Please do not upload this copyright pdf document to any other website. Breach of copyright may result in a criminal conviction.

This Acrobat document was generated by me, Colin Hinson, from a document held by me. I am unable to find the author to request permission to publish, so beware, this is not totally my copyright. It is presented here (for free) and this pdf version of the document is my copyright as it is part of my database of documents. If you believe the document to be under other copyright, please contact me.

The document should have been downloaded from my website https://blunham.com/Radar, or any mirror site named on that site. If you downloaded it from elsewhere, please let me know (particularly if you were charged for it). You can contact me via my Genuki email page: https://www.genuki.org.uk/big/eng/YKS/various?recipient=colin

You may not copy the file for onward transmission of the data nor attempt to make monetary gain by the use of these files. If you want someone else to have a copy of the file, point them at the website. (https://blunham.com/Radar). **Please do not point them at the file itself as it may move or the site may be updated.**

It should be noted that most of the pages are identifiable as having been processed by me.

I put a lot of time into producing these files which is why you are met with this page when you open the file.

In order to generate this file, I need to scan the pages, split the double pages and remove any edge marks such as punch holes, clean up the pages, set the relevant pages to be all the same size and alignment. I then run Omnipage (OCR) to generate the searchable text and then generate the pdf file.

Hopefully after all that, I end up with a presentable file. If you find missing pages, pages in the wrong order, anything else wrong with the file or simply want to make a comment, please drop me a line (see above).

It is my hope that you find the file of use to you personally – I know that I would have liked to have found some of these files years ago – they would have saved me a lot of time !

Colin Hinson
In the village of Blunham, Bedfordshire.

DISkASSEMBLER

```
AX      LIMI  >ØØØ2              >Ø3ØØ,>ØØØ2
EE      LIMI  >ØØØØ              >Ø3ØØ,>ØØØØ
AC      MOVB  *R13,R9            >D25D
AB      JLT   AM                 >11Ø5
        MOVB  R9,R4              >D1Ø9
        SRL   R4,12              >Ø9C4
        MOV   @AN(R4),R5         >C164,>ØC36
        B     *R5                >Ø455
AM      CLR   R4                 >Ø4C4
        MOV   R9,R5              >C149
        ANDI  R5,>Ø1ØØ           >Ø245,>Ø1ØØ
        BL    @AO                >Ø6AØ,>Ø77A
        SWPB  R4                 >Ø6C4
        MOV   R1,R3              >CØC1
        MOV   RØ,R2              >CØ8Ø
        CI    R9,>AØØØ           >Ø289,>AØØØ
        JL    AP                 >1AØ9
        COC   @AQ,R9             >226Ø,>ØØ3Ø
        JNE   AR                 >16ØC
        MOV   R13,R1             >CØ4D
        MOVB  *R1,RØ             >DØ11
        DEC   R1                 >Ø6Ø1
        BL    @AS                >Ø6AØ,>Ø7AA
        JMP   AT                 >1ØØ8
```

DISkASSEMBLER

MANUAL WRITTEN BY

R. KENT THOMSON

IN COLLABORATION WITH

THOMAS S. FREEMAN

The Information contained in this book is subject to change without notice.

Millers Graphics shall not be liable for technical or editorial errors or omissions contained herein; nor for incidental or consequential damages resulting from the furnishing, performance, or use of this material or product described by this manual.

This manual contains information protected by copyright. All rights are reserved. No part of this manual may be photocopied or reproduced in any form without prior written consent from Millers Graphics.

DISkASSEMBLER Software and Associated Manual
(Produced in the United States of America)

Copyright 1986

by

Millers Graphics
1475 W. Cypress Ave.
San Dimas, CA 91773

TABLE OF CONTENTS

Introduction to DISkASSEMBLER 1

Quick Reference Guide 2

Loading/Running DISkASSEMBLER 3

Default Options 3

Some General Information 4

Page One 6
- Function Keys During Input 6
- Disassembly From Disk 7
- The "File Information" 8
- Options 10
- Disassembly From Memory 13
- Disassembling Device Service Routines 14

Page Two - Data and Text Blocks 16

Page Three - Data Following Subroutines 17

Page Four - Specifying Output Devices 19
- Output to Printer 19
- Output to Disk 20

Function Keys During Passes 21

First Pass 23

Second Pass 25
- REF/DEF/END 26
- EQU List 27
- Symbol Table List 27

Reassembly 28
- Dis/Fix 80 File Without REFs 28
- Dis/Fix 80 File With REFs 30
- Program Image - Standard Loader File 32
- Program Image - Custom Loader File 33
- Memory Disassembly 33

Appendix A - Save Utility Tutorial 34
Appendix B - Files Larger Than 48 Sectors 42
Appendix C - Memory Maps 44
Appendix D - Common Equates & Subroutines....... 52

INTRODUCTION TO DISkASSEMBLER

DISkASSEMBLER is a program that allows you to disassemble assembly language object code (referred to as code in this manual). It is designed for use as a learning tool, allowing the novice or experienced assembly language programmer to investigate the techniques used in existing code and to allow you to modify it to suit your needs.

Unlike other disassemblers code can be disassembled directly from disk, thus the name DISkASSEMBLER, or from an area of memory in the TI 99/4A, such as the Device Service Routines. DISkASSEMBLER is a smart disassembler in that it will automatically; insert labels as needed, split up Data blocks, correctly insert Data for Subroutines and set up the Data for BLWPs.

As code is disassembled, it can be saved on disk or printed; simultaneously if you wish. Files saved to disk can be accessed later using the TI Editor/Assembler or TI Writer, customized by the user, and reassembled. Or, portions can be extracted for use in other programs.

In concert with programs like Miller Graphic's Explorer and Advanced Diagnostics, or the Gram Kracker, you'll have the tools you need to take advantage of the limitless capabilities TI built into the 99/4A.

SYSTEM REQUIREMENTS

To load and run DISkASSEMBLER you must have memory expansion, at least one disk drive, and one of the following cartridges:

Extended Basic
Mini Memory
Editor Assembler

QUICK REFERENCE GUIDE

Function and Control Keys During Input

FCTN 1	- Delete a Character
FCTN 2	- Insert a Space Character
FCTN 3	- Erase Input Line
FCTN 4	- Abort Back to the Top of Page One
FCTN 5	- not used
FCTN 6	- Go to the Next Input Page
FCTN 7	- not used
FCTN 8	- not used
FCTN 9	- Back One Input Page
FCTN 0	- not used
FCTN =	- not used - Quit is Turned Off
CTRL 1	- Change Screen and Text Colors
CTRL =	- QUIT DISkASSEMBLER (Top of Page One Only)

Option List

R	- Put in R before Register Number
B	- Add Basic Bias to TEXT (ASCII) Display
T	- Place TEXT before DATA
D	- Enable DSR Disassembly
X	- Extend Label Checking
V	- Override Automatic BLWP DATA for EXTERNAL REFs
M	- Input/Output from/to Myarc Floppy Disk Controller
G	- Input/Output from/to RAM Disk or Hard Disk

Function and Control Keys During Disassembly

FCTN 1	- DATA before TEXT
FCTN 2	- TEXT before DATA
FCTN 3	- not active
FCTN 4	- CANCEL Printer Output
FCTN 5	- Toggles Left and Right Screen Display
FCTN 6	- not active
FCTN 7	- PRINTER ON/OFF
FCTN 8	- OUTPUT FILE ON/OFF
FCTN 9	- ESCAPE Disassembly - Back to Top of Page One
FCTN 0	- RETRO SCREEN CAPTURE ON/OFF
FCTN =	- Scroll Speed Fast/Slow
CTRL 1	- Change Screen and Text Colors
CTRL =	not active

LOADING/RUNNING DISkASSEMBLER

Place the disk in any disk drive and follow the directions below for the specific cartridge you are using.

LOADING FROM THE EDITOR ASSEMBLER

Select 3 LOAD AND RUN from the E/A menu and type in:
DSKx.DISKASSEM (where x = the drive no. that the DISkASSEMBLER disk was placed in)
DISkASSEMBLER will load and auto start.

LOADING FROM EXTENDED BASIC

Place the DISkASSEMBLER disk in drive one and select Extended Basic from the menu. DISkASSEMBLER will auto load and run. Or, go into Extended Basic and execute CALL INIT. Then type in:
CALL LOAD("DSKx.DISKASSEM") and press Enter.
DISkASSEMBLER will load and auto start.

LOADING FROM THE MINI MEMORY

Select MINI MEMORY from the menu and then select 1 LOAD AND RUN from the MINI MEMORY menu and then type in:
DSKx.DISKASSEM (where x = the drive no. that the DISkASSEMBLER disk was placed in)
DISkASSEMBLER will load and auto start.

DEFAULT OPTIONS - DISKCONFIG FILE

On the DISkASSEMBLER diskette you will find the file called DISKCONFIG. This file contains the default colors, default options (see "Page One") and the default printer name. You can load this file into the E/A Editor or the TI-Writer Editor and change these defaults to suit your taste. The file will appear as follows when loaded into an Editor:

```
F4
R
PIO
```

COLORS - The FIRST LINE in the file MUST contain the screen and text colors you would like for DISkASSEMBLER. They are in Hex and they follow the same color values used for text mode in the E/A module. Examples:

F4	= White on Dark Blue
17	= Black on Cyan
15	= Black on Light Blue
FC	= White on Dark Green

DEFAULT OPTIONS - The SECOND LINE in the file MUST contain the Default Options (see Page One - Options). They can be in any order and they will appear in the "Options" input field when you first use DISkASSEMBLER. If you do not want any options then leave this line blank. Examples:

```
R
RM
G
```

PRINTER NAME - The THIRD LINE in this file MUST contain the Default Printer name that is used when you press FCTN 7 - Printer on Page Four or during the Passes. You can type in any valid output device on this line but the name must not be any longer than 38 characters. Examples:

```
PIO
RS232.BA=1200.DA=8.EC.TW
RS232/2.BA=9600.LF
```

SOME GENERAL INFORMATION

DISkASSEMBLER - is a two-pass disassembler. This means that the code requires two independent "looks" or "passes" to be completely disassembled with labels. For each pass, you can enter information to tell DISkASSEMBLER the specifics on how you want the code disassembled. This information is entered on four different screens referred to as "PAGES." You'll be introduced to the four different Pages in the sections coming up.

There are two ways code may be disassembled. You can disassemble machine language for its assembly language representation (referred to as symbolic or source code) or you can accomplish what is referred to as "block" disassembly. Block disassembly provides a listing, with line numbers, of the existing machine code in its hexadecimal (hex) representation along with its ASCII equivalent. This has specific uses that are described later.

RORG or AORG CODE - Throughout this manual there are a number of references to RORG or AORG CODE. When code is of the RORG type (Relocatable Origin, Dis/Fix 80) it will be loaded into the first available free address according to the pointers for the module that loads it (E/A, XB or Mini Mem). When code is of the AORG type (Absolute Origin) it will be loaded exactly where the programmer specified and the pointers to the first and last free addresses will not be adjusted by the module's loader. RORG code on the disk contains a bias or additive value for addressing. For example, an address reference in the file may be >00C2 this and the RORG Tag will instruct the loader to add >00C2 to the start load address for this reference. So, if the file starts to load at >A000 this reference will become >A0C2. In AORG type files the absolute or exact address is referenced so it would contain >A0C2 instead of >00C2 for the above example. This is why you are instructed to use a "Loads At" address of >0000 for RORG code IF you want to reassemble it as an RORG program.

OPCODE and OPERAND - These two terms are also used throughout the manual and they refer to the following:

> **OPCODE** is the INSTRUCTION to be performed, i.e. MOV, MOVB, BLWP etc. or an assembler directive i.e. AORG, RORG etc.
>
> **OPERAND(s)** is the memory location(s), value or register(s) to be used by the instruction.

```
AA     MOV  R1,R2
^      ^    ^  ^
|      |
|      |    Operands
|      |
|      Opcode
|
Label Field
```

TAG - Tags are single byte flags placed in Dis/Fix 80 Object Code assembly files to instruct the loader how to handle the next word(s). DISkASSEMBLER uses these Tags in the same manner when it disassembles these files. See pages 238 through 241 in the Editor/Assembler manual for additional information on the various Tags and their use.

PAGE ONE

Load and run the program in the manner previously described. After the Title Screen the following will appear on the screen. This is the beginning or Top of Page One.

```
                  DISkASSEMBLER
        ====================================
        Input File Name or Null for Memory
        DSK1.
        ====================================
```

Once DISkASSEMBLER is loaded you can remove the disk from your drive and replace it with a diskette containing the file you want to disassemble. If you are going to disassemble a file to another diskette you should have a couple of formatted diskettes ready for use. Before going any further, you should know something about the function keys active during this (the input) phase of the program.

Function and Control Keys During Input

Below are the function keys active during the input phase of the program. They work when you are entering information on pages 1-4. They are redefined during the disassembly or "passes" phase of the program and you'll be reintroduced to them later when they change.

FCTN 1 - Delete a Character - deletes a single character from the current input line or field.

FCTN 2 - Insert a Character - inserts a single space character into the current input line or field, characters at the very end of the input field will be deleted.

FCTN 3 - Erase Input Line - erases the entire input line or field.

FCTN 4 - Abort Back to the Top of Page One - takes you from any input page to the very beginning of Page One and cancels your input.

FCTN 6 - Proceed to the Next Input Page - instructs DISkASSEMBLER to go to the next input page and accept all the inputs on the current page.

FCTN 9 - Back One Input Page - instructs DISkASSEMBLER to go to the previous input page and accept all the inputs on the current input page.

CTRL 1 - Change Screen and Text Colors - toggles through 7 different text and screen color combinations.

CTRL = - QUIT DISkASSEMBLER (Only when the cursor is in the "Input File Name" field on the Top of Page One)

DISASSEMBLY FROM DISK

When selecting files for disassembly from disk, these files must be Display/Fixed 80 (Dis/Fix 80) or memory image (Program) files. A Program file cannot be longer than 47 sectors (listed as 48 in a disk catalog) and it MUST be an Assembly file to obtain a usable output. You must have 2 or more disk drives (or 1 drive and another output device) to disassemble a medium to large Dis/Fix 80 file since the file is worked on one record at a time. If you have a single drive system and if the Dis/Fix 80 File is small enough, you can put it on the same diskette as the one to be used for DISkASSEMBLER's output. Program type files, however, are loaded into memory before DISkASSEMBLER works on them so they can easily be disassembled on a single drive system.

Basic and X-Basic can also store their programs in Program format. Disassembly of these files is not recommended as the result is not usable in a pure assembly environment. You could, however, disassemble them solely as DATA and Text (block disassembly) just to see what's in the file (see page 16 for additional information on block disassembly).

After inputting the device and filename (i.e. DSK1.EDIT1) for the file you wish to disassemble, information particular to that file is displayed; something similar to what is shown below:

```
File Information

    Type      : Dis/Fix 80
    Code      : Uncompressed
    Length    : >07F4 Bytes Reloc.  Code
    Loader    : Standard
    Loads at  : >A000 RORG
    Sectors   : >00
    Options   : R

    E/A >A000  XB >24F4  MM >7118
```

The last line at the bottom of the screen indicates the default first free address that the Editor/Assembler, X-Basic, and Mini Memory modules use for loading and executing RORG code (Relocatable Origin code). It was placed here for easy reference. Note that the Mini Memory module will load the program at >7118 if it is small enough to fit in the Mini Mem Ram, otherwise it loads it at >A000.

The first four lines contain file information. The last three allow you to change parameters that affect the way code is disassembled. Each is discussed below.

Type The file type is indicated as Dis/Fix 80 or Program Image. Program Image files are files that contain an EXACT copy of the memory as it appears when the program is loaded into memory. There are no "tags" and the origin, or load address, is absolute. It is currently not relocatable code but it can be converted into RORG code, upon reassembly, with DISkASSEMBLER. This format is used in some applications for its compactness and its faster load times. Basic and X-Basic programs may also be stored in program image format. These Basic files can be disassembled but the result is unusable as an assembly program.

Code Dis/Fix 80 files will be compressed or uncompressed code and Program type files will be memory image code. Compressed code is stored in Hex and as the name implies it takes up less room than uncompressed code. It also loads slightly faster than uncompressed code, but it can not be loaded by the Extended Basic object code loader. Uncompressed code is stored in ASCII format, each nibble of the actual code is represented by one ASCII byte (0-F). Since it is in ASCII it can be loaded into the E/A editor and/or printed out.

Length This can be an indication of how much room some programs require in memory. The information is read from the beginning of the disk file if it is available, as in relocatable Dis/Fix 80 files, or standard format program image files. If the code is Dis/Fix 80 and it is all AORG'd, the length will be >0000 Relocatable Bytes since no value was placed here by the Assembler. With RORG code this value is used by the loader to determine if there is enough free space to load the file. With Program Image code this value is used by the loader to determine how many bytes to move out of VDP RAM, where the program is first loaded, into memory expansion. With AORG code this value is not needed by the loader. It is possible with Dis/Fix 80 files to have both RORG and AORG code mixed, such as the file named FORTH. In this case the value will be just the number of bytes of RORG code and it will not reflect the number of bytes of AORG code.

Loader Loaders will be listed as either Standard or Custom. Custom applies to Program Image files that are loaded by another loader written by the programmer, instead of the loader(s) built into the various modules. An example of a Custom Loader file is FORTHSAVE. All Dis/Fix 80 files and all files that can be loaded and executed by option 5 in the Editor/Assembler or option 3 in TI-Writer are Standard, such as EDIT1, EDITA1, EDITA2, ASSM1, ASSM2, FORMA1, and FORMA2. Standard also indicates that the first 3 words of a Program file contain:

1 A flag indicating whether another file follows, >0000 or >FFFF (this word tells DISkASSEMBLER the file is Standard. NOTE: If it is Custom and this word is >0000 or >FFFF you will get some strange values for "Length" and "Loads At")
2 The total number of bytes in the file (if it is Standard DISkASSEMBLER uses this for "Length")
3 The address to start loading the fourth word at (if it is Standard DISkASSEMBLER uses this for "Loads At")

Loads At Tells us where the beginning of the file will start to load and whether it is RORG or AORG. If the code is AORG (Absolute Origin), Dis/Fix 80 or Program type, you will see the address followed by the word AORG on this line. If it is RORG (Relocatable Origin), you will see >A000 followed by RORG. >A000 is the Editor Assembler module's default first free load address in High Memory Expansion.

If the code is RORG, type in >0000 if you wish to reassemble it as RORG. Or, type in one of the module defaults (see bottom of screen) if you wish to see how it will be located in memory when using that module. Additional work is required for reassembly of RORG'd code. Using >0000 will make the job a little easier. You can relocate AORG code to another location or convert it into RORG code by changing the AORGs prior to reassembly. But, be careful of the labels in the Equate list or DATA statements that reference original addresses.

If the code is of the Custom Loader Program type DISkASSEMBLER will put in >A000 as the default load address. You will need to disassemble and analyze the Custom Loader file to find out where it really "Loads At".

Sectors This value will either be >00, >2F or the actual number of sectors. For Dis/Fix 80 files this value will always be set at >00 and it SHOULD BE left there. A value of >00 tells DISkASSEMBLER to disassemble the file until it comes to the END OF FILE marker. The >2F value will show up for Custom Loader Program files loaded from a RAM or HARD disk. These devices do not use the disk buffer space in VDP RAM to store file information so DISkASSEMBLER uses this value since it is the maximum file size allowed. In this case you should set this value to 1 less than the catalog size, in HEX. This is generally the only time you will need to change this value.

For Standard Loader Program files loaded from any device and for Custom Loader Program files loaded from a floppy disk this value will reflect the size of the file. In this case you can change it if you want to disassemble just a portion of the file. However, DO NOT set it to >00 for these file types, since they do not contain an END OF FILE marker!

Options There are eight options you can select. When selecting options, type the capital letters corresponding to the options you want consecutively with no separators (spaces or commas). All options can be used at once and may appear in any order.

R Generates an 'R' before the register number in the output. In operations where registers are used, the 'R' is optional. This is correct in either case and is up to the programmer. However, if the code has been disassembled using this option, you must select the R option from the E/A when reassembling.

B For use with code that includes basic bias. If the code you are disassembling has text that it passes to basic, that is then displayed, it requires the addition of a screen bias (>60). Using this option adds the bias to the ASCII representation of the code in the output so you can read the text. This is rare and it is not needed for reassembly.

T Prints TEXT first in block disassemblies. Block disassembly is discussed further in the section on PAGE TWO - DATA AND TEXT BLOCKS.

D Tells DISkASSEMBLER that you want to disassemble a Device Service Routine. See PAGE ONE - DISASSEMBLING DEVICE SERVICE ROUTINES.

X To extend label checking outside the file's address range. This option should be used when disassembling "mixed relocatable and absolute origin" or "out of order origin" code and it will insure that labels are accurately defined in the disassembled output. "How do I know if there is mixed code," you say? DISkASSEMBLER will tell you at the end of the First Pass, WHICH MUST BE REDONE.

You may also want to use the option if you are disassembling a program that is contained in more than one file, i.e. ASSM1 or ASSM2. This will instruct DISkASSEMBLER to generate a label and Equate list, on the Second Pass, for any references to the other file. However, it is not needed for proper reassembly since DISkASSEMBLER will generate the correct address in the operand field instead of a label.

V This option overrides the automatic insertion of DATA statements for BLWPs. It MUST BE turned on at the beginning of the Second Pass when routines are branched to using BLWP and the locations are REF'd, which will show up at the end of the First Pass. This includes DSRLNK, GPLLNK, KSCAN and the VDP utilities such as VWTR, VMBW, and any routines contained in a separately loaded file.

In a non-REF'd file the the operand field of a BLWP points to an address that contains 2 words of data; the workspace for the routine and the start address. Without the V option DISkASSEMBLER will automatically track up to 15 of these pointers and treat the 2 words that they point to as as DATA. In a REF'd file the address pointed to by the operand is the address of the previous REF'd statement in the REF chain and not the actual 2 words of data. After the file is loaded the Tagged Object Code loader uses these pointer chains to resolve the External REFs found in a file. See Reassembly for more info.

NOTE: The following M or G options should be in your default parameters file, DISKCONFIG, since they are used by DISkASSEMBLER when the input file is opened and this happens before you have the opportunity to change them. If they are not in your default file or if you want to change them, tell DISkASSEMBLER to disassemble memory by inputting a null file name, change them, press FCTN 9 and then type in your file name. AFTER the input file is opened, you should change the M or G if the output file will be to a different device. (i.e. Input from a Myarc floppy and then output to a Ram Disk). On input these options tell DISkASSEMBLER where to get the "File Information", on output they tell it where to get the information it needs to split up the output into 65 sector files.

M Signals the use of a Myarc floppy disk controller as the input or output device. Unfortunately, this controller does not follow the TI standard for placement of file information in VDP and/or Scratch Pad Ram so DISkASSEMBLER must look elsewhere for it. Placing the M option in your default file instructs DISkASSEMBLER where to look for this information.

G Signals the use of a RAM disk or Hard disk for input or output and overrides the M option. These devices do not use the disk buffer space in high VDP Ram to store information on the file just opened so DISkASSEMBLER must use a different method of obtaining this information. This is especially important for Custom Loader Program files. If you do not specify the G option on this file type you will get an incorrect "Length" and incorrect "Sectors" and the file will NOT disassemble properly.

NOTE: The M or G option must be enabled BEFORE the input file is opened and/or changed to match the output device. The X option must be enabled BEFORE the First Pass since labels are actually resolved on this pass. The R, B, T and V options can be enabled on either the First or Second Pass.

DISASSEMBLY FROM MEMORY

When the prompt "Input File Name or Null for Memory" appears, use FCTN 3 to erase "DSK1." and then press ENTER. The "File Information" will appear as shown below:

```
File Information

          Type      : Disassemble Memory
          Code      :
          Length    : >
          Loader    :
          Loads at  : >
          Sectors   : >
          Options   : R

          Start     : >0000
          Finish    : >0000
```

The cursor will appear over the R option to allow selection of additional options. Pressing Enter will move it down so you can select hex values for Start and Finish to specify the range of memory you wish to disassemble. The allowable range is >0000 through >8000 and >A000 through >D000. However, DISkASSEMBLER uses >3000 through >3FFF in low memory expansion as a buffer area, so you will get some strange results if you disassemble this section of memory.

NOTE: If you specify a Start or Finish address that is not in the allowable range, DISkASSEMBLER will not accept it and it will return you back to the Top of Page One.

ALSO NOTE: The "Finish" address is NOT inclusive. So, inputting >0000 for the "Start" and >2000 for the "Finish" will actually disassemble >0000 through >1FFE, since 9900 assembly code must be on even word boundaries.

DISASSEMBLING DEVICE SERVICE ROUTINES (DSRs)

Place DISkASSEMBLER in the "Disassemble Memory" mode and place a D in the option list. Next specify an address range in the DSR space of >4000 through >6000. Before disassembling a DSR, you must turn it on by entering its Communication Register Unit (CRU) base address. A prompt for the CRU Base will appear at the bottom of Page One if you have specified the D option and a start address between >4000 and >6000. Example for the RS232 Card:

```
File Information

        Type     : Disassemble Memory
        Code     :
        Length   : >
        Loader   :
        Loads at : >
        Sectors  : >
        Options  : RD

        Start    : >4000
        Finish   : >5000

        CRU Base : >1300 >0000 >0000 >0000
```

You may specify up to four CRUs at a time. Multiple CRUs can be specified for devices like the CorComp and Myarc Disk Controllers, which contain "bank switched" DSR Roms. This allows you to activate Bank 2 for these devices. When a CRU address is specified, the code required to execute that DSR is enabled in memory between addresses >4000 and >5FFF. If you don't specify the CRU base, the code isn't transferred and you will be disassembling blank memory (>0000 or >FFFF depending on the console type). When you go to input "Page Two" DISkASSEMBLER will turn on the DSR and move it to a buffer area and then turn it back off. This allows the DSR to be used as an output device while its code is being disassembled.

NOTE: The valid CRU Base range is >1000 through >1FFF. If you use CRU Base values lower than >1000 unpredictable results may occur and you may lock up your console. This may happen because these lower CRU Bases are used internally by the console. Also, if you have the Myarc Ram disk, specifying CRU Base values of >1002, >1004, >1006 etc. will cause the card to do a 32K bank swap and lock up DISkASSEMBLER.

The Standard CRU addresses and devices assigned by TI are shown below. Your system may contain other third party cards that have their own CRU Base.

CRU Base	**Assigned Device**	**Your System**
>0000	Internal console use	
>0400	Unassigned	
>1000	RAM disk or hard disk (Some can be changed)	____________
>1100	Floppy disk controller	____________
>1100 & >1116	Bank 2 - CorComp	____________
>1100 & >1106	Bank 2 - Myarc	____________
>1200	Internal modem	____________
>1300	RS232 1 and 2 and PIO 1	____________
>1400	Unassigned	____________
>1500	RS232 3 and 4 and PIO 2	____________
>1600	Unassigned	____________
>1700	Unreleased HEX-BUS adaptor	____________
>1800	TI Thermal printer	____________
>1900	EPROM Programmer	____________
>1A00	Unassigned	____________
>1B00	Unreleased Debugger Board	____________
>1C00	Video controller	____________
>1D00	IEEE 488 Controller card	____________
>1E00	Unassigned	____________
>1F00	P-Code card	____________

PAGE TWO - DATA AND TEXT BLOCKS

This page allows you to specify up to 15 ranges of addresses for "block disassembly". These ranges should be specified for parts of the code that contain lines, or blocks of Data and/or Text. When specifying these ranges keep in mind that the "Finish Address" will NOT be included in the DATA or TEXT block.

Data and Text Blocks

Start Address	Finish Address
>0000	>0000
>0000	>0000
>0000	>0000

As DISkASSEMBLER looks through code, it can't differentiate data and text from assembly language object code. It is simply looking at a list of hex numbers. If no address blocks are specified, data and text get disassembled as if they were assembly language mnemonics and any illegal hex values are converted into DATA. The result is source code with parts that don't make any sense. This may cause a great deal of confusion, when inspecting the disassembled code, for even the experienced programmer. However, even though it doesn't make any sense to you, it will still reassemble properly.

When you specify a range of addresses for block disassembly, the hex representation of the code, with its ASCII equivalent, appears as DATA in the output. You then have the proper representation of the original source code (excluding any comments) which will make it much easier to follow and understand. You can use the T option, mentioned earlier, to indicate whether you want the hex (data) or ASCII (text) representation to appear first (left most) in your output. We recommend that you leave out the T option and allow the DATA to be first.

On the Second Pass DISkASSEMBLER will automatically split up these DATA blocks and insert labels in front of them, as required by the program being disassembled. This automatic feature will save you the tedious task of doing it yourself.

You surely won't know where the data and text are located when disassembling a program for the first time. In this case, just skip this Page and the next Page using FCTN 6 and go on to Page Four.

PAGE THREE - DATA FOLLOWING SUBROUTINES

For the same reasons discussed in PAGE TWO - DATA AND TEXT BLOCKS, you don't want to disassemble data following a subroutine branch, such as BLWP @AA or BL @AA. To avoid this, you can specify up to 15, non-REF'd, internal subroutine addresses followed by up to >FFFF (there will never be this many, usually 1 to 4) words of data.

Data Following Subroutines

Sub Address	No. of Words
>0000	>0000
>0000	>0000
>0000	>0000

Subroutines are accessed in assembly language using the BL and BLWP commands and they MAY be followed by one or more words of data. This is one convenient method of transferring information (data) to the subroutine. If there is data following one of these commands, you will want to avoid disassembling it by specifying the proper information on this page. GPLLNK and DSRLNK are example BLWP routines that use a word of data after them but, the data is usually a low value and it will not be interpreted as code by DISkASSEMBLER. However, user defined routines may use larger numbers which could be interpreted as code.

You can recognize whether data is used following subroutines by analyzing the First Pass printout to find the start address of the routine. For a BL, this is the operand (e.g. BL @>A0CE, look at address >A0CE). When using a BLWP, the operand address in the BLWP points to two words of data that specify the workspace and the start address of the subroutine. If the command was BLWP @>A0CE, you would get the two words of data at >A0CE. The first word is the workspace and the second word points to the start of the subroutine.

Next look through the subroutine's code for some MOV or MOVB instructions to determine how many words of data, if any, should follow the subroutine branch. If the subroutine is branched to with a BL, the return address (pointer to the next word or instruction following the BL @>xxxx) is stored in Register 11. If this type of subroutine branch uses data it will usually move it with a MOV *R11+,xxx instruction or two MOVB *R11+,xxx instructions.

If the subroutine is branched to with a BLWP, the return address is stored in Register 14 of the new workspace. If this type of branch uses data it will usually move it with a MOV *R14+,xxx instruction or two MOVB *R14+,xxx instructions. The number of MOV *R14+ or pairs of MOVB *R14+ instructions that you find determine the number of data words following a branch to a subroutine. Example:

```
MOVB *R14+,R0
MOVB *R14+,@>A00F
MOV  *R14+,R1
```

Two words of data were used, one for the two MOVB and one for the MOV.

After you have the number of words for each subroutine you can fill in the input fields on this page.

The "Sub Address" is ALWAYS the value in the operand field PROVIDED the subroutine IS NOT REF'd at the end of the file. If it is REF'd you can not specify that subroutine since, as we discussed before, the operand is just a pointer in the REF chain. And, since it is REF'd, every operand for that BLWP will a have different value pointing to the next BLWP in the REF chain. So for a non-REF'd BL @>A0CE or a BLWP @>A0CE you would use >A0CE as the "Sub Address".

The "No. of Words" is the number of words of data that you found the subroutine moving.

This input section of DISkASSEMBLER is most useful for Dis/Fix 80 files without REF's and Program Image files that contain user written subroutines.

NOTE: Failure to indicate any addresses for DATA FOLLOWING SUBROUTINES or DATA AND TEXT BLOCKS will not cause any errors with respect to reassembling the code.

PAGE FOUR - SPECIFYING OUTPUT DEVICES

When you have made it this far, you are ready to start the actual disassembly. Before doing so, you have the option to select devices for output. The output will always appear on the screen and you can also output to a disk or printer, separately or simultaneously.

```
PRINTER OFF  OUTPUT OFF  SCROLL SLOW
====================================

      Press Enter to Begin or

      Press      To

       F7    Set Printer Output

       F8    Set Output Device

       F=    Select Scroll Speed
```

Press FCTN 7 to specify and turn on output to printer and/or FCTN 8 to specify and turn on output to disk. FCTN = will toggle the scroll speed between Fast and Slow. Watch the status line at the top of the page change as you make your selections. Once your selections are made press ENTER to start disassembly.

OUTPUT TO PRINTER (FCTN 7) - When you press FCTN 7 the following will appear at the bottom of the screen with the cursor sitting on the first character of the device name.

```
===================================
Device Name: (Printer)
PIO.
```

The default device name is taken from the DISKCONFIG file. At this time you can press Enter to use the device shown, type in a new one, or you can press FCTN 4 or FCTN 9 to abort this option. Once the device is opened the status line at the top of the screen will show the device ON. FCTN 7 is recommended for use with a printer, and not to a disk drive, since it outputs one contiguous file. It is also recommended that you output the First Pass to a printer for reference on the Second Pass.

NOTE: If the device (filename) you select was one previously selected the prompt "Last File - Add to it Y/N:" will appear. If you are using this option to output to printer just press Enter. If you used this option to output to disk press Enter or Y to append to the end of the last file or press N to change the device.filename.

OUTPUT TO DISK (FCTN 8) - When you press FCTN 8 the following will appear at the bottom of the screen with the cursor sitting on the first character of the device filename area.

```
===================================
Device Name: (Output)
DSK1.
```

When specifying output to disk, you must have an initialized disk in the drive you select. DISkASSEMBLER will save the output to disk in 65 sector Dis/Var 80 blocks with consecutive ASCII filenames like TESTA, TESTB, etc. We recommend that you end the filename with the letter "A" so you can save up to 26 consecutive files to disk before running into an ASCII character like "[". The name of each disk file created will be in the beginning of the file, and on the printout, with an asterisk in front of it for your reference. You can then load these files using the Editor/Assembler or TI Writer for any additional manipulation.

NOTE: If the filename you select already exists on disk, you will be asked if you wish to add to it (Y) or erase (N). If you select Y, new information will be appended to the existing file. If you select N, the old information in this file, as well as the other consecutive files (i.e. TESTB TESTC etc.), will be written over and <u>YOU CAN NO LONGER APPEND TO ANY FILES.</u>

ALSO NOTE: Currently, the Myarc Ram disk can not append to EXISTING files. Do not "Add" to its files or you may wipe out ALL the files on your Ram disk. When prompted to add (Y) or erase (N), select N and DISkASSEMBLER will then <u>OPEN ALL FILES IN OUTPUT MODE</u> which, in effect, will erase the file before it writes to it. You must also select the G option on Page One to tell DISkASSEMBLER your output is going to a RAM Disk so it can properly split them up into 65 sector blocks for easy editing, otherwise, it may lock up and create a large number of 1 sector files.

Before pressing ENTER to start disassembly, you should be aware that the FCTN keys, while the program is disassembling code (making a pass), are different than when you were entering information on Pages One through Four.

FUNCTION KEYS DURING PASSES

FCTN 1 Toggles to DATA first during block disassembly. Overrides the T option from Page One.

FCTN 2 Toggles to TEXT first during block disassembly. Selecting the T option on Page One also specifies TEXT first.

FCTN 4 Aborts printer output if it is turned ON. It will also cancel a FCTN 7 or FCTN 8 option selection when the option's "Device Name:" input line is displayed.

FCTN 5 Toggles screen back and forth from left to right since the output is 78 columns.

FCTN 7 Allows you to turn your output to printer ON and OFF. When it is turned "ON" DURING disassembly the output will begin with the next line to be scrolled onto the screen. Once it is turned ON, selecting this option again will turn it OFF and close the file. If it is turned back "ON" again the prompt "Last File-Add to it Y/N" will appear. Press Enter or Y to continue printing. If you have selected a disk file with FCTN 7, press Enter or Y to append to the file, or press N to change the filename.

FCTN 8 Allows you to turn your output device ON and OFF. When it is turned "ON" DURING disassembly, the output will begin with the next line to be scrolled onto the screen. Once it is turned ON, selecting this option again will turn it OFF and close the file. If it is turned back "ON" again the prompt "Last File-Add to it Y/N" will appear. Press Enter or Y to append to the filename displayed. Press N to change the filename. NOTE: FCTN 8 outputs 65 sector Dis/Var 80 files and automatically closes the file at 65 sectors, increments the filename and opens a new file. DO NOT use this option to output to printer because it will change the printer name after 192 records (65 sectors) have been printed and you will halt DISkASSEMBLER with a "Bad Device Name" or "Bad File Attribute" error.

FCTN 9 Escape or Abort disassembly - once you press Enter at the prompt, this will take you back to the Top of Page One. It will also cancel a FCTN 7 or FCTN 8 option selection when the option's "Device Name:" input line is displayed.

FCTN 0 Retro Screen Capture. Similar to FCTN 7, but captures all lines that were shown on the screen. This FCTN remains on until FCTN 0 is pressed again. When the screen contents are captured, the message "Screen Printed" will appear. If you want to continue printing, press ENTER. If you want to stop, select FCTN 0 again and the output file will be closed, then press ENTER to continue. When you select this option again, you will be allowed to append to the previous one or select a new one.

FCTN = Allows you to toggle the screen scroll speed between Fast and Slow.

CTRL 1 Rotates through screen colors.

A FEW NOTES ABOUT OUTPUT DEVICES:

1. If you turn on more than one output option (i.e. FCTN 7, FCTN 8 and/or FCTN 0) to the SAME disk drive you will be creating BADLY fractured files. The DSR automatically uses the lowest free sector so, if you have two output or append files opened on the same diskette they will generate records on every other sector. This will cause the Data Chain Pointer Blocks in the File Descriptor Record (File Header) to fill up and you receive an "Out of Space" error.

2. Since DISkASSEMBLER automatically creates 65 sector files for the "Output Device" and increments the name you will find that a 33 sector Program file will create 15 files as an average (i.e. TESTA through TESTO). The last two files generated are usually less than 65 sectors so you will need approximately 875 free sectors to disassemble these files. DISkASSEMBLER will automatically close the file when the disk runs out of space and allow you to change diskettes. This way you can easily disassemble these files on a single sided single density disk system - but it will require the better part of 3 blank diskettes.

After you press ENTER on Page Four disassembly will start. If you have specified output to a printer on Page Four and it is not turned on, your computer will appear to lock up. TURN THE PRINTER ON or hit FCTN 4 to escape. Once disassembly starts you will see the "File Information" scroll onto the screen when disassembling files. If you are disassembling memory this information will not be displayed. The first line of code will always be an AORG or RORG statement with the address specified in "Loads At" or "Start" on Page One.

We recommend that when disassembling a file or an area of memory for the first time, that you output the First Pass to a printer. This will provide you with some useful information that may lead you to starting over to make a cleaner First Pass by specifying some DATA or TEXT Blocks and/or DATA following subroutines. Reassembly of the First Pass is possible, although not advised, so you would rarely output the First Pass to disk.

More specifically, a detailed look at the printout of the First Pass will show you where the data and text statements, and subroutine calls using BL and BLWP are located. This printout will list the addresses in the left most column. You could then redo the First Pass, specify addresses for data and text blocks on Page Two and, if needed, specify the start address (operand) for the BL or BLWP and the number of words of data following these subroutines using Page Three. This will result in a more readable First Pass. Or, you could make these specifications on Pages Two and Three before the Second Pass.

After selecting your output device(s) on Page Four, press ENTER and DISkASSEMBLER will begin disassembling the code you have specified. During disassembly, you can use any of the FCTN keys previously defined.

Selecting one of the function keys (options) during a pass will pause disassembly, except CTRL 1 - Colors, and FCTN = - Scroll Speed. Just press the space bar or press ENTER to continue. Pressing the Space Bar or ENTER will also pause the screen scrolling which can be restarted by pressing it again. With the scroll speed on SLOW you can press the Space Bar down and hold it and then tap the ENTER key to scroll the screen up one line at a time to position it for a Retro Screen Capture (FCTN 0).

The First Pass output contains six columns of information:

```
Address Source Code       Object Code    Text    RL   Addr
LN0070  LIMI >0002        >0300,>0002    '....'       0070
LN0074  LIMI >0000        >0300,>0000    '....'       0074
LN0078  MOVB *R13,R9      >D25D          '.]'         0078
```

The first and last columns are addresses; last column so you can tell what line you are on when toggling the screen (FCTN 5). The addresses in the first column are preceded by "LN." In this form, they are acceptable labels, and if your program is short, the First Pass can even be reassembled but it is not recommended since the symbol table will become full very rapidly. Columns two, three, and four contain the assembly language mnemonics and their operands (Source Code), the Hex values they represent (Object Code), and the ASCII representation of these values (Text) with a period replacing values that are out of the ASCII range. Column five contains a message (like RL1 or RL2) that tells you how many relocatable words of information are on that line when you disassemble an RORG Dis/Fix 80 file.

When you disassemble code that you have specified Data and Text blocks on Page Two the output will start with the LN followed by the word DATA (or TEXT if you used the T option) and then 6 words of hex data followed by 12 bytes of Text with the address at the end.

If you toggle the screen back and forth during a pass, you may notice that two characters in columns 39 and 40 of the printed output do not appear on the screen. This is because DISkASSEMBLER prints 78 column output to an output device while only 38 columns per side are displayed on the screen.

If you are working on a Dis/Fix 80 file, some important information about the code may appear at the end of the First Pass. DISkASSEMBLER will tell you if the code contains mixed RORGs and AORGs, or out of order origins. In these cases, you should make a new First Pass and select the X option for Extended Label Checking. If DISkASSEMBLER brings up a list of REFs at the end you should invoke the V option before you start the Second Pass. To make a new First Pass, hit ENTER at the end of the old First Pass and you will return to the "File Information" section of Page One. Then press FCTN 4 or FCTN 9 to escape from this area back to the Top of Page One. The filename you are working with will be displayed. Accept this by pressing ENTER and continue on for a new First Pass, don't forget to put in the X option if it was needed.

SECOND PASS

When DISkASSEMBLER finishes the First Pass, press ENTER and you will be returned to Page One. Select any new options you wish from the File Information block and continue through the pages. If you are planning on reassembling the code later, you should specify output to disk on Page Four before you press ENTER.

DISkASSEMBLER outputs six columns on this pass also. But, column one is now the label field.

```
Label   Source Code       Object Code     Text     RL   Addr
AM      LIMI >0002        >0300,>0002     '....'        0070
        LIMI >0000        >0300,>0000     '....'        0074
AB      MOVB *R13,R9      >D25D           '.]'          0078
AA      JLT  AG           >1105           '..'          007A
```

Labels are assigned using combinations of two letters in the alphabet or a letter and a number (e.g. "AB" or "A6). Addresses that are referenced in statements like JMP, and operands in MOV, CLR, etc. statements, can also be assigned labels. DISkASSEMBLER can create up to 936 labels.

DISkASSEMBLER will first generate labels AA through ZZ. Next it generates A0 through Z9. If DISkASSEMBLER finds that it will be generating labels R0 through R9 it will automatically turn off the R option for the Second Pass even thought it still appears in the "Options" list. If it was left on, the assembler would generate an error for the R0 through R9 labels it finds since these symbols are reserved for register designations. Without the R option the registers are designated simply as 0, 1, 2, ... 10, 11 etc. so the labels R0 through R9 are valid.

Besides adding labels, and the splitting up of DATA blocks for label insertion, the other significant difference in the results of the Second Pass are found at the end of the output. This information appears in groups. In some of these groups, the first character on a line is an asterisk (*). This is because these lines may or may not be required for reassembly, but can be included as comments in the reassembled code. Each group is discussed on the following pages in the order it appears. DISkASSEMBLER will place these groups into a separate file when your Second Pass is output to a disk, Ram disk or Hard disk. We call this last file the Equate File.

REF/DEF/END - If the file is not Dis/Fix 80 or if it does not contain any REFs or DEFs, this block will not appear.

```
*       SLAST  END    >C01A  (Absolute)
*       DEF    BOOT   >C01A  (Absolute)
*       REF    VWTR   >C0E0  (Absolute)
*       REF    VDPWA  >C16C  (Absolute)
```

SLAST END >xxxx will appear when the original code was an Auto Start file. The address designated by the >xxxx after END is the start address for the program. In the above example, you will also notice that the programmer DEF'd the label BOOT and that it also references the same start address. Following the SLAST END statement come the REFs and DEFs in alphabetical order. REFs are references to routines or addresses (pointers) that are loaded into memory either by the module or by another file. After the file is loaded by the Tagged Object code loader, the loader will go back through the code in memory and resolve the REFs. It does this by replacing the REF chain pointers (operand field) with the actual address for the routine or pointer that was REF'd. DEFs are a method of placing labels into the REF/DEF table for use as REFs for other files, start name(s) or CALL LINK names.

If there are references to routines or pointers, the word REF, name of the routine or pointer, and address of the last reference to that routine or pointer will appear in this block as well as the word absolute or relocatable. If the code contains REFs, and you plan on reassembling the code, you should output the Second Pass to your printer and to your output device. The printed output will help you, when editing the code, to resolve the REF chain pointers prior to reassembly of the code.

NOTE: All DEFs MUST BE an actual label, in the label field, somewhere in the source code in order for them to reference valid address. Also, REFs or DEFs with a >0000 address, were REF'd or DEF'd by the programmer but not actually used in the program. See Reassembly for additional information.

Following the REF/DEF group you may find some information pertaining to the AORGs and RORGs in the file. DISkASSEMBLER will tell you if the "File contains" - "Out of order origins" or "Absolute and relocatable data". In which case you should have used the X option on the First and Second Pass. If you are not disassembling a Dis/Fix 80 file, or if the file is all RORG these messages will not appear.

EQU LIST - Unlike any other disassembler, DISkASSEMBLER will automatically generate an EQU LIST for you. This is a list of all the labels created that weren't used as actual labels in the label field. These are operands (values) or addresses that have been assigned names. They are critical to the proper execution of the code, once it's reassembled, and are not preceded by asterisks. Without this EQU LIST, you would receive a number of "Undefined Symbol" errors when you reassemble the code.

SYMBOL TABLE LIST - In this group DISkASSEMBLER will generate a list of all the symbols that were used in the label field and the address (line numbers) where they reside. As you may have noticed, this group is preceded by an asterisk since, it was placed in the last file just for your reference. It is not needed for proper reassembly of the code. This table is followed by a line that tells you how many symbols (labels), in hex, that were generated by DISkASSEMBLER. As it was stated before, DISkASSEMBLER can generate up to 936 labels or symbols, which is >03A8 in hex. If this line states that there are >03A8 symbols or you see that label Z9 was used, there is a good chance that DISkASSEMBLER ran out of labels. This will only happen with LARGE files. By carefully specifying DATA or TEXT blocks on Page Two and/or leaving out the X option, you can reduce the number of labels used and, perhaps, DISkASSEMBLER won't run out.

*** DSKx.xxxxxx** - This is the last line that DISkASSEMBLER generates IF the Second Pass was output to a device (FCTN 8), other than a printer. It is placed on screen and in the last file to indicate the name of the Equate File for editing and adding COPY directives to. See Reassembly for additional information.

NOTE: The file name is also placed at the beginning of each file created and it is placed in the prinout to indicate where the code is split for easy editing.

THESE BLOCKS ARE THE KEY TO ACCURATELY REASSEMBLING THE CODE.

This part of the manual addresses reassembling code using the TI Editor/Assembler (E/A). You should have selected FCTN 8 on the Second Pass and assigned a filename like TESTA to the output file. We will begin by discussing reassembly of code that is not Program Image and does not contain any REFs. The following discussion is generic in that the same set of steps, with some additions, apply to reassembling the other types of code as well.

DIS/FIX 80 FILE WITHOUT REFs - If the file was Dis/Fix 80 and the code does not contain any REFs, follow these steps for reassembly.

1. Select 1 to EDIT from the * EDITOR/ASSEMBLER * screen and then select 1 to LOAD from the * EDITOR * screen.

2. Specify the FILE NAME of the last file created by DISkASSEMBLER, we will call this the EQUATE file for future reference. The name of the Equate file appeared at the end of the output for the Second Pass (something like TESTD). Catalog the disk if you have forgotten it.

3. Select 2 EDIT from the * EDITOR * screen. The Equate file contains the REF/DEF/END, EQU, and Symbol blocks. If the code contains DEFs, delete the leading asterisk from them so that they are not REM'd out.

4. The last line in this file will reference the file you are editing. Let's say, for illustrative purposes, that you find "* DSK1.TESTD." This means that DISkASSEMBLER has created files named TESTA, TESTB, TESTC, and TESTD. Provided, of course, that you specified TESTA as the output file (FCTN 8). Use the COPY directive, explained on page 229 of the E/A manual, and add enough "COPY" directives to include all the files that were generated, except the Equate file. Example:

```
* >003F SYMBOLS
* DSK1.TESTD
       COPY "DSK1.TESTA"
       COPY "DSK1.TESTB"
       COPY "DSK1.TESTC"
```

   You may want to designate the floppy name instead of a drive number so that reassembly will not be drive number dependent. i.e. COPY "DSK.SOURCE.TESTA" when the floppy containing the output files is named SOURCE.

5. Use FCTN 9 to escape to the * EDITOR * page and select 3 SAVE. Select "Y" for VAR 80 FORMAT and save the file with its original name (DSK1.TESTD for this example). If there isn't an SLAST END >xxxx statement or any DEFs in this file you are ready for reassembly! Go on to step 8.

6. If the DEF group contained an SLAST END >xxxx statement, the program was an auto start file. If you want it to be auto start after reassembly you will need to edit the last source code file, the one previous to the Equate file, which contains the END directive, DSK1.TESTC for this example. So load that into the editor and edit the last line.

   If the Equate file contains an SLAST END >xxxx and a DEF name >xxxx, where the >xxxx address are the same, then use this label name after the END directive. Example:

```
*       SLAST  END   >C01A  (Absolute)
*       DEF    BOOT  >C01A  (Absolute)
```

   For the above example you would change the END directive in this file to now read:

```
SLAST END BOOT
```

   If the file did not contain a DEF that had the same address as SLAST END >xxxx then place a new label there, like the name START.

```
SLAST END START
```

7. If there are any other DEFs in the Equate file note their addresses on your printout. If any of them are contained in the file now loaded into the editor then edit those lines. Pressing FCTN 5 twice in the E/A editor will display the addresses on the right hand edge of the screen. When you find the line press FCTN 5 again to get to the left hand edge, label field. Next type in the DEF name that corresponds with this address. If the line already contains a label press FCTN 8 to insert a new line above it and at the beginning of the new line type in the DEF name, followed by EQU $. Example:

```
BOOT   EQU  $
AA     LWPI AC
```

   This allows you to assign two labels to the same address. Do this for each DEF name listed in the Equate file. Don't forget to save the edited version of each file before you load the next one. After all the DEF names have been put into the file(s) you are ready for reassembly! That was easy.

8. After your file is saved use FCTN 9 to escape to the * EDITOR * screen and select 2 ASSEMBLE. Specify your EQUATE file name as the SOURCE file, DSK1.TESTD for this example. Specify a name for the object file. Don't forget to use the R option if it was used in DISkASSEMBLER on disassembly.

   The E/A will then reassemble your EQUATE file and include all the other files specified in your COPY directives. DSK1.TESTD will include TESTA, TESTB and TESTC for this example. The assembler will generate one object code file that contains all the pieces specified with the COPY directives. You can then use the LOAD AND RUN option from E/A to run the code. If the file was auto start it will start right up. If it starts on a start name it will use one of the DEF names. Or, it may be a file that is CALL LINKed to from Basic.

   NOTE: There are other methods that programmers use to auto start their programs without the use of END START type directives. One is to AORG >83C4, the ISR Hook, with the Start Address of the program. This starts the program running when the next interrupt occurs.

DIS/FIX 80 FILE WITH REFs - If you did not specify printed output on the Second Pass, then print out the files that DISkASSEMBLER generated. You'll need the printed output to help you make the proper changes. Mark the printed output as follows, then you can easily use the E/A EDITOR to make the proper changes to the files DISkASSEMBLER has created.

Each REF will be shown in the REF/DEF/END group with the address (line number) of the last line it was used in.
Example: **REF VMBW >C030 (Absolute)**
This is the start of REF Chain Pointers. Follow the steps below to resolve all of the REFs on your printout.

1. Go to the referenced address minus 2 (the REF points to the operand) and you will find an instruction such as BLWP @>xxxx, or BLWP @AA for subroutine REFs (i.e. VMBW, GPLLNK, DSRLNK etc.). For pointer REFs (i.e. GPLWS, SOUND, UTLTAB etc.) you may find any valid instruction such as, LWPI >xxxx or LWPI AA, or LI R1,>xxxx or LI R1,AA. Examples:

```
BLWP @AF        >0420,>0000
BLWP @AN        >0420,>C04E
LWPI AB         >02E0,>83E0
```

2. Replace the label generated by DISkASSEMBLER or the >xxxx with the REF name. Example:

   Change BLWP @AF
   To BLWP @VMBW

3. Next look at the Object Code column of the printout on this line. If you see >0000 as the operand, i.e. >0420,>0000, then you have reached the end of the REF Chain for this name. If there is an address there, i.e. >0420,>C04E, then go to that address-2 or line (>C04C for this example) and replace the label or >xxxx with the same REF name. As you noticed, we are moving (chaining) backwards through the program. This is a similar operation to what the loader does when the file is completely loaded into memory, to resolve the REFs.

4. Now that you have pencil whipped your printout, use the EDITOR to make the proper changes to each of the files. Then go back and perform the steps outlined for DIS/FIX 80 FILE WITHOUT REFs to handle any DEFs or Auto Start statements.

   NOTE: IF the file containing REFs that you are disassembling loads into High Memory Expansion between >A000 and >C800, you can let the loader resolve the REFs for you. But, the file MUST LOAD between >A000 and >C800 or part of it will get wiped out by DISkASSEMBLER when it is loaded and run.

   To do this, first load a NON-AUTO START Dis/Fix 80 file into memory with 3 LOAD AND RUN. Next load DISkASSEMBLER and disassemble memory >A000 to the end of your program. You see, once the file is loaded by the E/A loader all of the REFs have been resolved. This has the added bonus of allowing you to specify "Data Following Subroutines" on Page Three, and not having to worry about the V option. Disassembling memory will then have the same results as disassembling a file without REFs or DEFs. (If the file contained DEFs you can patch them in later, before reassembly).

   ALSO NOTE: The Extended Basic Tagged Object Code Loader (CALL LOAD) can not resolve REFs. All items that are REF'd must be, instead, placed into the EQU List and the asterisks left in front of the REFs in the Equate file. i.e. VMBW EQU >2024 - see Appendix D in this manual or pages 412 through 418 in the E/A manual for the proper Extended basic equates and additional information.

PROGRAM IMAGE - STANDARD LOADER FILES - Program Image files do not contain REFs or DEFs and they do not use any of the utilities that are loaded into Low Memory by the E/A Module for Dis/Fix 80 files. The 5 RUN PROGRAM loader auto starts these files so we do not have to worry about END START directives. They all start with the first instruction of the first file loaded, i.e. ASSM1, EDIT1, EDITA1, FORMA1, GAME etc.

1. Follow the first four steps in DIS/FIX 80 FILES WITHOUT REFs for adding the COPY directives to the Equate file.

2. To change the file back into a Program Image format you will need to add **DEF SFIRST,SLOAD,SLAST** to the beginning of the Equate file for the first Program loaded by the RUN PROGRAM loader. Next add the floowing two lines after the first AORG in the first file that DISkASSEMBLER generated.

   ```
   SFIRST  EQU $
   SLOAD   EQU $
   ```

   SLAST is automatically placed in front of the END directive for you, so you do not need to EQU it or place it there yourself.

   NOTE: If the Program Image file you disassembled is part of a multi-file program, i.e. ASSM1 & ASSM2 or EDITA1 & EDITA2, you will need to place the DEF SLAST in the Equate file for the last file loaded by the RUN PROGRAM loader not in the first file. For example; place DEF SLAST in the Equate file for ASSM2 or EDITA2 and leave it out of the Equate file for ASSM1 or EDITA1. The DEFs for SFIRST and SLOAD along with their EQU $ must still be in the first file, such as ASSM1 or EDITA1.

3. Reassemble the file(s). Next Load the file(s) with 3 LOAD AND RUN and then load the Save Utility found on the on the second E/A disk. See page 420 of the E/A manual for instructions on its use.

   NOTE: If the program that was disassembled resides in Low Memory Expansion (AORG >2000 - 3FFF) you MUST use the Mini Memory Module to load it and then load a modified Save Utility (see Appendix A of this manual). This is because the E/A Tagged Object Code Loader and the standard Save Utility both reside in Low Memory and as such either the loader will be wiped out by your program or your program will be wiped out by the Save Utility.

PROGRAM IMAGE - CUSTOM LOADER FILES - These Program Image file are loaded by a custom loader that was written by the programmer. You MUST disassemble and understand how this loader works and where it places the Program Image file in memory that it loads. Without knowing where in memory it places the file it loads, you can not disassemble or reassemble and run a Custom Loader file with much success.

This is especially important for files like FORTHSAVE that are placed all over memory when they are loaded. Since these files do not contain any Tags, DISkASSEMBLER has no way of knowing where they reside. It assumes that they reside in one contiguous area of memory, with a default address of >A000, and disassembles them as such. You will need to find out if this is true or not and add the proper AORG directives to the source code to get them to load right. Also, if they do reside all over memory you will need to write a special Save utility that groups them back into one block before they are Saved.

Even if they reside in one contiguqus area of memory you will still need to write your own Save utility to order to Save it as one file without the standard first 6 bytes of loader information in the beginning of the file.

MEMORY DISASSEMBLY - Usually when you disassembly memory, especially ROM or DSR ROMs, it is done strictly for reference and as such it is rarely reassembled. If it was reassembled you could not load it because ROM can not be written to. The exception to this is when you let the loader resolve the REFs for you in a Dis/Fix 80 file (See DIS/FIX 80 FILE WITH REFs), or if you have a device that contains RAM where ROM normally resides. In these cases, it is like reassembling an AORG'd Dis/Fix 80 file without REFs or DEFs.

GOOD LUCK, LEARN A LOT.

NOTES

APPENDIX A - TUTORIAL ON THE SAVE UTILITY

The purpose of this tutorial is twofold: 1) to guide the user through the actual process of using DISkASSEMBLER to break down and reassemble an existing file in a different way and 2) use the output to understand what the program is really doing. The file we are going to disassemble and convert into a relocatable file is called SAVE. It is on the second diskette that comes with the Editor/Assembler package.

Formatting the Output

Remember that it is always a good idea to scan the file once as text to see if there are obvious large blocks of text. This would be done as follows: after entering the name of the program (DSKx.SAVE) on Page One of the DISkASSEMBLER, you will see in the "File Information" that there are >0000 bytes of relocatable code, i.e. the file has an absolute origin, and that the origin is at >2800. Press Enter to place the cursor in the "Options" area and enter a T in your list of options so that text will come up on the left screen. Now on Page Two enter as a "block" of disassembled code >2800 for the start address and >FFFE as the end address. This should certainly take care of the whole range. Then use FCTN 6 to advance through the rest of the pages and press <ENTER> on Page Four to start the process. You will see at first large blocks of obvious text, since you can read them. As these blocks scroll by, look for periods since unprintable text is represented by a period. When you see one then check the data side of the screen to see if there is really a hex >2E there, in which case the period is real. (If it isn't then the program will not reassemble correctly as text.) In this particular case you will find that all the addresses from >2800 to >2BB2 contain only text with the exception of the words at >282C and >2B30. So we don't want these in our text block. It turns out that there are other short areas of data only, but we will ignore these for now.

You should also note that there are multiple external REFs listed at the end, so you should invoke the V option as well to avoid getting two word blocks of data where there should be opcode (see Options on page 11).

Having scanned through the entire program you should now see the message to press enter for the Second Pass. After you press enter the cursor will be in the "File Information" area, so press FCTN 9 to place it in the File Name area to redo the First Pass. Accept the default program name by pressing enter, and remember to add the V option. However, now on Page Two enter three separate blocks. The first should start at >2800 and end at >282C, the second start at >282E and end at >2B30, and the third start at >2B32 and end at >2BB2. Then press FCTN 6 to advance to Page Four and press enter, we don't need an output at this time.

When you get to the end of the First Pass press enter, you are now starting the Second Pass. Accept all defaults on Pages One through Three by pressing FCTN 6 for each page. On Page Four use FCTN 7 to output to your printer, and FCTN 8 to output to a disk file (for instructional purposes we will use the name SAVEA for the output file). At the end of the Second Pass DISkASSEMBLER is finished so press enter and then and you can exit the program by pressing CTRL = since the cursor is at the Top of Page One.

Modifying the Output

It is useful to have a printed output to make the following changes, before typing them in on the disk files, but not necessary.

First look at the file SAVEC (you should have seen the message at the end of the Second Pass that this was the final output file). As described in the section on Reassembly - Dis/Fix 80 With REFs, you will now have to trace the REF chain backwards, entering the proper name at each location, so that the file can be reassemble properly. First look at the last one listed - VSBW. The REF tells you that this is located at >2C40. Scan the reference addresses at the right and find 2C3E (line 150 of SAVEA).

```
       BLWP @>0000     >0420,>0000     '. ..'     2C3E
```
Replace the >0000 with VSBW so that it now reads:
```
       BLWP @VSBW      >0420,>0000     '. ..'     2C3E
```

You did not actually find 2C40 because it is the second word of the instruction. The >0000 tells you that there are no more references to VSBW.

Now do the same with the next REF - VSBR. This is at >2E10, so look for >2E0E which is at line 101 of SAVEB.

```
        BLWP @>0000     >0420,>0000     '. ..'     2E0E
```

Again, replace the >0000 with VSBR so that it now reads:

```
        BLWP @VSBR      >0420,>0000     '. ..'     2E0E
```

Once again there are no more (>0000). Next is VMBW at >2DA4. This is on line 72 of SAVEB.

```
        BLWP @BZ        >0420,>2D5A     '. -Z'     2DA2
```

Note that here you see BLWP @BZ. Replace the BZ with VMBW, and see that the data in the comment field tells you that BZ is >2D5A.

```
        BLWP @VMBW      >0420,>2D5A     '. -Z'     2DA2
```

So now look for >2D58 (line 49) and do the same thing.

```
        BLWP @>BP       >0420,>2D4E     '. -N'     2D58
```

so that it now reads:

```
        BLWP @>VMBW     >0420,>2D4E     '. -N'     2D58
```

In the case of VMBW there are TEN references and you must work each one backwards, replacing the label or address with VMBW. The last, indicated by BLWP @>0000 is on line 121 of SAVEA.

Now do the same with the other REFs. For your information, there are 2 for VMBR, and 1 each for SLOAD, SLAST, SFIRST, KSCAN, GPLWS, GPLLNK, and DSRLNK. GRMWA and GRMRD are never actually used since their REF address is >0000. Also, the label SAVE should be placed in the label field on the line for address >2BD6 (line 118 of SAVEA), since this is the DEF, or starting, address. If you are typing in the changes directly on the disk files already, remember to save them each time you go to a different file, or you will lose what you have done! That is why it is easier to pencil in the changes on paper, and then type them all at once.

Once you have the REFs and DEFs resolved and typed in you are ready to make the final changes in the disk file. Load SAVEA and decide whether you want an AORG (Absolute Origin) or RORG (Relocatable Origin) file. If it is to be an AORG file then you must change the addresses listed with each AORG on the file (lines 10, 85, and 87 of SAVEA) remembering to keep the relationship between them the same. For example, if you wanted to AORG the file at >D000 you would change the three AORGs to:

```
               AORG >D000
               AORG >D332    (>2B32 - >2800 + >D000)
               AORG >D352    (>2B52 - >2800 + >D000)
```

In this tutorial we will make it a relocatable file, by deleting the first AORG altogether. The other two may also be deleted, and then on the same line as the label AW, type BSS >20 (this is the difference between the two AORGs just deleted (>2B52 - >2B32 = >0020) and represents the area reserved for the name of the program to be created, as you will see below). While we are at it, you will see when we analyze the program that the following sequences are data only, so to make things look neat, type DATA over the opcode and delete the operand field so that the words of DATA which are presently a comment, slide over and become the operand: SAVEA line 16 (>282C), 100-101 (>2BB4->2BB6), 103 (>2BBA), 109-112 (>2BC4->2BCA), and 116 (>2BD2). There aren't any in SAVEB and SAVEC. So these lines in SAVEA should now look like:

```
0016        DATA >2E00     '..'      282C
0100        DATA >0600     '..'      2BB4
0101        DATA >1000     '..'      2BB6
0103        DATA >2000     ' .'      2BBA
0109        DATA >B70F     '..'
0110        DATA >800F     '..'      2BC6
0111        DATA >D000     '..'      2BC8
0112        DATA >B70F     '..'      2BCA
0116        DATA >7C7C     '||'      2BD2
```

Next load SAVEC, after saving SAVEA and at the the bottom of this file delete the asterisks at the beginning of each line of the REFs and DEFs. Next append to the end of this file the COPY directives for the other two files:

```
COPY "DSKx.SAVEA"
COPY "DSKx.SAVEB"
```

where x is the disk drive you are using. Save the file as DSKx.SAVEC and you are now ready for the final assembly process. Enter the ASSEMBLER in the E/A module. For the input file type DSKx.SAVEC and for the output file, use any name you wish (but not SAVE if the original file is on this disk). For options use R, since your source code uses R for registers. If you had not used the R option in the DISkASSEMBLER then it would not be necessary here.

Before you can actually run this program, read the instructions in the E/A manual. You need to have another program in memory which has as DEFs the labels SFIRST, SLOAD, and SLAST, referring to the first and last (END) addresses of the file, and normally the first instruction of the file should be B @LABEL, where LABEL is the actual start of your program. Since this file is relocatable it can't be used with AORG files unless its loaded into the Mini Memory Module. It should also be loaded last, so that the code saved begins where the module normally puts it.

What does SAVE do?

First we look at the DEF for the program which is SAVE. This is >2BD6 located on line 118 of SAVEA (you should have put it there already). The first 4 lines set R0 to >0000, which is the beginning of the screen image table, R1 to AB (line 17 of SAVEA) and R2 to >300, then does a BLWP at VMBW. Remember that in a VMBW or VMBR, R0 must always be the location in VDP that data is to be moved to or from, R1 is the location in CPU that it will be moved from or to, and R2 is the number of bytes to move. A complete screen in GRAPHICS mode is 24 rows by 32 columns, or 768 bytes (>300 in hex), so in other words a complete screen is written to VDP ram. This is a wasteful way of doing it, by the way, since the spaces don't need to be written. Next a VMBW is done for the 8 bytes at AC to VDP >400. This is the definition of sprite character >80, and if you look at the code (0000000000007C7C) it represents an underline. See the Basic Manual for explanation of graphics definitions. This program will be using a sprite as its cursor.

Next the sequence moves the word at AE (>B70F) to AF. You might wonder why, since AF already contains this word, but as you will see, it changes during the course of the program, and this instruction reinitializes it. Next R3 is loaded with >02E2, which will be used for the starting location for input on the screen (row 23, col 2). Finally a BL @AG is done, which uses the 8 bytes at AF to set the sprite attribute list for sprite >ǿ1 at VDP >300. It initially sets the pixel row to B7, pixel column to >0F, uses character >80, and color >F (white). The routine AG will always be used to move the sprite, because the 2nd byte at AF will be changed.

Now a BLWP @KSCAN is done, to look for a key press. MOVB @>837C looks for the condition bit at the GPL status bye; it is 0 if no new key was pressed. Hence the JNE AI tells the program that a new key WAS pressed, and JMP AJ goes back to look again, if a key WAS NOT pressed. (This sequence is looped through thousands of times per second).

We are now at line 137 of SAVEA. If a key was pressed, the byte at >8375 contains the ASCII value of the key, and it is moved to R1 and checked for the following values: A) >08 is the backspace - we jump to AM which checks to make sure that R3, which is keeping track of the screen location, is not less than >02E2. If it isn't, R3 is DECremented, 8 is subtracted from the value at AR, which is the Y location of the sprite (1 byte after AF) and we branch back to AJ, move the sprite (BL @AG) and then look for a key press. B) >0D is the enter key and will jump to the continuation of the program. C) >05 is the quit key and jumps to AO. This branches to AK, which clears the status byte so no error is detected, loads the GPL workspace and branches directly to >0070 (which in turn will return to the calling program). AK could not be branched to directly because it was more than >100 bytes away, and jumps can't handle this. D) >0F is the back key and does the same as quit in this case. E) >20 checks for any other ASCII less than the space key and doesn't allow it. Finally by moving the current screen location in R3 to R0 and BLWP to VSBW, the key pressed is moved to the current screen location, the latter is incremented, checked against the end of the line and if not, we go back to look for another key. Unfortunately a mistake in the program does not give you an error if you ARE at the end of the line - it just puts you back at the beginning and allows you to keep typing!

Now let us say that the whole file name has been entered and displayed on the screen, and the enter key pressed. We therefore arrive at AN (line 171 of SAVEA). First the byte >D0 is moved from AV to AF and the 8 from AF to VDP >300 via routine AG. The byte >D0 indicates no more active sprites and since it is at the start of the sprite attribute list, we are left with NO active sprites. This speeds up the rest of the program. Next the >20 bytes on the screen beginning with >02E2 are moved to CPU ram at AW via a BLWP @VSBR. It should be >1E bytes since we began in col 2!

The next sequence at AZ "parses" for the first space in the name at AW. R3 is initialized to AW. AZ begins by making sure that the end of the block reserved for the name as not been reached. If not it looks for the value of the byte contained in the address in R3 and increases R3. It is compared to the value at AY which is a space. If it is not a space, we go back to AZ and look again, R3 being 1 higher than before. If it is, we move on. R3 now contains a value TWO higher than the last non-space in the name. It is moved to R8, which is then decremented by two, and hence holds the last non-space of the name. The value AW (not the byte contained AT that location) is now subtracted from R3, which therefore represents the length of the name + 1. This is moved to R4 to save it, and then decremented so that R3 now holds the name length. The length is now compared to 3 and if less we branch to AX which uses the text at AS (bad file name) for an error message, eventually allowing you to start again. Follow this sequence in your source code - you should wind up at CA which is just after the start of the program.

If the name length is OK, the bytes in R3 are swapped so that the length is in the MSB, and this is moved (byte operations are always on the MSB, hence the swap) to location BA, which is 1 before AW. That is we now have at BA the length of the name followed by the actual name.

Now we are finally ready to work with the program that was loaded to be "saved," and happen to be exactly at the last line of SAVEA. R6 is loaded with SLOAD, the start address of the program to be saved, and is stored at location BB, and R7 with SLAST the end address. SFIRST is subtracted from R7 which is therefore the total length of the program. If it is negative (Jump if Less Than) an error has occurred, the message at BX is loaded into the error routine and the latter is branched to. Next at BW R7 (which will always contain what is left to save) is moved to R2 and compared to >1FFA. This is because >2000 (8K) bytes can be saved in each portion of the program, but the first 3 words are reserved for system information (the flags >FFFF or >0000 for more to load or no more to load, the total length in the segment, and the start address to put it at). Thus if R2 is less than >1FFA all of it will be used, otherwise a maximum of >1FFA.

Now at BD, R2 is moved to R10 to save it and then 6 is added to account for the 3 words mentioned above. This total length of bytes in the file is now moved to location BE, which is the 4th of a sequence of 5 words starting at AF. This will comprise all but the name of the file in a PAB, which must be set up before any disk access. You should read your E/A manual for a fuller explanation of PABs. >06 represents the OP-CODE for save (a memory image file) >1000 the location in VDP of the data to be saved, the third word is not used, the 4th (taken from R2) represents the number of bytes to be saved) and the LSB of the last is the length of the device name, which will be inserted in a moment. The first 9 bytes (up to the length) are moved to VDP at >0F80, a frequent but not necessary location for PABs. Now R4, which you may remember from above is the device name length plus 1, is moved to R2, and the total sequence of length plus name itself is moved to VDP at >0F89, the 10th byte of the PAB. This completes the setting up of the PAB.

The screen is now rewritten, wiping out the input device name, and R2 bytes (DECremented by 1) from AW are written to the screen on the line below "CURRENT FILE :" You can check this by seeing what it in this file at >202 + >282E = >2A30. Next location BK is cleared and then either left as 0 if R10 is equal to R7, or set as >FFFF if they are unequal.

(Remember that at BW, R7 was what was left to do in tl
entire program, and at BD, R10 was the number of bytes i
this segment - >1FFA or what was left). At BL the length c
this segment (including the extra 6 bytes) is moved to I
and SFIRST (where it is to start when the newly create
program is loaded into memory) is moved to BN. These si
bytes at BK,BM, and BN are moved to VDP at >1000 to form th
beginning of the data sequence to be saved. The moving c
data is completed by loading R0 with >1006, the location i
VDP after the 3 data words just moved, R1 with SFIRST (R6
the location of the beginning of the program in CPU, and I
with the number of bytes in R10 (>1FFA or less). Then BLW
@VMBW does the job.

The name at AW is now compared to "CS" at BQ and if equal
i.e. cassette is to be used, a branch to the routine at E
is peformed. We won't analyze this part now, but you ca
used it if you ever want to set up cassette access routine
in your programs. It involves setting up a lot of words i
scratch pad ram (>8300 to >8400) and then BLWP @GPLLNK, DAT
>3D.

If the cassette routine is not used, then the main progra
goes on. The location of the device name length (>0F89 i
this case) is moved to location >8356 in CPU (this must b
done before EVERY disk access) and then a BLWP @DSRLNK, DAT
>08 completes the process of writing this segment to disk
The equal bit of the status register is set if there is a
error, so JEQ BT branches to the error routine and will giv
an I/O error message, with the number of the error bein
determined from byte 1 of the PAB (this is done by th
console and the DSRLNK routine) and employed in the sequenc
at BU. For some strange reason the routine BU is branche
to anyway, but since the error byte is 0 it jumps ou
immediately with a JEQ CG, where CG contains B *R11, o
return.

We now move the "more" flag to R1 and if it is 0, i.e. n
more, go to the return to E/A routine at AK. This is th
same routine used if quit or back was pressed at th
beginning of the program. Next we add R10, which should b
>1FFA if there was more to do, to R6, which is SFIRST, s
that now R6 holds the start address for the next segment
used in the 8th line after BL (line 48 of SAVEB). R10 i
also subtracted from R7, leaving R7 with the total of wha
is left, and added to the word at BB, to hold the new SLOAD
used 1 line after BL. Finally we AB @BV,*R8. That is, the
byte at BV, which is >01, is added to the contents of the
location specified in R8, which you may remember from way

back, was the last character of the device name. So we have now set up a new filename differing by one ASCII character and we finally branch back to BW which will determine how much is left to do, and do it.

This tutorial has of necessity been rather long. However it shows you the power of DISkASSEMBLER, and how you can use it to take apart a program and LEARN!

APPENDIX B- PROGRAM FILES LARGER THAN 48 SECTORS

As noted in Section 2.2, program files larger than 47 sectors (48 when listed in the catalog) cannot be handled by DISkASSEMBLER, because of lack of buffer space. HERE ARE THE INSTRUCTIONS TO SPLIT such FILES up into manageable segments that can be disassembled. The only requirement is a sector editor such as Advanced Diagnostics (A/D). We will use A/D for this example.

First, copy the file to a blank initialized disk. This will make the following steps easier, because you will know where the file is located, and we can refer to absolute sector numbers. Next copy two dummy files to the same disk. Use names that follow the program name you are working with alphabetically. If the program name was TEST, use TESTA, and TESTB. Now enter Advanced Diagnostics, select the drive your working disk is in, and Edit Sector 1 (ES 1). Sector one is the alphabetical list of header sectors (file descriptor records or FDR) and should read 000200030004 and the rest 0's. We will leave this sector alone. Next ES 2. If you read it in ASCII you should see the name of your program - we will work with the hex code in a moment. Now go back to the command line and Write Sector 3 (WS 3) and WS 4. You have now replaced the header sectors of the two dummy files with exact copies of the one for your program. We will now modify sectors 3 and 4.

FIRST SPLIT - First let's edit sector 3 (ES 3). In ASCII mode, change the name of the file to the first name you selected above (TESTA). Next look at byte 15. This is the hex equivalent of the number of sectors used, which is ONE LESS than that listed in a catalog (the latter counts the FDR as well). For this file, we will use the maximum that DISkASSEMBLER allows (>2F or decimal 47 sectors). Type this in. Next move down to bytes 28-30. This is the beginning of the series of data chain pointer blocks telling the disk controller where the sectors are located. Because we have moved the file to a new disk there is only one. We will use, as an example, a file that has 50 sectors of data (listed as 51 in a catalog). The block will thus read 221003 (or 201003 if a Myarc Disk Manager was used with another sector editor).

These 3 bytes are composed of 6 hex digits, each called a "nibble." We will call them n1 to n6. n4n1n2 in that order indicate the start sector of the first contiguous block of data sectors (in this case >022, decimal 34). n5n6n3 indicate the offset from the beginning of the block to the end, and includes offsets in previous blocks, in this case none. For this file we have >031, so the last sector in the block is >022 + >031, or >053. These 3 nibbles are the ones we must change. We want to indicate that only >2F sectors are used. The offset must be >02E, since it is ADDED to the first sector. Type "02E" over n5, n6, and n1. The block should now read >22E002. That completes the change for this FDR. Get back to the command line and save this back to disk using Write Sector 3 (WS 3).

SECOND SPLIT - First, ES 4. Change the name of the file to the second name you selected when creating the dummy files. Now go to hex. In byte 15 type 03. This number is the number of sectors left (50-47). You must type whatever is appropriate for your file of course. Now move down to bytes 28-30. We want the "file" to start where the previous one left off, so we add >2F sectors to the original start sector, >22. This will be sector >051 so type in "051" over n4n1n2. The block should now read 51x0xx. The offset now becomes >002 (calculated by subtracting >02F the number of sectors in the first file, from >051, the original offset). Type this in and you now have 512000. Write this sector back to disk (WS 4) and you are finished.

If you catalog the disk now, you will see something like:

Filename	Size	Type
TEST	51	PROGRAM
TESTA	48	PROGRAM
TESTB	4	PROGRAM

TESTA and TESTB can be read and disassembled by DISkASSEMBLER. The same technique can be used for files of any size, of course, even ones that have to be split into three parts.

NOTE: Since no file larger than 33 sectors is meant to be loaded by Option 5 of the E/A or Option 3 of the TI-WRITER, these files must have a separate loader. You should disassemble this loader first to see how the file is placed into memory, and WHERE IT STARTS. This is necessary so that you may enter the start address, "Loads At", on Page One of the DISkASSEMBLER input screens, otherwise the code will not make sense.

ALSO NOTE: For a more complete explanation of FDR's and the Data Chain Pointer Blocks, see page 33 of the Advanced Diagnostics Manual.

Extended Basic LOW MEMORY EXPANSION after CALL INIT

>2000	>205A	XML link to name link routine pointer.
>2002	>24FA	First Free address in low mem-exp.
>2004	>4000	Last Free address in low mem-exp.
>2006	>AA55	Indicates CALL INIT has been executed.
	UTILITY	**VECTOR TABLE** (ie: BLWP @KSCAN)
>2008	>2038	**NUMASG** Utility workspace pointer
>200A	>2096	Start address for BLWP @NUMASG
>200C	>2038	**NUMREF** Utility workspace pointer
>200E	>217E	Start address for BLWP @NUMREF
>2010	>2038	**STRASG** Utility workspace pointer
>2012	>21E2	Start address for BLWP @STRASG
>2014	>2038	**STRREF** Utility workspace pointer
>2016	>234C	Start address for BLWP @STRREF
>2018	>2038	**XMLLNK** Utility workspace pointer
>201A	>2432	Start address for BLWP @XMLLNK
>201C	>2038	**KSCAN** Utility workspace pointer
>201E	>246E	Start address for BLWP @KSCAN
>2020	>2038	**VSBW** Utility workspace pointer
>2022	>2484	Start address for BLWP @VSBW
>2024	>2038	**VMBW** Utility workspace pointer
>2026	>2490	Start address for BLWP @VMBW
>2028	>2038	**VSBR** Utility workspace pointer
>202A	>249E	Start address for BLWP @VSBR
>202C	>2038	**VMBR** Utility workspace pointer
>202E	>24AA	Start address for BLWP @VMBR
>2030	>2038	**VWTR** Utility workspace pointer
>2032	>24B8	Start address for BLWP @VWTR
>2034	>2038	**ERR** Utility workspace pointer
>2036	>2090	Start address for BLWP @ERR
>2038	>0000	Start of Utility Workspace
>205A	Start of XML link to name link routine.	
>2090	Start of ERR Routine. (Return Error code to basic)	
>2096	Start of NUMASG Routine. (Numeric Assignment)	
>217E	Start of NUMREF Routine. (Numeric Reference)	
>21E2	Start of STRASG Routine. (String Assignment)	
>234C	Start of STRREF Routine. (String Reference)	
>2432	Start of XMLLNK Routine. (Link to sys Utilities)	
>246E	Start of KSCAN Routine. (Keyboard Scan)	
>2484	Start of VSBW Routine. (VDP single byte write)	
>2490	Start of VMBW Routine. (VDP multiple byte write)	
>249E	Start of VSBR Routine. (VDP single byte read)	
>24AA	Start of VMBR Routine. (VDP multiple byte read)	
>24B8	Start of VWTR Routine. (Write to VDP register)	
	(NOTE: No GPLLNK or DSRLNK in X-Basic CALL INIT)	
>24FA	**First Free Address** Low Mem-Exp pointed to by >2002	

Extended Basic LOW MEMORY EXPANSION Continued

	The REF/DEF Table resides at the end of
	Low Memory Expansion. Each entry is 8 bytes.
	6 for the Name and 2 for the starting address.
	CALL INIT in X-Basic leaves this space empty.
>3FF0	DEF Name (CALL LINK or BLWP @) 6 characters.
>3FF6	Start address of the above routine, 2 bytes.
>3FF8	DEF Name (CALL LINK or BLWP @) 6 characters.
>3FFE	Start address of the above routine, 2 bytes.

Extended Basic HIGH MEMORY EXPANSION usage

>A000	**START OF HIGH MEM-EXPANSION**
	(If Mem-Exp is present then the value at >8389
	will be >E7 while the program is running)
	NUMERIC VALUE TABLE (in RADIX 100 notation)
	Starting point of the Symbol table in VDP RAM is
	pointed to by >833E while the program is running.
	The Symbol table then points into the Numeric
	value table for each of the variable names.
	Highest Free Address Mem-Exp. pointed to by >8386
	LINE NUMBER TABLE - 4 Bytes per entry.
	| Line ¢=2 Bytes | Start Address of line=2 bytes |
	Line numbers are stored highest ¢ to lowest ¢
	Starting address of this table pointed to by >8330
	Ending address of this table pointed to by >8332
	Current line number being referenced
	in this table is pointed to by >832E
	PROGRAM SPACE (Last line entered is at the top)
	Start of program space = (value at >8332)+1
	Reserved words have been converted to Token values
	and line numbers are removed from the beginning of
	each line. The format for each line is as follows:
	1st Byte = Number of bytes for the line
	Following Bytes = Actual line code with Token
	values replacing reserved words.
	Last byte = >00
>FFE7	Highest address to be used in Mem-Exp by XB
>FFFC	Workspace Pointer for LOAD Interrupt
>FFFE	Start Address (PC) for LOAD Interrupt

Editor Assembler LOW MEMORY EXPANSION after CALL INIT

>2000	>A55A	Indicates CALL INIT has been executed.
>2002	>2128	Start address of NAME LINK routine
>2004	>2398	Start address of LOADER executed from GPL
>2006	>225A	Start address of CIF
>2008		Start of Variable Storage area
	UTLTAB	(Utility Table Area)
>2022	>0000	START address for program just loaded
>2024	>A000	First Free address in High Memory
>2026	>FFD7	Last Free address in High Memory
>2028	>2676	First Free address in Low Memory
>202A	>3F38	Last Free address in Low Memory and pointer to default REFs/DEFs thru >3FFF
>202C	>0000	Saved Checksum
>202E	>0000	Saved Pointer to FLAG byte in PAB (in VDP)
>2030	>0000	Saved GPL return address
>2032	>0000	Saved CRU base of Peripheral
>2034	>0000	Saved Entry address of DSR
>2036	>0000	Saved Device Name Length
>2038	>0000	Saved Pointer to Device Name (PAB in VDP)
>203A	>0000	Saved Version Number of DSR
>203C		Start of 80 BYTE RECORD Buffer
>2094		Start of UTILITY Workspace Registers
>209A		Start of DSRLNK Workspace Registers
>20BA		Start of USER Workspace Registers
>20D9		Start of LOADER Workspace Registers
>20FA	>0064	Data 100
>20FC	>2000	Data >2000 (H20 and H2000)
>20FE	>2E	Byte Decimal Point '.'
>20FF	>AA	Byte >AA for Validation
	UTILITY	**VECTOR TABLE** (ie: BLWP @KSCAN)
>2100	>2094	**GPLLNK** Utility workspace pointer
>2102	>21C4	Start address for BLWP @GPLLNK
>2104	>2094	**XMLLNK** Utility workspace pointer
>2106	>2196	Start address for BLWP @XMLLNK
>2108	>2094	**KSCAN** Utility workspace pointer
>210A	>21DE	Start address for BLWP @KSCAN
>210C	>2094	**VSBW** Utility workspace pointer
>210E	>21FA	Start address for BLWP @VSBW
>2110	>2094	**VMBW** Utility workspace pointer
>2112	>2200	Start address for BLWP @VMBW
>2114	>2094	**VSBR** Utility workspace pointer
>2116	>220E	Start address for BLWP @VSBR
>2118	>2094	**VMBR** Utility workspace pointer
>211A	>221A	Start address for BLWP @VMBR
>211C	>2094	**VWTR** Utility workspace pointer
>211E	>2228	Start address for BLWP @VWTR

Editor Assembler LOW MEMORY EXPANSION Continued

>2120	>209A **DSRLNK** Utility workspace pointer
>2122	>22B2 Start address for BLWP @DSRLNK
>2124	>20DA **LOADER** Utility workspace pointer
>2126	>23BA Start address for BLWP @LOADER
>2128	Start of Name Link routine.
>218A	Routine to Return to Assembly Language from GPLLNK
>2196	Start of XMLLNK Routine. (Link to sys Utilities)
>21C4	Start of GPLLNK Routine. (Link to GPL Routines)
>21DE	Start of KSCAN Routine. (Keyboard Scan)
>21F4	Start of VSBW Routine. (VDP single byte write)
>2200	Start of VMBW Routine. (VDP multiple byte write)
>220E	Start of VSBR Routine. (VDP single byte read)
>221A	Start of VMBR Routine. (VDP multiple byte read)
>2228	Start of VWTR Routine. (Write to VDP register)
>225A	Start of CIF Routine.
>22B2	Start of DSRLNK Routine. (Link to DSR routines)
>2398	Start of LOADER when comming from GPL
>23BA	Start of LOADER Routine. (Loads DIS/FIX 80 files)
>2676	**First Free address** Low Memory Pointed to by >2028)
>3F38	**Last Free address** Low Memory Pointed to by >202A)
	Start of Default E/A REF Table.

>3F38	UTLTAB	>2022	Pointer to Utility Table - Low Mem
>3F40	PAD	>8300	Start address of Scratch Pad Ram
>3F48	GPLWS	>83E0	GPL Workspace pointer
>3F50	SOUND	>8400	Location of the Sound Chip
>3F58	VDPRD	>8800	VDP Read Byte port
>3F60	VDPSTA	>8802	VDP Read Status port
>3F68	VDPWD	>8C00	VDP Write Byte port
>3F70	VDPWA	>8C02	VDP Write (set) Address port
>3F78	SPCHRD	>9000	Speech Read port
>3F80	SPCHWT	>9400	Speech Write port
>3F88	GRMRD	>9800	Grom/Gram Read Byte port
>3F90	GRMRA	>9802	Grom/Gram Read Address port
>3F98	GRMWD	>9C00	Grom/Gram Write Byte port
>3FA0	GRMWA	>9C02	Grom/Gram Write (set) Address port
>3FA8	SCAN	>000E	BL address for key scan routine
>3FB0	XMLLNK	>2104	BLWP address XMLLNK Routine
>3FB8	KSCAN	>2108	BLWP address keyboard scan
>3FC0	VSBW	>210C	BLWP address VDP Single Byte Write
>3FC8	VMBW	>2110	BLWP address VDP Multi Byte Write
>3FD0	VSBR	>2114	BLWP address VDP Single Byte Read
>3FD8	VMBR	>2118	BLWP address VDP Multi Byte Read
>3FE0	VWTR	>211C	BLWP address VDP Write To VDP Regs
>3FE8	DSRLNK	>2120	BLWP address DSRLNK Routine
>3FF0	LOADER	>2124	BLWP address DIS/FIX 80 Loader
>3FF8	GPLLNK	>2100	BLWP address GPLLNK Routine

Editor Assembler HIGH MEMORY EXPANSION

```
+-------------------------------------------------------------+
|>A000 |START OF HIGH MEM-EXPANSION                           |
|      |                                                      |
|>A000 | First Free address High Mem  Pointed to by >2024     |
|>FFD7 | Last  Free address High Mem  Pointed to by >2026     |
|      |                                                      |
|>FFD8 | XOP 1 Workspace                                      |
|>FFF8 | XOP 1 First Instruction                              |
|      |                                                      |
|>FFFC | Workspace Pointer  for LOAD Interrupt                |
|>FFFE | Start Address (PC) for LOAD Interrupt                |
|>FFFF |END OF HIGH MEMORY EXPANSION                          |
+-------------------------------------------------------------+
```

NOTE: When RORG (Relocatable Orign) DIS/FIX 80 files are loaded by the E/A Tagged Object Code Loader they are loaded at >A000 by default. The loader looks at the value in >2024, which contains >A000 at first, and loads the next word(s) according to this value.

If the file contains any AORG directives the loader will load the code where the programmer specified (i.e. AORG >C000) instead of where the pointer says to. Also, whith AORG code this pointer at >2024 is not updated as the file is loaded.

Since the E/A Tagged Object Coode Loader resides in Low Memory Expansion, >23BA - >2675, you can not use this loader to load this area of memory since the loader will get wiped out. Also, since this loader uses other routines in Low Mem and other address for storage, it should not be used to load >2000 - >23B9 either. However, you can use the Mini Mem's loader to load this area of memory since it's loader resides in the cartridge ROM and leaves ALL of Low and High Mem free for your programs.

If the file is a PROGRAM IMAGE type file the Program Loader built into the E/A module (5 RUN PROGRAM FILE) loads the file back into the memory it was SAVED from with the SAVE Utility. This loader is written in GPL code and resides entirely in the E/A module so it also leaves ALL of Low and High Mem free for your programs.

Mini Memory ROM >6000->6FFF

>6000	>AA00	Space for Standard Rom Header - All >0000
>6010	>605A	Start address of NAME LINK routine
>6012	>62CA	Start address of DIS/FIX 80 LOADER from GPL
>6014	>618C	Start address of CIF
>6016	>0000	not used
	UTILITY	**VECTOR TABLE** (ie: BLWP @KSCAN)
>6018	>7092	**GPLLNK** Utility workspace pointer
>601A	>60F6	Start address for BLWP @GPLLNK
>601C	>7092	**XMLLNK** Utility workspace pointer
>601E	>60C8	Start address for BLWP @XMLLNK
>6020	>7092	**KSCAN** Utility workspace pointer
>6022	>6110	Start address for BLWP @KSCAN
>6024	>7092	**VSBW** Utility workspace pointer
>6026	>6126	Start address for BLWP @VSBW
>6028	>7092	**VMBW** Utility workspace pointer
>602A	>6132	Start address for BLWP @VMBW
>602C	>7092	**VSBR** Utility workspace pointer
>602E	>6140	Start address for BLWP @VSBR
>6030	>7092	**VMBR** Utility workspace pointer
>6032	>614C	Start address for BLWP @VMBR
>6034	>7092	**VWTR** Utility workspace pointer
>6036	>615A	Start address for BLWP @VWTR
>6038	>7098	**DSRLNK** Utility workspace pointer
>603A	>61E4	Start address for BLWP @DSRLNK
>603C	>70D8	**LOADER** Utility workspace pointer
>603E	>62EC	Start address for BLWP @LOADER
>6040	>70F8	**NUMASG** Utility workspace pointer
>6042	>660E	Start address for BLWP @NUMASG
>6044	>70F8	**NUMREF** Utility workspace pointer
>6046	>66FE	Start address for BLWP @NUMREF
>6048	>70F8	**STRASG** Utility workspace pointer
>604A	>6768	Start address for BLWP @STRASG
>604C	>70F8	**STRREF** Utility workspace pointer
>604E	>6888	Start address for BLWP @STRREF
>6050	>70F8	**ERR** Utility workspace pointer
>6052	>6966	Start address for BLWP @ERR
>6054	>0064	Data 100
>6056	>2000	Data >2000 (H20 and H2000)
>6058	>2E	Byte Decimal Point '.'
>6059	>00	Byte >00
>605A		Start of Name Link routine.
>60BC		Routine to Return to Assembly Language from GPLLNK
>60C8		Start of XMLLNK Routine. (Link to sys Utilities)
>60F6		Start of GPLLNK Routine. (Link to GPL Routines)
>6110		Start of KSCAN Routine. (Keyboard Scan)
>6126		Start of VSBW Routine. (VDP single byte write)

Mini Memory ROM >6000->6FFF Continued

>6132	Start of VMBW Routine. (VDP multiple byte write)
>6140	Start of VSBR Routine. (VDP single byte read)
>614C	Start of VMBR Routine. (VDP multiple byte read)
>615A	Start of VWTR Routine. (Write to VDP register)
>618C	Start of CIF Routine. (Integer to Floating)
>61E4	Start of DSRLNK Routine. (Link to DSR routines)
>62CA	Start of DIS/FIX 80 LOADER when comming from GPL
>62EC	Start of LOADER Routine. (Loads DIS/FIX 80 files)
>660E	Start of NUMASG Routine. (Basic Numeric Assignment
>66FE	Start of NUMREF Routine. (Basic Numeric Reference)
>6768	Start of STRASG Routine. (Basic String Assignment)
>6888	Start of STRREF Routine. (Basic String Reference)
>6966	Start of ERR Routine. (Basic Error Message)
>697E	Thru >6F0C Not Used All >0000
	Can Be used in the Gram Kracker
>6F0E	Start of Default Mini Memory REF Table
>6F0E	UTLTAB >7020 Pointer to Utility Table - Low Mem
>6F16	PAD >8300 Start address of Scratch Pad Ram
>6F1E	GPLWS >83E0 GPL Workspace pointer
>6F26	SOUND >8400 Location of the Sound Chip (port)
>6F2E	VDPRD >8800 VDP Read Byte port
>6F36	VDPSTA >8802 VDP Read Status port
>6F3E	VDPWD >8C00 VDP Write Byte port
>6F46	VDPWA >8C02 VDP Write (set) Address port
>6F4E	SPCHRD >9000 Speech Read port
>6F56	SPCHWT >9400 Speech Write port
>6F5E	GRMRD >9800 Grom/Gram Read Byte port
>6F66	GRMRA >9802 Grom/Gram Read Address port
>6F6E	GRMWD >9C00 Grom/Gram Write Byte port
>6F76	GRMWA >9C02 Grom/Gram Write (set) Address port
>6F7E	SCAN >000E BL address for key scan routine
>6F86	XMLLNK >601C BLWP address XMLLNK Routine
>6F8E	KSCAN >6020 BLWP address Keyboard Scan
>6F96	VSBW >6024 BLWP address VDP Single Byte Write
>6F9E	VMBW >6028 BLWP address VDP Multi Byte Write
>6FA6	VSBR >602C BLWP address VDP Single Byte Read
>6FAE	VMBR >6030 BLWP address VDP Multi Byte Read
>6FB6	VWTR >6034 BLWP address VDP Write To VDP Regs
>6FBE	DSRLNK >6038 BLWP address DSRLNK Routine
>6FC6	LOADER >603C BLWP address DIS/FIX 80 LOADER
>6FCE	GPLLNK >6018 BLWP address GPLLNK Routine
>6FD6	NUMASG >6040 BLWP address Numeric Assignment
>6FDE	NUMREF >6044 BLWP address Numeric Reference
>6FE6	STRASG >6048 BLWP address String Assignment
>6FEE	STRREF >604C BLWP address String Reference
>6FF6	ERR >6050 BLWP address Error Message

Mini Memory RAM >7000 - >7FFF - After INIT

>7000	>A55A	Indicates that INIT MINI MEM has been done
>7002	>0000	Start of Identifiers passed by CALL LINKs
>701C	>7118	First Free address in Mini Memory Ram
>701E	>7FFF	Last Free address in Mini Memory Ram and pointer to user's REFs and DEFs thru >7FFF
	UTLTAB	
>7020	>0000	Default START address for program loaded
>7022	>A000	First Free address in High Memory
>7024	>FFE0	Last Free address in High Memory
>7026	>2000	First Free address in Low Memory
>7028	>3FFF	Last Free address in Low Memory
>702A	>0000	Saved Checksum
>702C	>0000	Saved Pointer to FLAG byte in PAB (in VDP)
>702E	>0000	Saved GPL return address
>7030	>0000	Saved CRU base of Peripheral
>7032	>0000	Saved Entry address of DSR
>7034	>0000	Saved Device Name Length
>7036	>0000	Saved Pointer to Device Name in PAB in VDP
>7038	>0000	Saved Version Number of DSR (i.e. >0001)
>703A		Start of 80 BYTE RECORD Buffer for LOADER
>708A		Start of DEVICE NAME Buffer
>7092		Start of UTILITY Workspace Registers
>7098		Start of DSRLNK Workspace Registers
>70B8		Start of USER Workspace Registers
>70D8		Start of LOADER Workspace Registers
>70F8		Start of Variable Storage area (temp data)
>7118	**First Free Address** in Mini Mem Pointed to by >701C	
>7FFF	**Last Free Address** in Mini Mem Pointed to by >701E Also Start of User REF/DEF Table	

NOTE: The Mini Memory Tagged Object Code Loader looks at the number of RELOCATABLE bytes (RORG) in the file and if they will fit in Mini Mem it loads it there. Otherwise it loads RORG (Relocatable Origin) Code into High Mem starting at >A000 by default.

If the file contains any AORG (Absolute Origin) Code the loader will load it where the programmer specified (i.e. AORG >2000). Also, since the Mini Mem Loader resides entirely in cartridge Rom and uses the Mini Mem Ram for temporary storage, it can load DIS/FIX 80 Tagged Object Code anywhere in Low or High Mem according to the RORG and AORG directives in the file.

Appendix D - Common Equates and Subroutines

Common CPU ROM/RAM Equates

>000E	SCAN	Scan Keyboard (BL @SCAN)
>006A	RESET	Return to GPL with COND Bit Reset (0)
>0070	NEXT	Return to GPL without changing COND Bit
>00CE	SET	Return to GPL with COND Bit Set (1)
>8300	PAD	Start of Scratch Pad Ram
>8322		Return Error Code from Assembly back to GPL
>834A	FAC	Floating Point and DSR useage
>8356	NAMLEN	Points to DSR Name Lenght Byte in PAB
>8374		Keyboard no. to be scanned
>8375		Key Code Returned by SCAN or KSCAN
>837C		GPL Status Byte
>83C2		Auto Sound, Auto Sprite and Quit Key flags
>83C4		ISR Hook - User Interrupt Start Address
>83F6		R11 of GPL Workspace
>83E0	GPLWS	GPL Workspace pointer
>8400	SOUND	Sound Chip port
>8800	VDPRD	VDP Read Byte port
>8802	VDPSTA	VDP Read Status Byte port
>8C00	VDPWD	VDP Write Byte port
>8C02	VDPWA	VDP Write Address and Registers port
>9000	SPCHRD	Speech Read Byte port
>9400	SPCHWD	Speech Write Byte port
>9800	GRMRD	Grom Read Byte port
>9802	GRMRA	Grom Read Address port
>9C00	GRMWD	Grom Write Byte port
>9C02	GRMWA	Grom Write Address port

Common VDP RAM Equates

>1000	Standard E/A PAB Location
>1080	Buffer for above PAB
>1100	Second E/A PAB
>1180	Buffer for second PAB
>1200	Third E/A PAB
>1280	Buffer for third PAB
>1300	Fourth E/A PAB (for COPY directives)
>1380	Buffer for fourth PAB and for PROGRAM files

Common Subroutine (BLWP) Addresses and Number of Data Words that follow them:

Name	E/A Add	XB Add	MM Add	No. Words
POWER UP	>0000	>0000	>0000	>0000
NUMASG	n/a	>2008	>6040	>0000
NUMREF	n/a	>200C	>6044	>0000
STRASG	n/a	>2010	>6048	>0000
STRREF	n/a	>2014	>604C	>0000
XMLLNK	>2104	>2018	>601C	>0001
KSCAN	>2108	>201C	>6020	>0000
VSBW	>210C	>2020	>6024	>0000
VMBW	>2110	>2024	>6028	>0000
VSBR	>2114	>2028	>602C	>0000
VMBR	>2118	>202C	>6030	>0000
VWTR	>211C	>2030	>6034	>0000
ERR	n/a	>2034	>6050	>0000
GPLLNK	>2100	n/a	>6018	>0001
DSRLNK	>2120	n/a	>6038	>0001
LOADER	>2124	n/a	>603C	>0000

NOTE: The E/A's NUMASG, NUMREF, STRASG, STRREF and ERR subroutines are in the file called BSCSUP on the E/A disk. This file is an RORG file so their addresses are relocatible and vary according to where the file is loaded.

MILLERS GRAPHICS - LIMITED WARRANTY

Millers Graphics warrants the DISkASSEMBLER program, which it manufactures, to be free from defects in materials and workmanship for a period of 90 days from the date of purchase.

During the 90 day warranty period Millers Graphics will replace any defective products at no additional charge, provided the product is returned, shipping prepaid to Millers Graphics. The Purchaser is responsible for insuring any product so returned and assumes the risk of loss during shipping.

Ship to:

Millers Graphics
1475 W. Cypress Ave.
San Dimas, California 91773

WARRANTY COVERAGE - This DISkASSEMBLER program is warranted against defective material and workmanship. THIS WARRANTY IS VOID IF THE PRODUCT HAS BEEN DAMAGED BY ACCIDENT, UNREASONABLE USE, NEGLECT, TAMPERING, IMPROPER SERVICE OR OTHER CAUSES NOT ARISING OUT OF DEFECTS IN MATERIALS OR WORKMANSHIP.

WARRANTY DISCLAIMERS - ANY IMPLIED WARRANTIES ARISING OUT OF THIS SALE, INCLUDING, BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO THE ABOVE 90 DAY PERIOD. MILLERS GRAPHICS. SHALL NOT BE LIABLE FOR LOSS OR USE OF THE SOFTWARE OR OTHER INCIDENTAL OR CONSEQUENTIAL COSTS, EXPENSES, OR DAMAGES INCURRED BY THE CONSUMER OR ANY OTHER USE.

Some states do not allow the exclusion or limitation of implied warranties or consequential damages, so the above limitations or exclusion may not apply to you in those states.

LEGAL REMEDIES - This warranty gives you specific legal rights, and you may also have other rights that vary from state to state.

REPLACEMENT AFTER WARRANTY - After the 90 Warranty period has expired you may return any original defective diskette, along with a check for 4.00 to cover shipping and diskette costs, and we will replace it.

MILLERS GRAPHICS
1475 W. Cypress Ave.
San Dimas, CA 91773