1
(0C16)  CE 00 22   |        LDX  #NAME
(0C19)  DF 53      |        STX  STR2
(0C1B)  BD 09 FE   |        JSR  STR_CM
(0C1E)  7D 00 2E   |        TST  ANSW       IF NOT EQUAL THEN
(0C21)  23 00      |        BLS  GAD9         CONTINUE
(0C23)  DE 4E      |        LDX  PGMCNT     ELSE GET PGMCNT
(0C25)  96 31      |        LDAA CLASS        IF VERB CLASS
(0C27)  81 04      |        CMPA #&4            NOT = 4 THEN
(0C29)  26 01      |        BNE  *+3            CONTINUE
(0C2B)  09         |        DEX               ELSE DECREMENT PGMCNT
(0C2C)  DF 29      |        STX  AD01         OUTPUT ADDRESS
(0C2E)  7F 00 2D   |        CLR  ALPHA        SET ALPHA OFF
(0C31)  39         | GAD9   RTS             RETURN
(0C22)  0E         |
                   | **SKIP
                   | *
                   | *
                   | *     ****************************************************************
                   | *                    CONVERT NAME TO HEX IN AD01, AD01+1
                   | *
(0C32)  CE 00 22   | CVTHEX LDX  #NAME      START AT NAMF
(089B)  0C 32      |
(0C35)  A6 00      | CVT1   LDAA X
(0C37)  91 27      |        CMPA BLNK       IF CHAR = BLANK THEN
(0C39)  27 00      |        BEQ  CVT9         GO TO END
(0C3B)  8C 00 26   |        CPX  #NAME+4    IF AT END OF INPUT THEN
(0C3E)  27 00      |        BEQ  CVT9         GO TO END
(0C40)  81 30      |        CMPA #'0        IF CHAR < 0
(0C42)  2B 00      |        BMI  CVT3         OR
(0C44)  81 46      |        CMPA #'F          > F
(0C46)  22 00      |        BHI  CVT3         THEN SIGNAL ERROR
(0C48)  20 06      |        BRA  *+8        ELSE CONTINUE
(0C4A)  86 4A      | CVT3   LDAA #'J
(0C43)  06         |
(0C47)  02         |
(0C4C)  BD 07 A7   |        JSR  ERRS
(0C4F)  39         |        RTS
(0C50)  81 41      |        CMPA #'A        IF CHAR < A THEN
(0C52)  2B 02      |        BMI  *+4          CONTINUE
(0C54)  8B 09      |        ADDA #$09       ELSE ADD 9 TO LOW NIBBLE
(0C56)  84 0F      |        ANDA #$0F       DROP HIGH NIBBLE
(0C58)  A7 00      |        STAA X          RESTORE CHARACTER TO NAME
(0C5A)  96 29      |        LDAA AD01       MULTIPLY
(0C5C)  D6 2A      |        LDAB AD01+1       CURRENT VALUE
(0C5E)  58         |        ASLB                BY 16
(0C5F)  49         |        ROLA
(0C60)  58         |        ASLB
(0C61)  49         |        ROLA
(0C62)  58         |        ASLB
(0C63)  49         |        ROLA
(0C64)  58         |        ASLB
(0C65)  49         |        ROLA
(0C66)  EB 00      |        ADDB X          ADD VALUE OF THIS CHARACTER
(0C68)  D7 2A      |        STAB AD01+1     RESTORE TOTAL VALUE
(0C6A)  97 29      |        STAA AD01
(0C6C)  08         |        INX             GO TO NEXT CHAR
(0C6D)  20 C6      |        BRA  CVT1
(0C6F)  39         | CVT9   RTS             RETURN
(0C3A)  34         |
(0C3F)  2F         |
                   | **SKIP
                   | *
                   | *     ****************************************************************
                   | *                  PROCESS A NAME AGAINST THE SYMBOL TABLE
                   | *                  IF NOT THERE, SET IT UP.  RETURN ADDRESS
                   | *
(0C70)  96 22      | SYMB_S LDAA NAME       IF LABEL STARTS WITH A ' (QUOTE) THEN
(089D)  0C 70      |
(0B7F)  0C 70      |
(0C72)  81 27      |        CMPA #''          RETURN
(0C74)  26 01      |        BNE  *+3
(0C76)  39         |        RTS
(0C77)  81 40      |        CMPA #$40       IF LABEL STARTS WITH A CODE LESS THAN ALPHA THEN
(0C79)  22 06      |        BHI  *+8          SIGNAL ERROR
(0C78)  86 4B      |        LDAA #'K
(0C7D)  BD 07 A7   |        JSR  ERRS
(0C80)  39         |        RTS
(0C81)  DE 6D      |        LDX  STBL       SET INDEX TO TABLE ENTRY POINT
(0C83)  DF 59      | SM01   STX  TBLS       IF DIFFERENCE BETWEEN
(0C85)  96 48      |        LDAA NOWSTK+1     CURRENT LOCATION IN TABLE (TBLS)
(0C87)  90 5A      |        SUBA TBLS+1       AND BOTTOM OF BGN STACK (NOWSTK)
(0C89)  D6 47      |        LDAB NOWSTK       IS LESS THAN 6 CONTINUE
(0C8B)  D0 59      |        SUBB TBLS
(0C8D)  2B 00      |        BMI  SM02
(0C8F)  2E 00      |        BGT  SM03
(0C91)  81 06      |        CMPA #6
(0C93)  22 00      |        BHI  SM03
(0C95)  86 45      | SM02   LDAA #'E        ELSE
(0C8E)  06         |
(0C97)  BD 07 A7   |        JSR  ERRS       PUT OUT TABLE FULL ERROR (E)
(0C9A)  39         |        RTS               AND RETURN
(0C98)  DF 53      | SM03   STX  STR2       LOAD SYMBOL INTO STRING 2
(0C90)  0A         |
(0C94)  06         |
(0C9D)  CE 00 22   |        LDX  #NAME      LOAD NAME
(0CA0)  DF 51      |        STX  STR1         INTO STRING 1
(0CA2)  BD 09 FE   |        JSR  STR_CM     CALL STRING COMPARE
(0CA5)  7D 00 2E   |        TST  ANSW       IF NO HIT THEN
(0CA8)  23 00      |        BLS  NOHIT        GO TO NO-HIT LOOP
(0CAA)  DE 59      |        LDX  TBLS       RESTORE INDEX TO SYMB
(0CAC)  7D 00 41   |        TST  LAB_SW     IF LABEL,
(0CAF)  22 00      |        BHI  SM10         BRANCH TO LABEL PROC
(0CB1)  EE 04      |        LDX  4,X        LOAD ADDRESS -- (NOT NEW, NOT LABEL)
(0CB3)  DF 29      |        STX  AD01         INTO HOLD ADDRESS
(0CB5)  9C 48      |        CPX  NULL       IF ADDER = 0 SET
(0CB7)  26 03      |        BNE  *+5          UNRESOLVED
(0CB9)  BD 00 00   |        JSR  S_FWD        ADDR.
(0CBC)  39         |        RTS             RETURN
                   | **SKIP
                   | *
                   | ***** SYMB_S CONT
                   | *
(0CBD)  EE 04      | SM10   LDX  4,X        LABEL PROC
(0CB0)  0C         |
(0CBF)  9C 48      |        CPX  NULL       IF ADDR NOT = 0 THEN
(0CC1)  27 06      |        BEQ  *+8          GO TO ERROR ROUTINE
(0CC3)  86 46      |        LDAA #'F
(0CC5)  BD 07 A7   |        JSR  ERRS
(0CC8)  39         |        RTS
(0CC9)  7C 00 38   |        INC  FWD        SIGNAL ADDRESSES TO RESOLVE
(0CCC)  DE 59      |        LDX  TBLS
(0CCE)  96 4E      |        LDAA PGMCNT
(0CD0)  A7 04      |        STAA 4,X
(0CD2)  96 4F      |        LDAA PGMCNT+1
(0CD4)  A7 05      |        STAA 5,X
(0CD6)  DF 42      |        STX  LSTLAB
(0CD8)  EE 04      |        LDX  4,X        LOAD ADDRESS
(0CDA)  DF 29      |        STX  AD01         INTO HOLD ADDRESS
(0CDC)  39         |        RTS             RETURN
(0CDD)  DE 59      | NOHIT  LDX  TBLS       TEST FOR EMPTY SLOT IN TABLE
(0CA9)  33         |
(0CDF)  6D 00      |        TST  X            IF NOT, CONTINUE
```

```
(0CE1)  26 00      |        BNE  SM90
(0CE3)  BD 00 00   |        JSR  SMBLOD     ELSE LOAD ITEM
(0CE6)  7D 00 41   |        TST  LAB_SW     IF NOT
(0CE9)  22 03      |        BHI  *+5          LABEL
(0CEB)  BD 00 00   |        JSR  S_FWD        THEN SET UNRESOLVED FWD ADDRESS
(0CEE)  39         |        RTS             RETURN
(0CEF)  DE 59      | SM90   LDX  TBLS       TRY NEXT
(0CE2)  0C         |
(0CF1)  08         |        INX               ITEM.
(0CF2)  08         |        INX               INCR IX BY 6
(0CF3)  08         |        INX
(0CF4)  08         |        INX
(0CF5)  08         |        INX
(0CF6)  08         |        INX
(0CF7)  20 8A      |        BRA  SM01       GO TO MAIN SYMB LOOP
                   | **SKIP
                   | *
                   | *       ***************************************************************
                   | *                   LOAD A NAME INTO THE SYMBOL TABLE
                   | *
(0CF9)  DE 59      | SMBLOD LDX  TBLS       STORE LOCATION OF NEXT
(0CE4)  0C F9      |
(0CFB)  08         |        INX               SLOT IN NEXT SLOT POINTER (NOWPOS)
(0CFC)  08         |        INX
(0CFD)  08         |        INX
(0CFE)  08         |        INX
(0CFF)  08         |        INX
(0D00)  08         |        INX
(0D01)  DF 45      |        STX  NOWPOS
(0D03)  DE 59      |        LDX  TBLS
(0D05)  9F 57      |        STS  SVST       SAVE STACK PTR
(0D07)  9E 22      |        LDS  NAME       LOAD
(0D09)  AF 00      |        STS  X            NAME
(0D0B)  9E 24      |        LDS  NAME+2
(0D0D)  AF 02      |        STS  2,X
(0D0F)  6F 04      |        CLR  4,X        CLEAR NEW ADDRESS
(0D11)  6F 05      |        CLR  5,X
(0D13)  7D 00 41   |        TST  LAB_SW     IF NOT LABEL,
(0D16)  23 00      |        BLS  SMB5         BRANCH TO ENDING
(0D18)  9E 4E      |        LDS  PGMCNT     LOAD
(0D1A)  AF 04      |        STS  4,X          ADDR
(0D1C)  DF 42      |        STX  LSTLAB     SAVE LABEL ADDRESS
(0D1E)  6F 06      | SMB5   CLR  6,X        CLEAR THE NEXT SLOT IN THE TABLE
(0D17)  06         |
(0D20)  EE 04      |        LDX  4,X        STORE ADDR
(0D22)  DF 29      |        STX  AD01         IN HOLD ADDR
(0D24)  9E 57      |        LDS  SVST       RESTORE STACK PTR
(0D26)  39         |        RTS             RETURN
                   | **SKIP
                   | *
                   | *       ***************************************************************
                   | *                   SET FOREWARD UNRESOLVED ADDRESS
                   | *
(0D27)  DE 67      | S_FWD  LDX  FTBL       SET IX TO FWD BRANCH TABLE
(0CBA)  0D 27      |
(0CEC)  0D 27      |
(0D29)  9C 6B      | SFW1   CPX  TB2TOP     IF NOT AT TOP OF TABLE
(0D2B)  26 06      |        BNE  *+8          THEN CONTINUE
(0D2D)  86 49      |        LDAA #'I        ELSE
(0D2F)  BD 07 A7   |        JSR  ERRS         WRITE OUT TABLE ERROR AND
(0D32)  39         |        RTS               RETURN
(0D33)  9C 69      |        CPX  TB2WNG     IF NOT ONE FROM TOP OF TABLE
(0D35)  26 00      |        BNE  SFW3         THEN CONTINUE
(0D37)  86 48      |        LDAA #'H        ELSE
(0D39)  BD 07 A7   |        JSR  ERRS         WARN OF NEAR TABLE ERROR,
(0D3C)  7F 00 20   |        CLR  ERORSW       CLEAR ERROR SWITCH
(0D3F)  E6 00      | SFW3   LDAB X          TEST FOR EMPTY SLOT
(0D36)  08         |
(0D41)  26 00      |        BNE  SFW5         AND IF NOT, CONTINUE
(0D43)  E6 01      |        LDAB 1,X        ELSE
(0D45)  26 00      |        BNE  SFW5         GO TO SET ADDRESS
(0D47)  20 00      |        BRA  SFW7
(0D49)  08         | SFW5   INX               INCTREMENT
(0D42)  06         |
(0D46)  02         |
(0D4A)  08         |        INX               IX BY 5
(0D4B)  08         |        INX
(0D4C)  08         |        INX
(0D4D)  08         |        INX
(0D4E)  20 D9      |        BRA  SFW1         GO TO MAIN LOOP
                   | **SKIP
                   | *
                   | *****  S_FWD CONT.
                   | *
(0D50)  9& 21      | SFW7   LDAA MOD        IF INDEXED ADDRESSING THEN
(0D48)  07         |
(0D52)  81 58      |        CMPA #'X          SET TYPE AS -1
(0D54)  26 02      |        BNE  *+4
(0D56)  6A 04      |        DEC  4,X
(0D58)  81 23      |        CMPA #'#        IF IMMEDIATE ADDRESSING AND
(0D5A)  26 00      |        BNE  SFW8         NOT A TWO BYTE VERB THEN
(0D5C)  7D 00 37   |        TST  FLAGS        SET TYPE AS -1
(0D5F)  22 00      |        BHI  SFW8
(0D61)  6A 04      |        DEC  4,X
(0D63)  96 31      | SFW8   LDAA CLASS      IF RELATIVE ADDRESSING THEN
(0D5B)  07         |
(0D60)  02         |
(0D65)  81 04      |        CMPA #4           SET TYPE AS +1
(0D67)  26 02      |        BNE  *+4
(0D69)  6C 04      |        INC  4,X
(0D6B)  9F 57      |        STS  SVST       SAVE STACK POINTER
(0D6D)  9E 59      |        LDS  TBLS       LOAD ADDRESS OF SYMB IN SYMB TABL
(0D6F)  AF 00      |        STS  X            INTO FWD ADDRESS TABLE
(0D71)  9E 4E      |        LDS  PGMCNT       (IF OPCODE IS PRESENT, ADDR = ADDR + 1)
(0D73)  81 0F      |        CMPA #$F
(0D75)  27 00      |        BEQ  SFW9
(0D77)  31         |        INS
(0D78)  AF 02      | SFW9   STS  2,X
(0D76)  01         |
(0D7A)  9E 57      |        LDS  SVST       RESTORE STACK POINTER
(0D7C)  7C 00 61   |        INC  UNRESL
(0D7F)  39         |        RTS             RETURN
                   | **SKIP
                   | *       ***************************************************************
                   | *                   RESOLVE FOREWARD ADDRESS REFERENCES
                   | *
(0D80)  7D 00 38   | R_FWD  TST  FWD        IF NO FORWARD REFERENCES TO RESOLVE
(08EA)  0D 80      |
(0D83)  22 01      |        BHI  *+3          RETURN, ELSE
(0D85)  39         |        RTS
(0D86)  DE 67      |        LDX  FTBL       INITIALIZE IX TO START OF TABLE
(0D88)  DF 3C      |        STX  IXS2
(0D8A)  7D 00 20   | RF01   TST  ERORSW     MAIN LOOP
(0D8D)  22 00      |        BHI  RF02       IF ERROR, RETURN
(0D8F)  9C 6B      |        CPX  TB2TOP
(0D91)  26 00      |        BNE  RF03       IF NOT AT TOP OF TABLE, CONTINUE
(0D93)  7F 00 38   |        CLR  FWD        ELSE CLEAR FWD SWITCH
(0D96)  7F 00 34   |        CLR  DELETE       DELETE SWITCH
(0D99)  7F 00 36   |        CLR  ENDFL        ENDFL SWITCH
(0D9C)  39         | RF02   RTS               AND RETURN - END OF SEARCH
(0D8E)  0D         |
(0D9D)  7D 00 36   | RF03   TST  ENDFL      IF END OF BLOCK,
(0D92)  0A         |
(0DA0)  23 00      |        BLS  RF05
(0DA2)  BD 00 00   |        JSR  RF70       CALL DUMP-UNRESOLVED
(0DA5)  20 00      |        BRA  RF80       GO TO INCREMENT LOOP
(0DA7)  DE 3C      | RF05   LDX  IXS2       COMPARE FWD ADDR OFFSET
(0DA1)  05         |
(0DA9)  A6 00      |        LDAA X          IF EITHER DOES NOT MATCH WITH LAST LABEL
(0DAB)  E6 01      |        LDAB 1,X          THEN GO TO INCREMENT LOOP
(0DAD)  91 42      |        CMPA LSTLAB
(0DAF)  26 00      |        BNE  RF80
(0DB1)  D1 43      |        CMPB LSTLAB+1
(0DB3)  26 00      |        BNE  RF80
(0DB5)  DE 4E      |        LDX  PGMCNT     ELSE FOUND HIT TO RESOLVE
(0DB7)  DF 55      |        STX  SVPC       SAVE PGM_CNT
(0DB9)  BD 07 F1   |        JSR  WRITE      WRITE OUT CURRENT OUTPUT
(0DBC)  DE 3C      |        LDX  IXS2       LOAD PGM-CNT
(0DBE)  EE 02      |        LDX  2,X          WITH
(0DC0)  DF 4E      |        STX  PGMCNT       PGM-CNT IN SYMBOL TABLE
(0DC2)  DE 3C      |        LDX  IXS2       TEST TYPE INDICATOR
(0DC4)  6D 04      |        TST  4,X
(0DC6)  2B 00      |        BMI  RF10         IF = -1 HANDLE ONE BYTE OUT
(0DC8)  26 00      |        BNE  RF20         IF = +1 HANDLE RELATIVE ITEMS
(0DCA)  DE 42      |        LDX  LSTLAB     WRITE OUT
(0DCC)  E6 04      |        LDAB 4,X          ADDRESS
(0DCE)  D7 2F      |        STAB BYTOUT       IN SYMBOL TABLE
(0DD0)  BD 07 B8   |        JSR  CALCWR       FOR THIS SYMBOL
(0DD3)  DE 42      | RF10   LDX  LSTLAB
(0DC7)  0B         |
(0DD5)  E6 05      |        LDAB 5,X
(0DD7)  D7 2F      |        STAB BYTOUT
(0DD9)  BD 07 B8   |        JSR  CALCWR
(0DDC)  20 00      |        BRA  RF30       GO TO RESTORE
```

```
                        **SKIP
                        *
                        ***** R_FWD CONT
                        *
(0DDE)  DE  42      RF20    LDX  LSTLAB     HANDLE RELATIVE ITEMS
(0DC9)  14          
(0DE0)  A6  04      
(0DE2)  E6  05          LDAA 4,X       LOAD RELATIVE ADDR WITH
(0DE4)  CE  00  4E      LDAB 5,X         SYMBOL TABLE ADDR
(0DE7)  E0  01          LDX  #PGMCNT     MINUS
(0DE9)  A2  00          SUBB 1,X         PGM_CNT AT PREVIOUS REFERENCE
(0DEB)  DE  4E          SBCA X
(0DED)  09              LDX  PGMCNT    ADJUST PGM_CNT
(0DEE)  DF  4E          DEX              FOR RELATIVE
(0DF0)  D7  2F          STX  PGMCNT      ITEMS
(0DF2)  BD  0A  88      STAB BYTOUT
(0DF5)  BD  07  B8      JSR  BRTST     TEST RANGE OF
(0DF8)  DE  55          JSR  CALCWR      RELATIVE ADDR
(0DDD)  1A          RF30    LDX  SVPC      RESTORE
(0DFA)  DF  4E          
(0DFC)  DE  3C          STX  PGMCNT      PGM-CNT
(0DFE)  6F  00          LDX  IXS2      CLEAR
(0E00)  6F  01          CLR  X           ITEMS
(0E02)  6F  02          CLR  1,X         IN FWD  (POSITION IN SYMBOL TABLE)
(0E04)  6F  03          CLR  2,X         ADDR    (PGM
(0E06)  6F  04          CLR  3,X         TABL     CNT)
(0E08)  DE  3C          CLR  4,X                 (TYPE INDICATOR)
(0DB0)  57          RF80    LDX  IXS2      INCREMENT LOOP
(0DA6)  61          
(0DB4)  53          
(0E0A)  08          
(0E0B)  08              INX              INCREMENT
(0E0C)  08              INX              IX
(0E0D)  08              INX              BY
(0E0E)  08              INX              5
(0E0F)  DF  3C          INX              AND
(0E11)  7E  0D  8A      STX  IXS2        SAVE IT
                        JMP  RF01      GO TO MAIN LOOP
                        **SKIP
                        *
                        *      ************************************************************
                        *                              PRINT OUT UNRESOLVED ITEM
                        *
(0E14)  DE  3C      RF70    LDX  IXS2      IF ITEM POINTS TO A LOCATION
(0DA3)  0E  14          
(0E16)  E6  00          LDAB X           IN THE SYMBOL TABLE BEYOND THE
(0E18)  D1  5D          CMPB TMPBTM      TEMPORARY BOTTOM (TEMPBTM),
(0E1A)  2B  00          BMI  RF79        RETURN
(0E1C)  2E  00          BGT  RF72
(0E1E)  E6  01          LDAB 1,X
(0E20)  D1  5E          CMPB TMPBTM+1
(0E22)  2B  00          BMI  RF79
(0E24)  7D  00  34  RF72    TST  DELETE    ELSE, IF DELETE SWITCH IS ON,
(0E1D)  06          
(0E27)  23  00          BLS  RF75        CLEAR THE ENTRY
(0E29)  6F  00          CLR  X           AND RETURN
(0E28)  6F  01          CLR  1,X
(0E2D)  6F  02          CLR  2,X
(0E2F)  6F  03          CLR  3,X
(0E31)  6F  04          CLR  4,X
(0E33)  39              RTS
(0E34)  EE  00      RF75    LDX  X         ELSE, START DUMPING ITEM TO
(0E28)  0B          
(0E36)  86  04          LDAA #4          TERMINAL
(0E38)  97  5B          STAA TALY
(0E3A)  A6  00      RF76    LDAA X
(0E3C)  BD  04  94      JSR  CHROUT
(0E3F)  08              INX
(0E40)  7A  00  5B      DEC  TALY
(0E43)  26  F5          BNE  RF76
(0E45)  BD  04  A2      JSR  BLKOUT    WRITE OUT BLANK
(0E48)  DE  3C          LDX  IXS2        AND UNRESOLVED ADDRESS
(0E4A)  A6  02          LDAA 2,X
(0E4C)  BD  04  9A      JSR  HEXOUT
(0E4F)  A6  03          LDAA 3,X
(0E51)  BD  04  9A      JSR  HEXOUT
(0E54)  CE  06  70      LDX  #CRLF
(0E57)  BD  04  A5      JSR  STROUT    GO TO NEXT LINE
(0E5A)  39          RF79    RTS
(0E18)  3E          
(0E23)  36          
```

```
                        **SKIP
                        *
                        *      ************************************************************
                        *                        START PSEUDO OP PROCESSING
                        *
(0E5B)  CE  00  00  PSEUDO LDX  #PSTBL     LOCATE PSEUDO OP
(08D0)  0E  5B          
(0E5E)  5D          PSU1    TSTB               USING OP CODE (B) AND
(0E5F)  27  00          BEQ  PSU3          JUMP TABLE (PSTBL)
(0E61)  08              INX
(0E62)  08              INX
(0E63)  08              INX
(0E64)  5A              DECB
(0E65)  20  F7          BRA  PSU1
(0E67)  BD  07  59  PSU3    JSR  NX_ITM    LOAD START OF OPERAND
(0E60)  06          
(0E6A)  AD  00          JSR  X           CALL OPERAND PROCESSING
(0E6C)  39              RTS              RETURN
                        *
                        *   ASSEMBLER DIRECTIVES
                        *   EQU
                        *
(0E6D)  BD  08  2A  EQU     JSR  GENADR    GET OPERAND
(0E70)  9F  57          STS  SVST        SAVE STACK PTR
(0E72)  9E  29          LDS  AD01        MOVE OPERAND
(0E74)  DE  42          LDX  LSTLAB        INTO
(0E76)  AF  04          STS  4,X           SYMBOL TABLE
(0E78)  9F  62          STS  OUTADR        AND PRINTED ADDRESS
(0E7A)  9E  57          LDS  SVST        RESTORE STACK PTR
(0E7C)  39              RTS              RETURN
                        *
                        *   FCB - FDB
                        *
(0E7D)  4F          FCB     CLRA             FOR SINGLE BYTE
(0E7E)  20  00          BRA  FB00            CLEAR FLAGS
(0E80)  4F          FDB     CLRA             FOR DOUBLE BYTE
(0E81)  4C              INCA               SET FLAGS
(0E82)  97  37      FB00    STAA FLAGS
(0E7F)  02          
(0E84)  86  23          LDAA #'#         SET INPUT AS IMMEDIATE
(0E86)  97  21          STAA MOD
(0E88)  BD  08  2A  FB01    JSR  GENADR    OBTAIN OPERAND AND USE OUTPUT FOR
(0E8B)  BD  0A  C2      JSR  IMD2          IMEDIATE ADDRESSING TO PUT IT OUT
(0E8E)  96  30          LDAA CC          IF NEXT CHATR
(0E90)  81  2C          CMPA #$2C          IS A COMMA
(0E92)  27  01          BEQ  *+3           CONTINUE
(0E94)  39              RTS              ELSE RETURN
(0E95)  BD  07  30      JSR  NX_CHR      GET NEXT CHAR
(0E98)  20  EE          BRA  FB01        PUT OUT NEXT CHAR
                        **SKIP
                        *
                        *   FCC
                        *
(0E9A)  96  30      FCC     LDAA CC          IF FIRST CHARACTER IS
(0E9C)  81  30          CMPA #'0           NOT NUMERIC,
(0E9E)  2D  00          BLT  FCC5          GO TO DELIMITER PROCESSING
(0EA0)  81  39          CMPA #'9
(0EA2)  2E  00          BGT  FCC5
(0EA4)  BD  08  2A      JSR  GENADR      ELSE GET CHAR
(0EA7)  96  2A          LDAA AD01+1        COUNT
(0EA9)  97  5B          STAA TALY
(0EAB)  26  01      FCC1    BNE  *+3           IF COUNT IS NOT EXHAUSTED, CONTINUE
(0EAD)  39              RTS              ELSE RETURN
(0EAE)  BD  07  30      JSR  NX_CHR      PUT OUT
(0EB1)  97  2F          STAA BYTOUT        NEXT CHAR
(0EB3)  BD  07  B8      JSR  CALCWR
(0EB6)  7A  00  5B      DEC  TALY        DECREMENT COUNT
(0EB9)  20  F0          BRA  FCC1        GO TO NEXT CHAR
(0EBB)  97  5B      FCC5    STAA TALY        DELIMITER PROCESSING
(0E9F)  18          
(0EA3)  17          
(0EBD)  BD  07  30  FCC6    JSR  NX_CHR      IF DELIMITERS MATCH,
(0EC0)  91  5B          CMPA TALY            END
(0EC2)  27  00          BEQ  FCC9        ELSE PUT OUT NEXT CHARACTER
(0EC4)  97  2F          STAA BYTOUT
(0EC6)  BD  07  B8      JSR  CALCWR
(0EC9)  20  F2          BRA  FCC6          AND LOOP TO NEXT ITEM
(0EC8)  39          FCC9    RTS
(0EC3)  07          
                        *
                        *   ORG
                        *
(0ECC)  BD  08  2A  ORG     JSR  GENADR      GET OPERAND
(0ECF)  9F  57          STS  SVST        SAVE THE STACK
(0ED1)  9E  29          LDS  AD01        IF A LABEL WAS PRESENT (VALUE OF LAST
```

```
(0ED3) DE 42     |        LDX  LSTLAB      LABEL IN SYMBOL TABLE = CURRENT PGMCNT)
(0ED5) EE 04     |        LDX  4,X         STORE VALUE OF OPERAND IN SYMBOL TABLE
(0ED7) 9C 4E     |        CPX  PGMCNT
(0ED9) 26 00     |        BNE  ORG1
(0EDB) DE 42     |        LDX  LSTLAB
(0EDD) AF 04     |        STS  4,X
(0EDF) 9F 4E     | ORG1   STS  PGMCNT      LOAD PROGRAM COUNTER
(0EDA) 04        |
(0EE1) 9F 62     |        STS  OUTADR      AND PRINT ADDRESS
(0EE3) 9E 57     |        LDS  SVST
(0EE5) 39        |        RTS
                 | **SKIP
                 | *   RMB
                 | *
(0EE6) DE 4E     | RMB    LDX  PGMCNT      LOAD PRINT ADDRESS
(0EE8) DF 62     |        STX  OUTADR
(0EEA) BD 08 2A  |        JSR  GENADR      ADD OPRAND
(0EED) 96 4F     |        LDAA PGMCNT+1      TO
(0EEF) D6 4E     |        LDAB PGMCNT        PGMCNT
(0EF1) 9B 2A     |        ADDA AD01+1
(0EF3) D9 29     |        ADCB AD01
(0EF5) 97 4F     |        STAA PGMCNT+1      AND STORE IT
(0EF7) D7 4E     |        STAB PGMCNT
(0EF9) 39        |        RTS
                 | *
                 | *
                 | *   BGN
                 | *
(0EFA) DE 47     | BGN    LDX  NOWSTK      COMPARE CURRENT STACK LOCATION (NOWSTK) - 2
(0EFC) 01        |        NOP                TO CURRENT TOP OF SYMBOL TABLE (NOWPOS)
(0EFD) 01        |        NOP
(0EFE) 9C 45     |        CPX  NOWPOS
(0F00) 26 06     |        BNE  *+8         IF NOT EQUAL, CONTINUE
(0F02) 86 45     |        LDAA #'E         ELSE, SYMBOL TABLE FULL ERROR
(0F04) BD 07 A7  |        JSR  ERRS
(0F07) 39        |        RTS
(0F08) 9F 57     |        STS  SVST        SAVE THE STACK
(0F0A) 9E 47     |        LDS  NOWSTK      LOAD THE TABLE STACK
(0F0C) 96 46     |        LDAA NOWPOS+1    PUSH CURRENT LOCATION OF SYMBOL TABLE
(0F0E) 36        |        PSHA               ONTO THE TABLE STACK
(0F0F) 96 45     |        LDAA NOWPOS
(0F11) 36        |        PSHA
(0F12) 9F 47     |        STS  NOWSTK      RESTORE TABLE STACK
(0F14) 9E 57     |        LDS  SVST        RESTORE PROGRAM STACK
(0F16) 7C 00 40  |        INC  LEVEL       INCREMENT THE STRUCTURAL LEVEL INDICATOR
(0F19) 39        |        RTS                AND RETURN
                 | *
                 | *   END OF ASSEMBLER DIRECTIVES
                 | *
(0F1A) 7E 0E 6D  | PSTBL  JMP  EQU          TABLE OF STARTING ADDRESSES FOR PSEUDO ROUTINES
(0E5C) 0F 1A     |
(0F1D) 7E 0E 7D  |        JMP  FCB
(0F20) 7E 0E 9A  |        JMP  FCC
(0F23) 7E 0E 80  |        JMP  FDB
(0F26) 7E 0E CC  |        JMP  ORG
(0F29) 7E 0E E6  |        JMP  RMB
(0F2C) 7E 0E FA  |        JMP  BGN
                 | **SKIP
                 | *
                 | *
                 | *  ***************************************************************
                 | *                          END OF BLOCK
                 | *
(0F2F) BD 07 F1  | END    JSR  WRITE       WRITE FIRST HALF OF 'END' LINE
(08C1) 0F 2F     |
(0F32) 9F 57     |        STS  SVST
(0F34) 9E 47     |        LDS  NOWSTK      PULL TEMPORARY BOTTOM OF THE SYMBOL TABLE
(0F36) 32        |        PULA               (TEMPBTM) FROM THE TABLE STACK)
(0F37) 97 5D     |        STAA TMPBTM
(0F39) 32        |        PULA
(0F3A) 97 5E     |        STAA TMPBTM+1
(0F3C) 9F 47     |        STS  NOWSTK
(0F3E) 9E 57     |        LDS  SVST
(0F40) CE 06 4D  |        LDX  #MSG1       WRITE OUT
(0F43) BD 04 A5  |        JSR  STROUT        UNRESOLVED TITLE
(0F46) 7C 00 36  |        INC  ENDFL       SET END OF BLOCK FLAG
(0F49) 7C 00 38  |        INC  FWD           AND FWD FLAG
(0F4C) BD 0D 80  |        JSR  R_FWD       SEARCH FOR UNRESOLVED ITEMS
(0F4F) CE 06 62  |        LDX  #MSG2       ASK FOR SYMBOL DUMP
(0F52) BD 04 A5  |        JSR  STROUT
(0F55) BD 04 97  |        JSR  CHRIN
(0F58) 81 59     |        CMPA #'Y         IF ANSWER IS 'Y',
(0F5A) 27 00     |        BEQ  END5          GO TO DUMP TABLE
(0F5C) DE 47     |        LDX  NOWSTK      ELSE
(0F5E) 09        |        DEX                RESTORE TABLE STACK
(0F5F) 09        |        DEX
(0F60) DF 47     |        STX  NOWSTK
(0F62) 20 00     |        BRA  END9          AND RETURN
(0F64) BD 00 00  | END5   JSR  SMDP        DUMP SYMBOLS
(0F5B) 08        |
(0F67) CE 06 44  | END9   LDX  #LINFED
(0F63) 03        |
(0F6A) BD 04 A5  |        JSR  STROUT
(0F6D) 39        |        RTS              RETURN
                 | **SKIP
                 | *
                 | *  ***************************************************************
                 | *                          DUMP SYMBOL TABLE
                 | *
(0F6E) BD 00 00  | SMDP   JSR  SORT        SORT SYMBOL TABLE
(0F65) 0F 6E     |
(0F71) DE 5D     |        LDX  TMPBTM      POSITION IX TO START OF TABLE
(0F73) 86 05     | SMD2   LDAA #5          LOAD TALY WITH 5
(0F75) 97 5B     |        STAA TALY
(0F77) DF 3A     |        STX  IXS1
(0F79) CE 06 70  |        LDX  #CRLF       PUT OUT CARRIAGE RETURN/LINE FFED
(0F7C) BD 04 A5  |        JSR  STROUT
(0F7F) DE 3A     |        LDX  IXS1
(0F81) 86 04     | SMD4   LDAA #4          LOAD COUNT WITH 4
(0F83) 97 33     |        STAA COUNT
(0F85) A6 00     | SMD6   LDAA X           WRITE OUT FIRST 4 CHARACTERS
(0F87) 27 00     |        BEQ  SMD9          IF NULL CHARACTER, GO TO END
(0F89) BD 04 94  |        JSR  CHROUT
(0F8C) 08        |        INX
(0F8D) 7A 00 33  |        DEC  COUNT
(0F90) 26 F3     |        BNE  SMD6
(0F92) BD 04 A2  |        JSR  BLKOUT      WRITE OUT A BLANK
(0F95) A6 00     |        LDAA X
(0F97) BD 04 9A  |        JSR  HEXOUT      WRITE OUT FIRST HALF OF ADDRESS
(0F9A) 08        |        INX
(0F9B) A6 00     |        LDAA X
(0F9D) BD 04 9A  |        JSR  HEXOUT      WRITE OUT SECOND HALF OF ADDRESS
(0FA0) 08        |        INX
(0FA1) BD 04 A2  |        JSR  BLKOUT      WRITE OUT TWO BLANKS
(0FA4) BD 04 A2  |        JSR  BLKOUT
(0FA7) 7A 00 5B  |        DEC  TALY        IF 5 SYMBOLS HAVE NOT BEEN PRINTED,
(0FAA) 26 D5     |        BNE  SMD4          CONTINUE
(0FAC) 20 C5     |        BRA  SMD2          GO TO NEXT GROUP OF 5 SYMBOLS
(0FAE) 7C 00 34  | SMD9   INC  DELETE      SET THE DELETE FWD FLAG
(0F88) 25        |
(0FB1) 7C 00 36  |        INC  ENDFL         AND PROCESS FWD
(0FB4) 7C 00 38  |        INC  FWD
(0FB7) BD 0D 80  |        JSR  R_FWD
(0FBA) DE 5D     |        LDX  TMPBTM      STORE TMP BOTTOM INTO CURRENT TOP
(0FBC) DF 45     |        STX  NOWPOS        OF SYMBOL TABLE
(0FBE) 6F 00     |        CLR  X           CLEAR OLD SYMBOLS
(0FC0) 7A 00 40  |        DEC  LEVEL       DECREMENT STRUCTURAL INDICATOR
(0FC3) 26 03     |        BNE  *+5           IF ZERO, END OF JOB
(0FC5) 7C 00 5C  |        INC  TERM
(0FC8) 39        |        RTS              AT END, RETURN
                 | **SKIP
                 | *
                 | *  ***************************************************************
                 | *                          SORT SYMBOL TABLE
                 | *
(0FC9) DE 5D     | SORT   LDX  TMPBTM      GET START OF TABLE
(0F6F) 0F C9     |
(0FCB) 7F 00 2E  |        CLR  ANSW        CLEAR SWAP SWITCH
(0FCE) 20 00     |        BRA  SORT4       GO TO CHECK FOR END OF TABLE
(0FD0) C6 06     | SORT1  LDAB #6          LOAD LENGTH OF ENTRY
(0FD2) A6 00     | SORT2  LDAA X           COMPARE CHAR OF ENTRY WITH NEXT ENTRY
(0FD4) A1 06     |        CMPA 6,X
(0FD6) 2D 00     |        BLT  SORT3         IF < GO TO CHECK NEXT SYM
(0FD8) 26 00     |        BNE  SWAP          IF > GO TO SWAP SYMBOLS
(0FDA) 08        |        INX              ELSE GO TO NEXT CHAR (LAST CANT BE EQUAL)
(0FDB) 5A        |        DECB             B = PORTION OF SYMBOLS NOT EQUAL
(0FDC) 20 F4     |        BRA  SORT2       GO TO NEXT CHARACTER
(0FDE) 08        | SORT3  INX              SKIP TO
(0FD7) 06        |
(0FDF) 5A        |        DECB               NEXT SYMBOL
(0FE0) 26 FC     |        BNE  SORT3
(0FE2) 6D 06     | SORT4  TST  6,X         IF NOT AT END OF TABLE
(0FCF) 12        |
(0FE4) 26 EA     |        BNE  SORT1         KEEP GOING
(0FE6) 7D 00 2E  |        TST  ANSW        IF SWAP THEN
(0FE9) 26 DE     |        BNE  SORT          KEEP GOING
(0FEB) 39        |        RTS              ELSE RETURN
(0FEC) D7 33     | SWAP   STAB COUNT       STORE NUMBER OF CHAR'S TO SWAP
(0FD9) 12        |
```

```
(0FEE)   D7  2E      |         STAB ANSW        SET SWAP INDICATOR
(0FF0)   A6  00      | SWAP1   LDAA X           SWAP LOOP
(0FF2)   E6  06      |         LDAB 6,X
(0FF4)   A7  06      |         STAA 6,X
(0FF6)   E7  00      |         STAB X
(0FF8)   08          |         INX
(0FF9)   7A  00  33  |         DEC  COUNT
(0FFC)   26  F2      |         BNE  SWAP1
(0FFE)   20  E2      |         BRA  SORT4       CONTINUE
                     | *
                     | *
                     | *       ****************************************************************
                     | *       ****************************************************************
                     | *                              END OF PROGRAM
```

END OF ASSEMBLY, UNRESOLVED ITEMS ARE:

SYMBOL TABLE FOLLOWS:

MAIN	1661	CR	13	LF	10	EOT	4	SPAE	32
EMOW	32	MOD	33	NAME	34	SIGN	38	BLNK	39
AD01	41	SAVR	43	ALPA	45	ANSW	46	BYTT	47
CC	48	CLAS	49	CMPN	50	COUT	51	DELE	52
EJET	53	ENDL	54	FLAS	55	FWD	56	HEX	57
IXS1	58	IXS2	60	IXS3	62	LEVL	64	LABW	65
LSTB	66	LTRL	68	NOWS	69	NOWK	71	NXCR	73
NULL	75	OPCE	77	PGMT	78	P_PS	80	STR1	81
SIR2	83	SVPC	85	SVST	87	TBLS	89	TALY	91
TERM	92	TMPM	93	TRNF	95	UNRL	97	OUTR	98
BYT1	100	BYT2	101	BYT3	102	FTBL	103	TB2G	105
TB2P	107	STBL	109	BUF1	111	BUFD	167	TB01	168
TB02	268	TB03	269	TB04	1169	CHRT	1172	CHRN	1175
NEXT	1178	BLK1	1186	STRT	1189	MIKG	1192	TBL1	1195
TTBL	1247	ERRA	1567	ERRB	1582	HEAR	1586	LIND	1604
NALE	1608	MSG1	1613	MSG2	1634	CRLF	1648	ASTR	1652
XUPD	1656	EOL	1660	CLR1	1705	ENTY	1730	CMD	2153
LODF	1779	LOB1	1793	LDB5	1801	BUFO	1816	NX_R	1840
NXC9	1871	WRIE	2033	NX_M	1881	NXI1	1883	LD_M	1893
LDI1	1902	LDI9	1938	LDI5	1925	LD99	1957	LD98	1956
ERRS	1959	CALM	1976	CLC4	2015	CLC2	2001	CLC9	2027
CLC3	2012	WRT0	2044	WRT1	2069	WRT2	2096	WRT4	2085
WRT3	2104	WRT5	2126	MN01	2164	MN02	2194	CMNT	2287
MN04	2207	SYMS	3184	MN08	2281	TRNS	2358	MN05	2249
END	3887	MN06	2260	PSEO	3675	MN07	2272	M_SH	2298
VERB	2595	R_FD	3456	CMN9	2297	SCM5	2337	SCH3	2327
SCH9	2357	TR01	2387	TR02	2394	TR03	2423	STRM	2558
TR08	2455	OP_N	2471	OP03	2505	OP09	2557	OP08	2554
OP06	2543	OP04	2533	ST01	2575	ST99	2589	VB02	2618
RELV	2654	VB03	2628	IMEM	2728	VB20	2638	INDD	2784
VB99	2648	DFLI	2802	GENR	2858	RL10	2670	BRTT	2696
RLER	2710	BRT9	2727	IMD2	2754	IMD3	2764	IMD5	2778
IMD4	2776	DFL0	2828	DFL1	2833	DFL4	2850	GETR	2946
GEN8	2927	GEN1	2867	GEN2	2878	GEN3	2913	GEN9	2945
GAD1	2974	CVTX	3122	GAD2	3004	GAD5	3061	GAD4	3057
GAD3	3035	GAD6	3089	GAD9	3121	CVT1	3125	CVT9	3183
CVT3	3146	SM01	3203	SM02	3221	SM03	3227	NOHT	3293
SM10	3261	S_FD	3367	SM90	3311	SMBD	3321	SMB5	3358
SFW1	3369	SFW3	3391	SFW5	3401	SFW7	3408	SFW8	3427
SFW9	3448	RF01	3466	RF02	3484	RF03	3485	RF05	3495
RF70	3604	RF80	3592	RF10	3539	RF20	3550	RF30	3576
RF79	3674	RF72	3620	RF75	363A	RF76	3642	PSTL	3866
PSU1	3678	PSU3	3687	EQU	3693	FCB	3709	FB00	3714
FUB	3712	FB01	3720	FCC	373A	FCC5	3771	FCC1	3755
FCC6	3773	FCC9	3787	ORG	3788	ORG1	3807	RMB	3814
BGN	3834	END5	3940	END9	3943	SMDP	3950	SORT	4041
SMD2	3955	SMD4	3969	SMD6	3973	SMD9	4014	SOR4	4066
SOR1	4048	SOR2	4050	SOR3	4062	SWAP	4076	SWA1	4080

END OF NORMAL EXECUTION

Appendix C: Bar Code representation of Tiny Assembler version 3.1 object code

Beginning on page 71 is a complete machine readable representation of the object code for *Tiny Assembler 6800 Version 3.1,* as assembled in the listing found on pages 54 to 66 of this book.

This representation uses the absolute loader format, in which each bar code frame (one line of bars running from top to bottom of the page) contains a two byte address followed by data which is loaded in ascending order starting at that address.

The object code listing shown below gives the information in hexadecimal form, for use as a confirmation copy or for manual entry of this program.

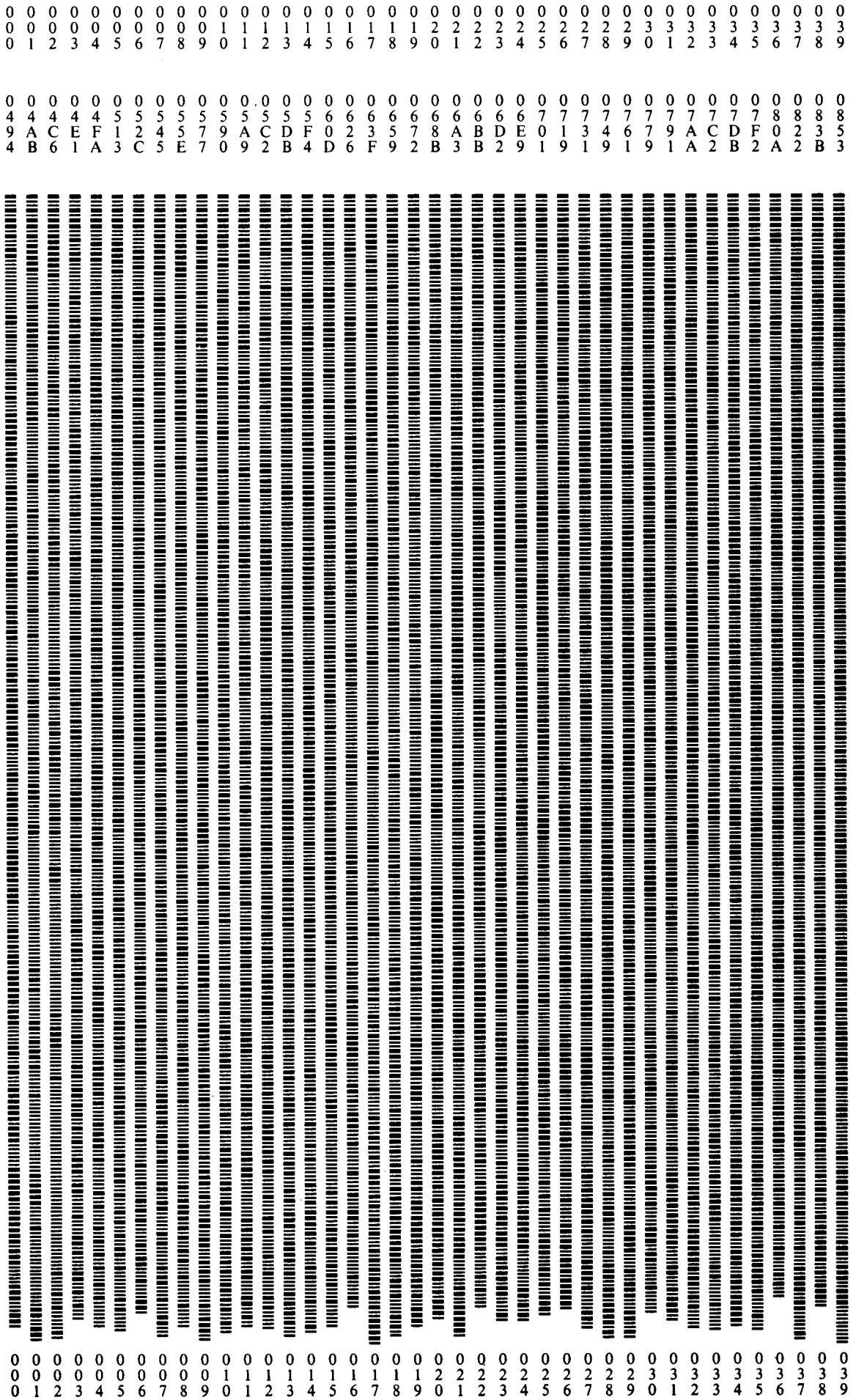
For details on the frame format and absolute loader format used in this and all PAPERBYTE™ books, see the PAPERBYTE publication *Bar Code Loader* by Ken Budnick. This book contains a brief history on bar codes, a general bar code loader algorithm with flow charts and complete program listings for 6800, 6502 and 8080 or Z-80 based systems.

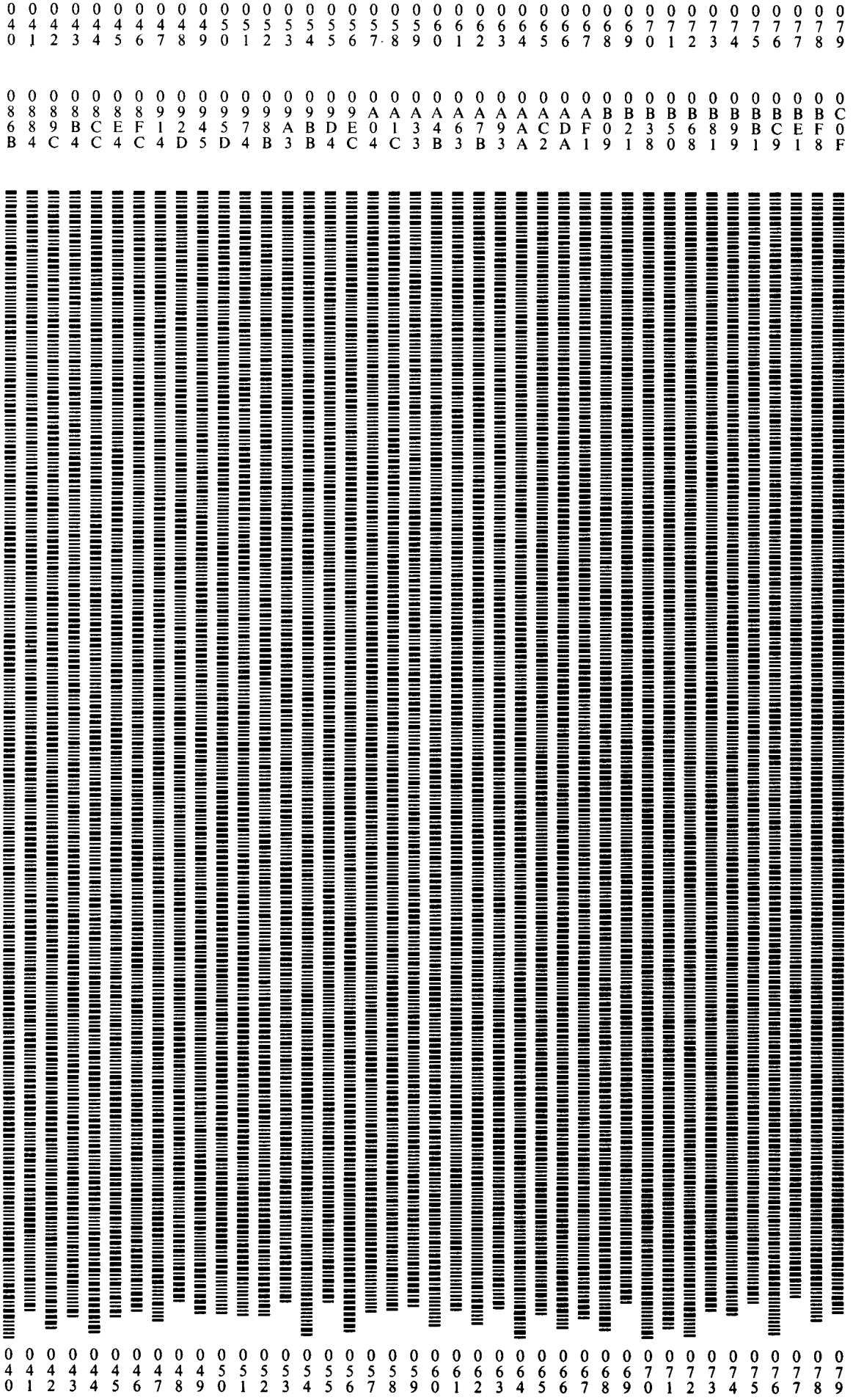
```
0494 7E E1 D1 7E E1 AC 36 BD E0 67 32 7E E0 6B 7E E0
04A4 CC 7E E0 7E 7E E0 E3 00 06 06 12 18 08 20 04 24
04B4 03 27 03 00 00 00 00 2A 03 2D 02 00 00 2F 04 00
04C4 00 33 02 35 02 37 02 00 00 39 05 3E 0A 48 07 00
04D4 00 00 00 4F 01 00 00 00 00 00 00 42 41 1B 50 44
04E4 43 89 10 44 44 8B 10 4E 44 84 10 53 4C 48 30 53
04F4 52 47 30 43 43 24 40 43 53 25 40 45 51 27 40 47
0504 45 2C 40 47 4E 06 F0 47 54 2E 40 48 49 22 40 49
0514 54 85 10 4C 45 2F 40 4C 53 23 40 4C 54 2D 40 4D
0524 49 2B 40 4E 45 26 40 50 4C 2A 40 52 41 20 40 53
0534 52 8D 40 56 43 28 40 56 53 29 40 42 41 11 50 4C
0544 43 0C 50 4C 49 0E 50 4C 52 4F 30 4C 56 0A 50 4D
0554 50 81 10 4F 4D 43 30 50 58 8C 11 41 41 19 50 45
0564 43 4A 30 45 53 34 50 45 58 09 50 4E 44 00 E0 4F
0574 52 88 10 51 55 00 F0 43 42 01 F0 43 43 02 F0 44
0584 42 03 F0 4E 43 4C 30 4E 53 31 50 4E 58 08 50 4D
0594 50 4E 31 53 52 8D 31 44 41 86 10 44 53 8E 11 44
05A4 58 CE 11 53 52 44 30 45 47 40 30 4F 50 01 50 52
05B4 41 8A 10 52 47 04 F0 53 48 36 00 55 4C 32 00 4D
05C4 42 05 F0 4F 4C 49 30 4F 52 46 30 54 49 3B 50 54
05D4 53 39 50 42 41 10 50 42 43 82 10 45 43 0D 50 45
05E4 49 0F 50 45 56 0B 50 54 41 87 20 54 53 8F 21 54
05F4 58 CF 21 55 42 80 10 57 49 3F 50 41 42 16 50 41
0604 50 06 50 42 41 17 50 50 41 07 50 53 54 4D 30 53
0614 58 30 50 58 53 35 50 41 49 3E 50 0D 00 00 2A 2A
0624 2A 20 45 52 52 4F 52 20 2D 20 00 0A 20 04 0D 0A
0634 0A 0A 4C 4F 43 4E 20 42 31 20 42 32 20 42 33 20
0644 0A 0A 00 04 0D 00 00 12 04 0D 00 2A 2A 2A 20 55
0654 4E 52 45 53 4F 4C 56 45 44 3A 0D 0A 00 04 0D 00
0664 2A 2A 2A 20 53 59 4D 42 4F 4C 53 3F 0D 0A 00 04
0674 2A 20 20 20 58 20 20 20 0D 86 01 97 35 97 40 86
0684 39 CE 01 0C DF 6B A7 00 AD 00 09 09 09 09 09 DF
0694 69 CE 00 00 DF 4E DF 62 86 20 97 64 97 65 97 66
06A4 CE 00 A8 DF 67 6F 00 08 8C 04 91 26 F8 DF 47 8E
06B4 01 0D 9F 45 AF 01 9F 6D 86 39 DE 6B A7 00 7D 00
06C4 40 27 B6 CE 00 6F DF 49 CE 06 C2 FF A0 48 8E A0
06D4 7F CE 00 00 DF 29 DF 4B 4F 97 36 97 5C 97 38 97
06E4 50 97 5B 97 61 86 04 97 32 BD 08 69 7E 04 A8 7D
06F4 00 35 23 09 CE 06 32 BD 04 A5 7F 00 35 CE 00 6F
0704 86 3E BD 04 94 BD 04 97 81 21 26 08 8C 00 6F 27
```

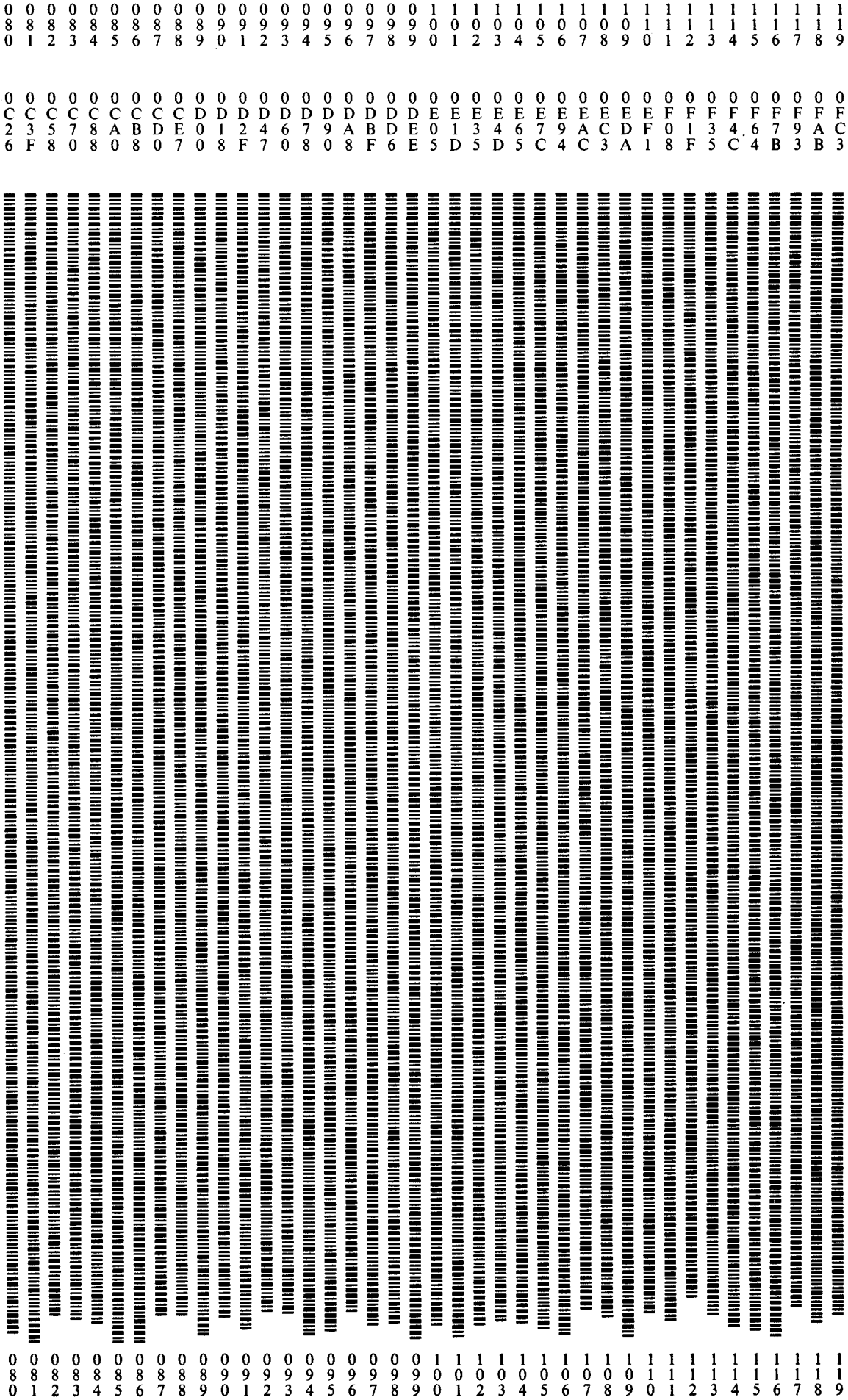
```
0714 F4 09 20 F1 A7 00 B1 06 7C 26 01 39 8C 00 A7 27
0724 E4 81 20 2B E0 81 5F 22 DC 08 20 D9 DF 3E DE 49
0734 96 30 B1 06 7C 26 14 7D 00 50 27 03 BD 07 F1 86
0744 01 97 50 BD 06 F3 CE 00 6F DF 49 A6 00 97 30 08
0754 DF 49 DE 3E 39 96 30 81 20 27 01 39 BD 07 30 20
0764 F6 DE 27 DF 22 DF 24 CE 00 22 D6 30 D1 27 27 1E
0774 F1 06 7C 27 19 C1 2B 27 15 C1 2D 27 11 C1 2C 27
0784 0D E7 00 8C 00 25 27 01 08 BD 07 30 20 DC 09 A6
0794 00 81 27 26 0C 09 A6 00 81 27 27 04 08 08 20 E1
07A4 08 08 39 B7 06 2E DF 3A CE 06 1F BD 04 A5 DE 3A
07B4 7C 00 20 39 7D 00 20 23 01 39 96 2F D6 50 C1 01
07C4 27 19 C1 02 26 07 97 65 7C 00 50 20 1A C1 03 26
07D4 07 97 66 7C 00 50 20 0F BD 07 F1 DE 4E DF 62 96
07E4 2F 97 64 86 02 97 50 DE 4E 08 DF 4E 39 DF 3E CE
07F4 06 48 BD 04 A5 CE 75 30 6D 00 09 26 FB 86 01 BD
0804 04 94 96 62 BD 04 9A 96 63 BD 04 9A C6 01 CE 00
0814 64 D1 50 2B 17 26 0A 86 01 BD 04 94 86 14 BD 04
0824 94 BD 04 A2 BD 04 A2 BD 04 A2 20 08 BD 04 A2 A6
0834 00 BD 04 9A 08 5C C1 04 26 D7 96 50 81 04 26 0A
0844 86 01 BD 04 94 86 14 BD 04 94 BD 04 A2 DE 4E DF
0854 62 DE 3E 86 0A BD 04 94 96 27 97 64 97 65 97 66
0864 86 01 97 50 39 CE 00 21 C6 20 7F 00 20 7F 00 61
0874 E7 00 08 8C 00 29 26 F8 B6 06 7C 97 30 BD 07 30
0884 B1 06 7C 27 E0 81 2A 26 05 BD 08 EF 20 D7 91 27
0894 27 09 BD 07 65 7C 00 41 BD 0C 70 BD 07 59 B1 06
08A4 7C 27 42 BD 07 65 86 02 97 32 BD 09 36 86 04 97
08B4 32 7D 00 20 22 AF 96 31 81 0E 26 09 BD 0F 2F 7D
08C4 00 5C 23 A1 39 D6 4D 81 0F 26 05 BD 0E 5B 20 15
08D4 BD 07 59 81 23 26 05 97 21 BD 07 30 BD 07 59 BD
08E4 08 FA BD 0A 23 BD 0D 80 7E 08 69 BD 07 30 81 50
08F4 26 03 7C 00 35 39 DE 49 96 30 81 58 26 1F A6 00
0904 91 27 27 0F B1 06 7C 27 0A 81 2B 27 06 81 2D 27
0914 02 20 0A 86 58 97 21 86 30 97 30 20 14 A6 00 08
0924 91 27 27 0D B1 06 7C 27 08 81 2C 26 F0 A6 00 97
0934 21 39 96 22 81 41 2C 06 86 41 BD 07 A7 39 81 5A
0944 2F 06 86 41 BD 07 A7 39 84 1F 4A 48 CE 04 AB 4D
0954 27 04 08 4A 20 F9 E6 01 D7 44 E6 00 4F 58 49 58
0964 49 CE 04 DF DF 5F DB 60 99 5F D7 60 97 5F CE 00
0974 23 DF 51 7D 00 44 26 06 86 42 BD 07 A7 39 DE 5F
0984 DF 53 BD 09 FE 7D 00 2E 23 09 BD 09 A7 7D 00 2E
0994 23 01 39 08 08 08 08 DF 5F CE 00 23 DF 51 7A 00
09A4 44 20 D0 A6 02 97 4D A6 03 16 C4 0F D7 37 44 44
09B4 44 44 97 31 D6 25 D1 27 26 0B 7D 00 37 22 3A 81
09C4 02 23 33 20 34 81 03 22 2D 7D 00 37 22 28 C1 41
09D4 27 19 C1 42 26 20 D6 4D 4D 26 06 CB 01 D7 4D 20
09E4 0A 81 03 27 02 CB 30 CB 10 D7 4D 4C 85 02 26 09
09F4 86 05 97 31 20 03 7F 00 2E 39 36 DF 3A 9F 57 7F
0A04 00 2E 96 32 97 5B DE 51 9E 53 34 32 A1 00 26 09
0A14 08 7A 00 5B 26 F5 7C 00 2E DE 3A 9E 57 32 39 96
0A24 31 D6 4D 81 03 23 0F D7 2F BD 07 B8 96 31 81 04
0A34 26 03 BD 0A 5E 39 96 21 81 23 26 04 BD 0A A8 39
0A44 81 58 26 06 CB 20 BD 0A E0 39 91 27 26 06 CB 10
0A54 BD 0A F2 39 86 44 BD 07 A7 39 BD 0B 2A 7D 00 61
0A64 23 08 96 2A 97 2F BD 07 B8 39 CE 00 4E 96 29 D6
0A74 2A E0 01 A2 00 C0 01 92 4B D7 2F BD 0A 88 01 01
0A84 BD 07 B8 39 91 4B 27 06 81 FF 27 02 20 04 98 2F
0A94 2A 11 86 47 BD 07 A7 DE 4E 7D 00 38 22 01 09 DF
0AA4 4E DF 62 39 96 31 81 01 27 06 86 43 BD 07 A7 39
0AB4 37 BD 0B 2A 33 D7 2F BD 07 B8 BD 0A C2 39 96 22
```

```
0AC4 81 27 26 04 96 23 20 0E 7D 00 37 23 07 96 29 97
0AD4 2F BD 07 B8 96 2A 97 2F BD 07 B8 39 37 BD 0B 2A
0AE4 33 D7 2F BD 07 B8 96 2A 97 2F BD 07 B8 39 37 BD
0AF4 0B 2A 33 7F 00 5B 7D 00 29 26 0D 7D 00 61 22 08
0B04 96 31 81 03 27 02 20 05 CB 20 7C 00 5B D7 2F BD
0B14 07 B8 7D 00 5B 27 07 D6 29 D7 2F BD 07 B8 D6 2A
0B24 D7 2F BD 07 B8 39 7F 00 41 BD 0B 82 BD 0B 6F 96
0B34 30 81 2B 27 05 81 2D 27 01 39 97 26 DE 29 DF 2B
0B44 BD 07 30 BD 0B 82 BD 0B 6F 96 26 81 2B 27 0E 96
0B54 2C 90 2A 97 2A 96 2B 92 29 97 29 20 D2 96 2C 9B
0B64 2A 97 2A 96 2B 99 29 97 29 20 C4 7D 00 20 22 0D
0B74 7D 00 39 22 08 7D 00 2D 23 03 BD 0C 70 39 7C 00
0B84 2D 7C 00 39 7F 00 29 7F 00 2A 96 30 81 24 26 0A
0B94 BD 07 30 BD 07 65 BD 0C 32 39 7F 00 39 81 26 26
0BA4 03 BD 07 30 BD 07 65 CE 00 22 A6 00 91 27 27 08
0BB4 81 30 2B 3D 81 39 22 39 91 27 27 31 8C 00 26 27
0BC4 2C 85 30 26 06 86 4A BD 07 A7 39 84 0F A7 00 C6
0BD4 09 D7 5B D6 2A 96 29 DB 2A 99 29 7A 00 5B 26 F7
0BE4 EB 00 99 4B D7 2A 97 29 08 A6 00 20 CB 7F 00 2D
0BF4 39 CE 06 78 DF 51 CE 00 22 DF 53 BD 09 FE 7D 00
0C04 2E 23 0A DE 4B DF 22 DF 24 7F 00 2D 39 CE 06 74
0C14 DF 51 CE 00 22 DF 53 BD 09 FE 7D 00 2E 23 0E DE
0C24 4E 96 31 81 04 26 01 09 DF 29 7F 00 2D 39 CE 00
0C34 22 A6 00 91 27 27 34 8C 00 26 27 2F 81 30 2B 06
0C44 81 46 22 02 20 06 86 4A BD 07 A7 39 81 41 2B 02
0C54 8B 09 84 0F A7 00 96 29 D6 2A 58 49 58 49 58 49
0C64 58 49 EB 00 D7 2A 97 29 08 20 C6 39 96 22 81 27
0C74 26 01 39 81 40 22 06 86 4B BD 07 A7 39 DE 6D DF
0C84 59 96 48 90 5A D6 47 D0 59 2B 06 2E 0A 81 06 22
0C94 06 86 45 BD 07 A7 39 DF 53 CE 00 22 DF 51 BD 09
0CA4 FE 7D 00 2E 23 33 DE 59 7D 00 41 22 0C EE 04 DF
0CB4 29 9C 4B 26 03 BD 0D 27 39 EE 04 9C 4B 27 06 86
0CC4 46 BD 07 A7 39 7C 00 38 DE 59 96 4E A7 04 96 4F
0CD4 A7 05 DF 42 EE 04 DF 29 39 DE 59 6D 00 26 0C BD
0CE4 0C F9 7D 00 41 22 03 BD 0D 27 39 DE 59 08 08 08
0CF4 08 08 08 20 8A DE 59 08 08 08 08 08 08 DF 45 DE
0D04 59 9F 57 9E 22 AF 00 9E 24 AF 02 6F 04 6F 05 7D
0D14 00 41 23 06 9E 4E AF 04 DF 42 6F 06 EE 04 DF 29
0D24 9E 57 39 DE 67 9C 6B 26 06 86 49 BD 07 A7 39 9C
0D34 69 26 08 86 48 BD 07 A7 7F 00 20 E6 00 26 06 E6
0D44 01 26 02 20 07 08 08 08 08 08 20 D9 96 21 81 58
0D54 26 02 6A 04 81 23 26 07 7D 00 37 22 02 6A 04 96
0D64 31 81 04 26 02 6C 04 9F 57 9E 59 AF 00 9E 4E 81
0D74 0F 27 01 31 AF 02 9E 57 7C 00 61 39 7D 00 38 22
0D84 01 39 DE 67 DF 3C 7D 00 20 22 0D 9C 6B 26 0A 7F
0D94 00 38 7F 00 34 7F 00 36 39 7D 00 36 23 05 BD 0E
0DA4 14 20 61 DE 3C A6 00 E6 01 91 42 26 57 D1 43 26
0DB4 53 DE 4E DF 55 BD 07 F1 DE 3C EE 02 DF 4E DE 3C
0DC4 6D 04 2B 0B 26 14 DE 42 E6 04 D7 2F BD 07 B8 DE
0DD4 42 E6 05 D7 2F BD 07 B8 20 1A DE 42 A6 04 E6 05
0DE4 CE 00 4E E0 01 A2 00 DE 4E 09 DF 4E D7 2F BD 0A
0DF4 88 BD 07 B8 DE 55 DF 4E DE 3C 6F 00 6F 01 6F 02
0E04 6F 03 6F 04 DE 3C 08 08 08 08 08 DF 3C 7E 0D 8A
0E14 DE 3C E6 00 D1 5D 2B 3E 2E 06 E6 01 D1 5E 2B 36
0E24 7D 00 34 23 0B 6F 00 6F 01 6F 02 6F 03 6F 04 39
0E34 EE 00 86 04 97 5B A6 00 BD 04 94 08 7A 00 5B 26
0E44 F5 BD 04 A2 DE 3C A6 02 BD 04 9A A6 03 BD 04 9A
0E54 CE 06 70 BD 04 A5 39 CE 0F 1A 5D 27 06 08 08 08
0E64 5A 20 F7 BD 07 59 AD 00 39 BD 0B 2A 9F 57 9E 29
```

```
0E74 DE 42 AF 04 9F 62 9E 57 39 4F 20 02 4F 4C 97 37
0E84 86 23 97 21 BD 0B 2A BD 0A C2 96 30 81 2C 27 01
0E94 39 BD 07 30 20 EE 96 30 81 30 2D 1B 81 39 2E 17
0EA4 BD 0B 2A 96 2A 97 5B 26 01 39 BD 07 30 97 2F BD
0EB4 07 B8 7A 00 5B 20 F0 97 5B BD 07 30 91 5B 27 07
0EC4 97 2F BD 07 B8 20 F2 39 BD 0B 2A 9F 57 9E 29 DE
0ED4 42 EE 04 9C 4E 26 04 DE 42 AF 04 9F 4E 9F 62 9E
0EE4 57 39 DE 4E DF 62 BD 0B 2A 96 4F D6 4E 9B 2A D9
0EF4 29 97 4F D7 4E 39 DE 47 01 01 9C 45 26 06 86 45
0F04 BD 07 A7 39 9F 57 9E 47 96 46 36 96 45 36 9F 47
0F14 9E 57 7C 00 40 39 7E 0E 6D 7E 0E 7D 7E 0E 9A 7E
0F24 0E 80 7E 0E CC 7E 0E E6 7E 0E FA BD 07 F1 9F 57
0F34 9E 47 32 97 5D 32 97 5E 9F 47 9E 57 CE 06 4D BD
0F44 04 A5 7C 00 36 7C 00 38 BD 0D 80 CE 06 62 BD 04
0F54 A5 BD 04 97 81 59 27 08 DE 47 09 09 DF 47 20 03
0F64 BD 0F 6E CE 06 44 BD 04 A5 39 BD 0F C9 DE 5D 86
0F74 05 97 5B DF 3A CE 06 70 BD 04 A5 DE 3A 86 04 97
0F84 33 A6 00 27 25 BD 04 94 08 7A 00 33 26 F3 BD 04
0F94 A2 A6 00 BD 04 9A 08 A6 00 BD 04 9A 08 BD 04 A2
0FA4 BD 04 A2 7A 00 5B 26 D5 20 C5 7C 00 34 7C 00 36
0FB4 7C 00 38 BD 0D 80 DE 5D DF 45 6F 00 7A 00 40 26
0FC4 03 7C 00 5C 39 DE 5D 7F 00 2E 20 12 C6 06 A6 00
0FD4 A1 06 2D 06 26 12 08 5A 20 F4 08 5A 26 FC 6D 06
0FE4 26 EA 7D 00 2E 26 DE 39 D7 33 D7 2E A6 00 E6 06
0FF4 A7 06 E7 00 08 7A 00 33 26 F2 20 E2
```


000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015 016 017 018 019 020 021 022 023 024 025 026 027 028 029 030 031 032 033 034 035 036 037 038 039
044 04AB 04C6 04E1 04FA 513 52C 545 55E 577 590 5A9 5C2 5DB 5F4 60D 626 63F 659 672 68B 6A3 6BB 6D2 6E9 701 719 731 749 761 779 791 7AA 7C2 7DB 7F2 80A 822 83B 853


040 041 042 043 044 045 046 047 048 049 050 051 052 053 054 055 056 057 058 059 060 061 062 063 064 065 066 067 068 069 070 071 072 073 074 075 076 077 078 079
086B 0884 089C 08B4 08CC 08E4 08FC 0914 092D 0945 095D 0974 098B 09A3 09BB 09D4 09EC 0A04 0A1C 0A33 0A4B 0A63 0A7B 0A93 0AAA 0AC2 0ADA 0AF1 0B09 0B21 0B38 0B50 0B68 0B81 0B99 0BB1 0BC9 0BE1 0BF8 0C0F


0800 0801 0802 0803 0804 0805 0806 0807 0808 0809 0900 0901 0902 0903 0904 0905 0906 0907 0908 0909 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014 1015 1016 1017 1018 1019
0C26 0C3F 0C58 0C70 0C88 0CA0 0CB8 0CD0 0CE7 0D00 0D18 0D2F 0D47 0D60 0D78 0D90 0DA8 0DBF 0DD6 0DEE 0E05 0E1D 0E35 0E4D 0E65 0E7C 0E94 0EAC 0EC3 0EDA 0EF1 0F08 0F1F 0F35 0F4C 0F64 0F7B 0F93 0FAB 0FC3

120
121
122
0FDB
0FF2
120
121
122