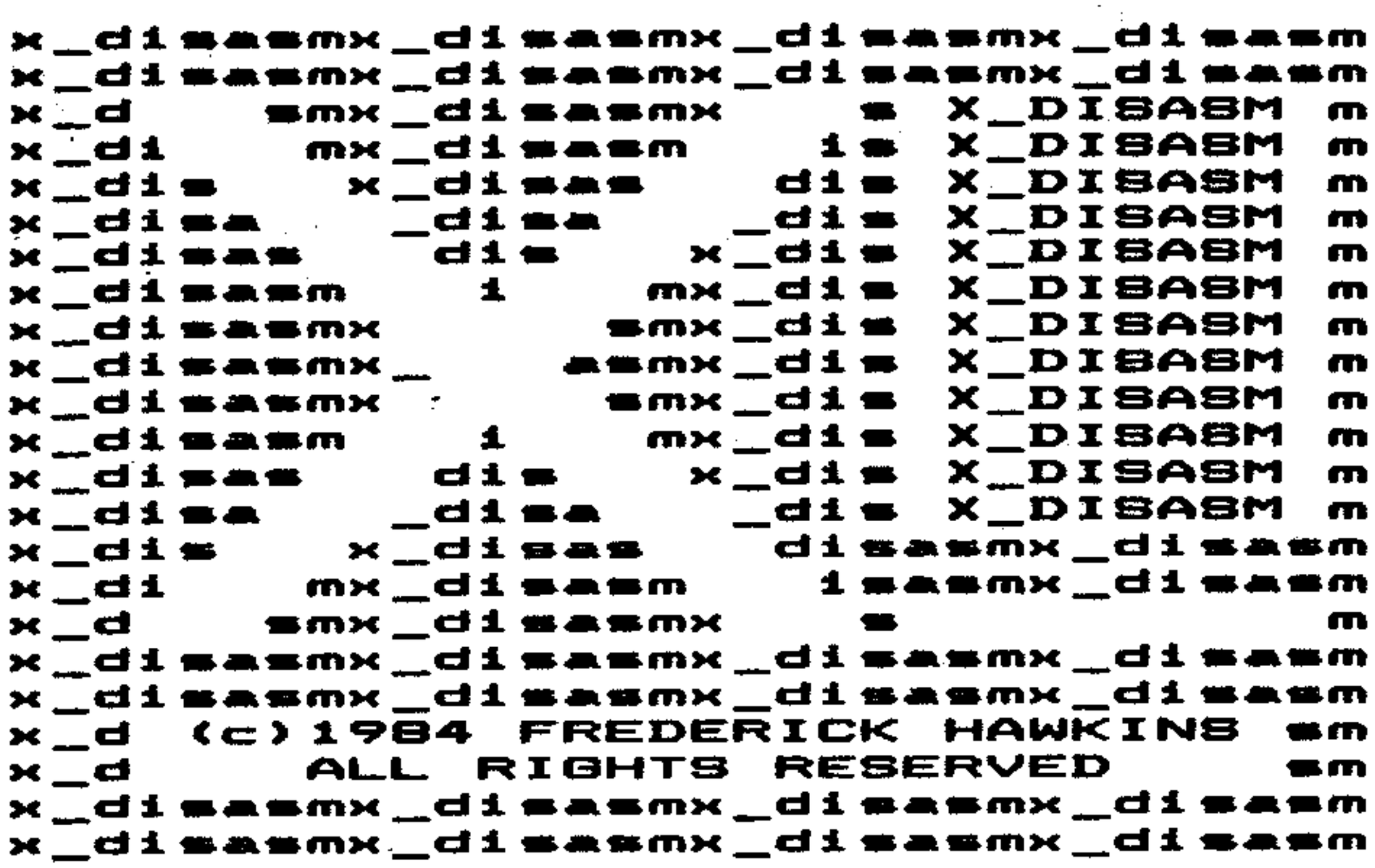


an EXTENDED BASIC disassembler program for the TI 99/4A.

(32K card and disk required)

> DISASSEMBLES to three files AND screen SIMULTANEOUSLY

-- printer, disks, cassette, etc. --

in three different formats:
assembly listing
standard screen
compilable source

> MEMORY DUMP option.

> HEX TO DECIMAL CONVERSION (includes BASIC equivalent).

> FULL FILE CONTROL -- OPEN, CLOSE, DELETE output files.
All are compatible with TI WRITER and EDITOR/ASSEMBLER.

> DISK DIRECTORY option flags Assembly Language files.
Can automatically LOAD XBASIC compatible AL files.

> Built-in program break allows use of XBASIC immediate mode.

> Menu-driven with defaults.

> Disk includes an AL clock program.

Frederick Hawkins -$- 1020 N. 6th St. -$- Allentown, Penna 18102

contents

introduction

First of all, congratulations!

For having the wisdom to buy the best TI 99/4A disassembler available. Certainly it's the best one I could write.

Secondly and perhaps more importantly, X_DISASM is designed to be 'user-friendly'. In other words, the prose weary can reasonably run this program without reading the documentation. (A quick reference is on the back page.)

X_DISASM IS intended to give the TI user what other systems seem to get at the start: a thorough grounding in the operating system. Towards this end, X_DISASM outputs to not only the screen but also a printer, a disk or even a cassette file. The files are compatible with TI WRITER, the EDITOR/ASSEMBLER's Editor or any appropriate BASIC program.

X_DISASM is a learning and development tool for the serious user of the TI 99/4A. It is most useful for making complete listings of console ROMs and relocatable XBASIC AL files. Its output files are designed to fulfill three needs:

1) readability. The primary output file is an assembly listing, the most familiar AL form.

2) disk space. 8K of disassembly in standard screen format easily fits on one disk.

3) practicality. The compilable source option permits using X_DISASM to aid in creating, developing and copying AL routines without tedious entry.

X_DISASM IS NOT intended to be a means to copy third-party software. Consequently, it does not lend itself well to disassembling files prepared and loaded by the EDITOR/ASSEMBLER or those XBASIC files that load into high memory (AORG >A000 and up can potentially overwrite X_DISASM). On the other hand, that's not to say it isn't possible. I suppose that in an ideal world I wouldn't have to mention that. However, inasmuch as many users seem to justify utility programs on the basis of their potential for theft, let's just say that X_DISASM does not go out of its way to help them out.

HOW IT WORKS. With mirrors.

running X_DISASM

X_DISASM will run automatically if the program disk is in DSK1. when XBASIC option is selected.

If you are already in the XBASIC environment, the fastest entry (and the least typing) is to BYE and reselect XBASIC.

For the absolute purist, OLD DSK1.LOAD, followed with RUN, will do.

AL files (uncompressed only) should be loaded under program control with option 5. Any file that is ADRS'ed into high memory MUST be loaded after X_DISASM is running. Care must be taken to avoid overwriting variables.

X_DISASM can not be reRUN after selecting the quit option. You must reload the program again. If you wish to only temporarily halt execution, select the interrupt program option which will allow you to CONtinue running.

If you find a way to cause a run-time error that requires you to rerun the program (CAN'T CONTINUE error message), start over. X_DISASM will not run correctly the second time.

Once X_DISASM is loaded, remove the program disk. Although it will not crash if you attempt to OPEN a file on the program disk, FILE CONTROL and the quit option will henceforth operate incorrectly. (Symptoms: a file cannot be CLOSED or DELETEd and quit returns you to FILE CONTROL. In this case, exit from option 6 --interrupt program-- and exit with either BYE or NEW).

BOOT TIMINGS: X_DISASM requires 12 seconds to load, starts running at 13. The second title appears at roughly 28 seconds and FILE CONTROL at 41 seconds.

menu and screens

X_DISASM has eight different screens. The first two are title screens, both killing time during initializations. The second contains an image that transforms into an instruction bit pattern. (0011=opcode, aa and bb t-fields, dddd specifies the destination word and ssss, the source word). This screen is the only one that calls sound, mostly as a convenience: the second tone signals that X_DISASM is finished booting.

The first operating screen is FILE CONTROL, corresponding to menu option #3. This screen allows you to determine which files X_DISASM will output to and later change those files. The default file #0 is the monitor screen. Note that the menu hasn't yet appeared. Although this sequence may be somewhat disconcerting, the practiced user will find it convenient to set up her files before continuing.

The next screen contains the menu which is essentially a reminder list. The warning about the 99/4A's memory-mapped block specifies the range of addresses that do not contain memory that can be either disassembled or dumped. Because X_DISASM does not prevent access to this block, the potential for system lock-up exists if you allow either option into this block. Additionally, the method for accessing the hex to decimal conversion is noted.

The option prompt always contains a reminder that #8 will list the menu. From this prompt, you have eight choices, selected by their corresponding numbers.

1) disassembly) both include
2) memory dump) hex to decimal
3) file control
4) multiple output
5) disk directory and load
6) interrupt program
7) quit program
8) repeat prompts

Options 3, 4, 5 and 8 will erase the monitor's current contents. Options 1, 2, 6 and 7 will simply scroll it off. Note that options 1, 2 and 8 stay selected until changed. For example, to continue DISASSEMBLY simply press ENTER.

INPUTS, IN GENERAL: Use the Alpha Lock; most of X_DISASM's input fields require upper case. The option prompt and many of the secondary fields require only a single letter and will prevent entry of a second character. The current option can always be changed before you ENTER, by continuing to type. For those fields that require more (addresses, file names) full editing is possible; deletion, insertion, clear are enabled. Although X_DISASM attempts to make entry as bullet-proof as possible with much error recovery, be careful with those fields that require numeric entry.

file control

X_DISASM's file handling allows you to name, OPEN, CLOSE, and DELETE output files, as well as redirect output. FILE CONTROL error traps for most types; however, it does not check for legal device names.

File numbers 1, 2 and 3 have default devices, names and record formats. Each file's current status, OPEN or CLOSED and device.filename, is shown whenever FILE CONTROL is entered. After OPENING a file, the program allows you to select and OPEN additional files. The primary output file is always the last file chosen.

The FILE CONTROL screen has four input fields:

1) output redirection:

Chooses the file to which output will be directed. ENTER the appropriate file number. Upon re-entry, FILE CONTROL will always show the last selection.

The default output file, #0, is the monitor screen. Selecting #0 will skip the remaining prompts because the screen file is always OPEN and requires no attributes.

Unselecting an output file does not CLOSE the previous file. Depending on the file's OPEN status FILE CONTROL will perform either input 2 or 3, below.

If you are returning to the FILE CONTROL environment after using the MULTIPLE OUTPUT mode, you will notice that the file number is 4. An error trap prevents you from not changing this to a legal number (0 through 3).

2) filename validation:

Selecting a CLOSED file allows changing or accepting the current device.filename. Full XBASIC editing is possible.

File #1: Assumes printer whose default name is "RS232.BA=4800.DA=8". NOTE: This file may not be DELETEd. Record format is assembly listing.

File #2: Assumes "DSK1.DX1_" as default. The record format is identical to the screen image. Most useful for saving the greatest amount of disk space.

File #3: Assumes "DSK2.DX2_" as default. Format is like an AL EDITOR/ASSEMBLER source file.

The validation routine always returns to the redirection prompt. This allows you to OPEN all the files you wish to use.

3) exit and error recovery:

If your primary file is not the monitor screen, FILE CONTROL prevents you from entering the menu environment until you answer affirmatively. Notice that the file status line is redisplayed and that the default assumes a correct filename.

A negative reply passes control to input field 4, next.

4) close or delete files:

Closing a file allows you to redefine a file channel. If you CLOSE and then OPEN a file under the same filename, X_DISASM will overwrite the first version.

XBASIC has a limit of three OPEN files. In order for the DISK DIRECTORY option to operate you may have to CLOSE a file.

If you have created a disk file with an incorrect filename you may DELETE it. Deletion will also CLOSE the file. FILE CONTROL can DELETE only files 2 and 3; it will rename either file with "DSK1.DX3_". To DELETE file #1, you must use the immediate mode. DELETE "device.filename", OPEN #1 with a dummy file (RS232 or CS1) and return to FILE CONTROL to CLOSE the dummy. In this manner X_DISASM's file flags will be correct.

The BACK option is for inadvertent entry into this routine and does not restart FILE CONTROL. If you wish to redirect the program's output, yet not CLOSE the primary file, you may use this sequence: B)ack [ENTER] Y [ENTER] 3 [ENTER].

Error trapping on DELETE and CLOSE will prevent a crash. However, should you be unable to CLOSE or DELETE a file (disk is removed, program disk in place when OPENed, etc.), X_DISASM will not exit correctly. If the correct disk is inserted, or a satisfactory dummy created (in the immediate mode), you may DELETE or CLOSE the file.

file attributes

All X_DISASM output files are defined DISPLAY,SQUENTIAL,VARIABLE 80. This format is readable from TI WRITER and EDITOR/ASSEMBLER Editor.

file formats

Each of the output files format the DISASSEMBLE option differently. Additionally, the third disassembles the machine code into somewhat more difficult form. Each has its advantages and uses.

Format/file #1: ASSEMBLY LISTING

```
25C8 0285        CI R5,>012C     This is the preferred printer format,
25CC 1302        JEQ >25D2       allowing ample space for labeling code.
25CE 0585        INC R5
25D0 0380        RTWP            Jumps are fully decoded to an absolute
25D2 04C5        CLR R5          address, as are branches.
25D4 0283        CI R3,>258E
25D8 110D        JLT >25F4       The fields are memory location, contents
25DA 0203        LI R3,>2546     at that location, blank label, and
25DE 0606        DEC R6          disassembled code.
25E0 1609        JNE >25F4
25E2 0206        LI R6,>0005     Note that what ordinarily would be a
25E6 0224        AI R4,>0006     label is decoded as an absolute address.
25EA 0284        CI R4,>258E
25EE 1102        JLT >25F4
25F0 0204        LI R4,>2546
25F4 C1C4        MOV R4,R7
25F6 C837        MOV @R7+,@>2536
```

Format/file #2: STANDARD SCREEN

```
25C8 0285  CI R5,>012C       Here is the same code fragment showing
25CC 1302  JEQ >25D2         the compacted label space.  Although this
25CE 0585  INC R5            format isn't, in an absolute sense, smaller
25D0 0380  RTWP              than the third, all the information of file
25D2 04C5  CLR R5            #1 is here.  The layout is identical to the
25D4 0283  CI R3,>258E       monitor display, save that lines longer than
25D8 110D  JLT >25F4         28 characters do not wrap to the next line.
25DA 0203  LI R3,>2546
25DE 0606  DEC R6
25E0 1609  JNE >25F4
25E2 0206  LI R6,>0005
25E6 0224  AI R4,>0006
25EA 0284  CI R4,>258E
25EE 1102  JLT >25F4
25F0 0204  LI R4,>2546
25F4 C1C4  MOV R4,R7
25F6 C837  MOV @R7+,@>2536
```

Format/file #3: COMPILABLE SOURCE

```
CI R5,>012C
JEQ $+2
INC R5
RTWP
CLR R5
CI R3,>258E
JLT $+13
LI R3,>2546
DEC R6
JNE $+9
LI R6,>0005
AI R4,>0006
CI R4,>258E
JLT $+2
LI R4,>2546
MOV R4,R7
MOV *R7+,@>2536
```

This format provides a blank space for labels. Although you must provide DEF's, REF's and the like, the intent is to speed development of modules that use code already in ROM/RAM.

Jumps are relative, and will compile without changes into relocatable code.

Branches and all other references are fully decoded as absolute addresses. Some, like references to PAD locations (>8300 to B3FF), can stand unchanged. Others will require labeling.

```
CI R5,>012C
JEQ >25D2
INC R5
RTWP
CLR R5
CI R3,>258E
JLT >25F4
LI R3,>2546
DEC R6
JNE >25F4
LI R6,>0005
AI R4,>0006
CI R4,>258E
JLT >25F4
LI R4,>2546
MOV R4,R7
MOV *R7+,@>2536
```

Disassembly with MULTIPLE OUTPUT option. Notice that the jumps are absolute. In effect, each jump is now to a label, which you must figure and provide.

Branches likewise require replacement.

DATA statements (no example here, sorry) are actually decoded on the basis of illegal opcodes. Consequently, you must examine each one separately and adjust code as needed. Furthermore, in the COMPILABLE SOURCE format DATA directives carry no information into the file. This will protect you as the file will not compile without flagging each as an error.

disassembly

DISASSEMBLY is the whole point of X_DISASM. Perversely, there isn't a whole lot to be said about it.

The disassemble option (#1) requires a starting address (in hex). The last four characters are always decoded as the address. Therefore, you may correct an error with the BASIC edit functions or keep typing. Valid hex digits are 0 through 9 and A through F. Any other character will evaluated as zero.

Attempting to disassemble in the middle of a multiple-word instruction like MOV @>8300,>834A will give incorrect results. You must examine the start address for correctness. (>8300 = C R0,R12).

If you reply to the address prompt with either 'H' or null, you will access the hex to decimal conversion. X_DISASM will then convert any four digit hex number to a decimal number and its two's complement (BASIC's standard). The conversion routine restarts the disassembly option. You may repeat this conversion routine as many times as you like.

The second prompt in the DISASSEMBLY environment requires a decimal count of the expected numbers of words to be disassembled. This number, unfortunately, is only a guide. The error involved depends upon how many multiple-word instructions are disassembled. Thus, for short sequences, it may be ignored. Longer sequences, say about 4K or more, require that you underestimate. Usually the most dependable method is to create a file in sections, disassembling about 500 words at a time.

You may disassemble to the screen, noting addresses and then using FILE CONTROL, redirect output to another file. Repeating the same address and count will duplicate what the screen showed to the file (format may differ, depending upon file number). Until any file is CLOSEd you may select it and add to it.

When disassembling to a file, a current location is displayed. DISASSEMBLY can output to more than one file at a time; see the MULTIPLE OUTPUTS section.

Quick exit: give starting address of 0, number of words 0. For ordinary browsing purposes, the average screenful of disassembled text has a word count of 24.

What's taking so long? Although DISASSEMBLY requires words (16 bits or two bytes), XBASIC accesses a byte at a time. These decimal values are converted to a hex word. The high byte is broken into two nybbles and checked for an opcode match. On no-match the low byte is broken. After the opcode is found, the word is broken down to a bit image. Depending upon the opcode's format, the bits are decoded to get the operands. Additional memory words are read at this time as required. At every step the information is formatted and finally printed to the appropriate files.

It sometimes helps to consider that disassembly occurs at a comfortable reading speed.

Quantitative benchmarks: a rough guess is that X_DISASM will take about 13 minutes per 1K of disassembly.

memory dump

The MEMORY DUMP option (#2) lists the contents and their equivalent ASCII characters, when printable. Each line begins with the hex address of the first byte of the sequence of six bytes. Bytes are doubled up into words. However, you will notice that unlike DISASSEMBLY, you can dump from an odd starting address.

For ease of use, MEMORY DUMP shares with DISASSEMBLY the same front end, including the hex to decimal routine. It does not share the same problems with the decimal count (of bytes, this time). Rather, the dump will access the given number of bytes plus enough to fill out the current line.

To fill the screen without scrolling any off, specify a byte count of 120. When output is to a file, the screen displays the beginning address of each line.

MEMORY DUMP can not output to more than one file at a time. If you select this option after the MULTIPLE OUTPUTS option, output will be directed to the screen. (A warning message is displayed.) This was to done to keep up the routine's speed, because MULTIPLE OUTPUTS option degraded the performance too much to justify the change.

There is no change in format from file to file. This is particularly apparent when outputing to a printer. At best, you'll have lots of room to write comments. At worst, you'll use a lot of paper.

```
2000=>205A 2614 3FF0  Z&_?_

3FF0=>434C 4F43 4B20 CLOCK
3FF6=>258E 405F 2020 %_@_

2514=>2183 03F0 2536 !___%6
251A=>0008 256A 2546 __%j%F
2520=>0090 0001 254C ____%L
```

MEMORY DUMP: typical examples. The blank lines were inserted for legibility.

hex to decimal conversion

Access to the conversion routine is through either the DISASSEMBLY or MEMORY DUMP options. The address query will branch to the hex to decimal conversion if you answer 'H' (or a null string —this is an XBASIC bug— but useful).

The conversion will decode the last four hex digits as the input string and display its decimal value and its two's complement. The subroutine will then return to the address prompt of the calling option.

The output of this routine is to the screen only. You may select it as often as you wish. You will eventually have to either DISASSEMBLE or DUMP before X_DISASM returns to the MENU.

multiple outputs

One of X_DISASM's unique features is the MULTIPLE OUTPUTS option (#4) which permits you to DISASSEMBLE to more than one file and the screen, simultaneously. This option is particularly useful in creating a disk file and a hardcopy in one pass as well as monitoring on the screen what is being written to disk.

There is a single input field and one accept/redo prompt in MULTIPLE OUTPUTS. The input field lists, by file number, the currently OPEN files and gives you the opportunity to strike a file from the list. In addition, the current status line for each OPEN file is displayed.

Note that striking a file IS NOT the same as a DELETE. The file is merely taken off the list of output files only as far as the present entry to MULTIPLE OUTPUTS is concerned. The files are not CLOSEd, DELETEd or changed in any way.

The accept/redo prompt allows you to reconfigure your list before exiting. A third choice, "Open files", will transfer you to the FILE CONTROL screen.

Striking a file is performed by merely spacing over the corresponding file number. (Full BASIC editing is enabled.) It is possible to create a not very useful null list. Probably runs faster, though. You may unselect the screen (file #0), in which case the current disassembly address will be displayed.

MULTIPLE OUTPUTS only creates the output list. You will have to return to the DISASSEMBLY environment to actually use it.

MEMORY DUMP resets the output list to the screen. MULTIPLE OUTPUTS uses the output file number as a flag (4). Therefore, FILE CONTROL will prevent exit until you select a legal file number. The other MENU options have no effect and may be used freely.

directory & load

X_DISASM's DIRECTORY routine (#5) lists, up to a maximum of 44, the filenames on DSK1. Files with attributes that mimic AL files (DISPLAY,FIXED 80) are flagged "$=>"; otherwise the file's size in sectors is displayed. No attempt is made to distinguish progrees from other files. The diskname and available sectors is also displayed.

DIRECTORY has two secondary options: LOAD AL files and re-INITialize memory (CALL INIT).

DIRECTORY has three levels and an omnibus error trap. Inputs are all single characters, with defaults.

1) preparation — Asks whether correct disk is in place. You may perfora a CALL INIT from this screen, skip right to the directory, or exit ('E') to to the MENU. This last is unprompted.

2) directory — Lists directory. Allows restarting if you do have an incorrect disk in place, branching to the LOAD AL files screen or returning to the MENU.

3) load files — Finds each flagged file and waits for your decision. The default is yes. Any other character will prevent loading. An incorrect file (usually compressed AL) will not load; an error message is displayed and the search for next file continues. Upon completion, control is returned to the MENU.

The major error trap routine catches the program if the disk isn't initialized, device error, etc. etc. You may either exit to the MENU or retry the directory from this screen.

The CALL INIT makes it possible to LOAD and DISASSEMBLE several files, without risking a memory full error. LOADING programs to the same general address is also convenient.

The LOAD routine does not check for files AORG'ed into the XBASIC program space, nor can it prevent an AL program with END START capability from immediately executing.

interrupt program

The INTERRUPT option (#6) permits you, the user, to have your cake and eat it, too. Given that no program is perfect, you can, within the limits of your patience, make use of XBASIC's immediate mode to rectify X_DISASM's (hopefully few) shortcomings. Some typical uses might include ordinary calculator math, LOAD and LINK AL programs, send printer control codes, or even insert comments in files.

Essentially, INTERRUPT is a menu-selected BREAK. To resume the program, use the command 'CON'. INTERRUPT may be used to correct certain error conditions: see QUIT, below.

USE three letter variables. Although in most cases two letter names will not interfere with X_DISASM's, there are no variables with three characters.

Files are not CLOSED or affected; therefore do not use INTERRUPT to exit the program unnecessarily. In addition, you may not OPEN a fourth file from the INTERRUPT mode. In other words, if you have three files OPEN, you'll have to choose one to CLOSE or forego the fourth.

A simple demonstration:

```
select option 6
CALL LOAD("DSK1.CLOCK)::CALL LINK("CLOCK")
CON
```

quit program

The QUIT menu option (#7) does just that. Since X_DISASM will not RUN correctly without a fresh load, and equally important, all OPEN files are CLOSEd by QUIT, you must answer an exit prompt. This prompt, unlike all others, has no default.

As noted on pages 4 and 7, QUIT will return you to FILE CONTROL if a file is not on-line (disk removed) or incorrectly OPENed (program disk left in drive #1 on OPEN). In the former instance, put the correct disk in its appropriate drive and QUIT again. The second instance requires that you select the INTERRUPT PROGRAM option and either create a dummy file with the same file number as the invalid file, CON and try to QUIT again or simply BYE, NEW, STOP or QUIT (XBASIC keyword, this time) from the immediate mode.

Useful dummy filetypes are "RS232" and "CS1". Neither one requires any preparation and either may be used without actually turning on a device. (Except that for "RS232", you'll need the card.)

'real time' clock

The X_DISASM program disk contains a sample AL program that demonstrates the real time function. Although the disk's protection prevents you from listing its files and thereby disables X_DISASM's auto-LOAD, you may LOAD it from the INTERRUPT option. The proper syntax is listed, left.

The 'real time' clock shows only minutes and seconds in increments of 5 seconds and 5 minutes. This is a function of the image resolution. It's tough enough to show a clock in one character; to move any hand 60 times is beyond a reasonable image. The undisplayed accuracy is the more typical 60 parts per second, derived from the user interrupt vector.

The ins and outs of how it all works is left to you... Using X_DISASM, you should be able to disassemble the program and dump the data, as well as create your compilable version. The CLOCK program is henceforth in the public domain, with one condition: that all copies distributed carry credit to TOM BONGE and FRED HAWKINS.

getting around

The important XBASIC pointers are at locations >2002 and >2004. The first shows the lowest available byte in low memory. The second points to the beginning of the user REF/DEF table. When the first is subtracted from the second, the available space for LOADing AL files is found.

```
2000=>205A 2614 3FF0  Z&_?_
```

>2004 points to REF/DEF start at >3FF0.
>2002 points to 1st available byte. (Where next file LOADs.)

```
3FF0=>434C 4F43 4B20 CLOCK
3FF6=>258E 405F 2020 @_9_
```

REF/DEF table showing a file named CLOCK. The entrance point (not necessarily the file's beginning) is at >258E.

In this example you would start disassembly at >258E. Although you may skip this step, you might dump the contents of >2002 before LOADing an AL file. Using this data and the contents of REF/DEF table pointer after LOADing, you may determine potential machine code and data you might otherwise overlook.

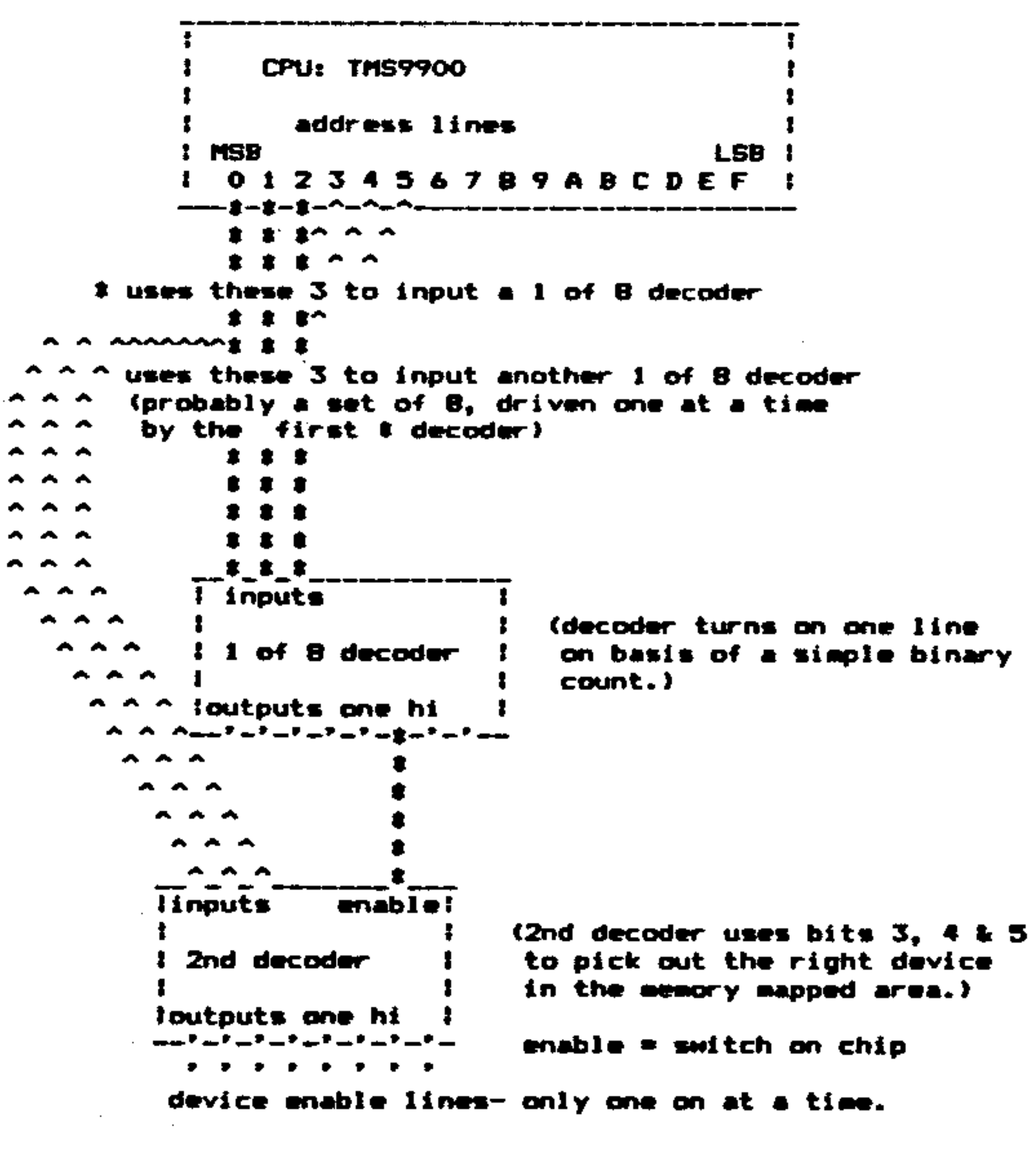
The single most important location required to make sense of console ROM is >0CFA. This points to a set of eight pointers, that in turn point to tables of routine starting addresses. The first two table pointers point to the console ROM tables.

Additionally, hardware-generated interrupts (like turning on the system -- RESET) use memory locations >0000 and >0004 as BLWP vectors. The first is the RESET, the second is the INTERRUPT vector (generated by VDP, internal timer or peripheral box).

memory-mapped block

CPU MEMORY MAP: Memory Mapped Device Block >8000 to >9FFF

Addresses >8000 to >80FF, >8100 to >81FF, and >8200 to >82FF are really phantom locations of PAD. If you use the EDITOR/ASSEMBLER'S DEBUG and compare the memory blocks looking for diferences, there aren't any. Using this info, we can make a stab at guessing the hardware:



Back to the CPU chip and its most significant 8 bits:

```
        !
        ! 0 1 2 3    4 5 6 7  address lines
        --!-!-!-^----^-^-----------------------
```

! 0 0 0	0 0 0 0	>8000 to 80FF	PAD chip
! 0 0 0	0 0 0 1	>8100 to 81FF	
! 0 0 0	0 0 1 0	>8200 to 82FF	
! 0 0 0	0 0 1 1	>8300 to 83FF	
! 0 0 0	0 ^ 0 0	>8400	SOUND chip
! 0 0 0	^ 0 0 0	>8800	VDP read data
! 0 0 0	^ 0 0 0	>8802	VDP status
! 0 0 0	^ ^ 0 0	>8C00	VDP write data
! 0 0 0	^ ^ 0 0	>8C02	VDP write addr
! 0 0 ^	0 0 0 0	>9000	SPEECH read
! 0 0 ^	0 ^ 0 0	>9400	SPEECH write
! 0 0 ^	^ 0 0 0	>9800	GROM read data
! 0 0 ^	^ 0 0 0	>9802	GROM read addr
! 0 0 ^	^ ^ 0 0	>9C00	GROM write data
! 0 0 ^	^ ^ 0 0	>9C02	GROM write addr

Not to be awfully obscure, I'll explain. When you CALL LOAD(-32768,0) or, in AL, LABEL MOV R3,@>8000 you're really setting the address lines to hex 8000.! The bit image would be:

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

The first six bits are decoded to turn on the PAD chip, and the last eight get the appropriate address in memory.
Bits 6 and 7 are not used; therefore hex >8000 and >8300 are effectively,the same memory location. I suspect TI originally designed a 1K byte PAD and then cut manufacturing costs with the smaller 256 byte chip. A more sinister supposition might be that the 1K chip was planned for a later, upward version.

What's it all mean? Put simply, 'Memory Mapping' uses addresses to turn on devices external to the CPU. It also makes those addresses unavailable for ordinary data storage-- so the 64K 9900 becomes the 36K 99/4A. Consequently, X_DISASM (or for that matter, any other program) can not access this block for either disassembly or dumps.

It also means you can write your own subroutines equivalent to VSBW, and so on, as long as you follow the protocol required by the appropriate chip.

! The data bus gets the value you're transfering, >00 in the BASIC command, or the contents of register 3 in the assembly language example.

reformatting

Files of disassembled code created with X_DISASM can be directly edited by either TI WRITER or the EDITOR/ASSEMBLER's editor. Skillful use of several of their commands can aid the readability of the files.

```
**************************************************************
*          (dummy header- you might put a filename here)
*
*(Place at lines 1 through 6 of the disassembly file.
*        Copy throughout the listing as you wish)
**************************************************************
```

Insert, using the Copy function, line 1 after each unconditional break in the program's logic, (after JMPs, Bs, RTWPs). Insert a blank line in front of all lines that are branched or jumped to.

Using the Replace String function, flag the label space in front of all jumps and branches. Codes: 'e' in front of JMP and B, 'o' for BL and BLWP, 'u' for any other jump that goes towards a lower address. Additionally, a very easy sort of flowcharting uses this RS command: '18 18 / / !/' or 'V,24(18,18)/ / !/'. This should be used after short jumps (limit to one page or about 20 lines). At the line to which control is passed, insert a blank and terminate the line with a '/'. Use a half-page dotted line to mark off the code not executed. The Replace String is also extremely useful for changing operands. For instance, change @>8C02 to @VDPWA.

```
25CB 0285          CI R5,>012C
25CC 1302          JEQ >25D2
                   !-------------------------------------
25CE 0585          ! INC R5
25D0 0380        e ! RTWP
                   !-------------------------------------
                   /
25D2 04C5          CLR R5          the same old program fragment,
25D4 0283          CI R3,>258E     compare with pages 8 and 9.
25D8 110D          JLT >25F4
                    !------------------------------------
25DA 0203           !LI R3,>2546
25DE 0606           !DEC R6
25E0 1609           !JNE >25F4
                    ! !-----------------------------------
25E2 0206           ! !LI R6,>0005
25E6 0224           ! !AI R4,>0006
25EA 0284           ! !CI R4,>258E
25EE 1102           ! !JLT >25F4
                    ! ! !---------------------------------
25F0 0204           ! ! !LI R4,>2546
                    / / /
25F4 C1C4          MOV R4,R7
25F6 C837          MOV @R7+,@>2536
```

compressed AL

Because X_DISASM doesn't attempt to rewrite XBASIC's CALL LOAD utility, it can not handle compressed AL files. What follows is a description of uncompressed and compressed, and some notes on how you might write a program that gets around these difficulties.

First of all, it is imperative that you read section 15 of the EDITOR/ASSEMBLER manual. This is the sole description of tagged object code.

The fundamental insight into tagged object code turns on the observation that the tags occur at regular and more or less predictable places in each AL file record. The difference between compressed and uncompressed files then becomes only one of length -- a compressed file has more fields per record.

An uncompressed field is made of the CHARACTERS 0 through 9 and A through F, representing the data. The corresponding compressed field IS the actual data. For example, compare CHR$(0);CHR$(255) with "00FF".

Thus, the simplest method to enable LOADing compressed files by XBASIC requires expanding the compressed file. Each record will hold fewer fields. Each file will have more records.

The translation program should:

- Dependably distinguish all tags, and properly handle their following field or fields.
- Make absolute code relocatable. (Change tag character 9 to tag A. Requires keeping a count of words.)
- Change checksum tag 7 to tag 8, ignore checksum.
- Follow external references (tags 3 and 4) back up through the file, replacing each reference with an absolute value. (XBASIC accepts only EQU's, not REF's.)
- Eliminate code generated by END START assembly.

Your attempt at first should be to enable XBASIC to LOAD the file, not necessarily LINK.

A simple XBASIC program that prints to the screen a compressed file record by record could look like this:

```
 96  OPEN #1:"DSK1.(compressed AL filename)",FIXED 80
100 LINPUT #1:A$
104 FOR A=1 TO LEN(A$)::PRINT ASC(SEG$(A$,A,1));" ";
108 NEXT A
112 INPUT "enter for next record":A$
116 GOTO 100  !pgm will end on an error
```

references

FUNDAMENTALS OF MICROCOMPUTER DESIGN (MPB30A, $15; AKA 16-Bit Microprocessor Systems, McGraw-Hill, about $45) Written for a college level class. Not too specific about 'this wire to that wire' but very good and thorough about hardware in general. Specific examples are based on the TMS 9900. The second half runs through software pretty well. It has the best explanations of the AL commands and the graphics are good.

TM9900 FAMILY SOFTWARE DEVELOPMENT HANDBOOK (MPA29, $8.30 from TI) I used to think this was the best book for the money, but I use 'Fundamentals' more. Worth reading for an understanding of structured programming, as well as PASCAL (not UCSD), 'POWER' BASIC (again a different sort) and finally assembly. Includes a short yet interesting section on tips and techniques for AL programming and a complete bibliography.

MICROPROCESSORS/MICROCOMPUTERS/SYSTEM DESIGN (LCB5351, 29.95) Used to be 9900 Family Systems Development Handbook, is little more broadly based but easy-to-read intro to hardware and software. Its AL is just as useful as Software Dev., merely cleaned up graphically. It doesn't have the tips and hints, although the last chapter is a complete listing of a floppy controller program.

TM 990/UBG USER'S GUIDE (MPB06C, $11.45) Pretty specific about the University board, hardware, and in-ROM monitor. OK on AL but the other books do the job. Interesting from a systemic approach to the subject of 9900-based micros. Some wiring diagrams and modifications included but nothing directly useful to the 99/4A user. Best used as another source to demystify the hardware.

OTHER DATA BOOKS:

TMS 9900 MICROPROCESSOR DATA MANUAL (MP001 REV A)
TMS 9918A VIDEO DISPLAY PROCESSOR D.M. (MP010A)
TMS 5220 VOICE SYNTHESIS PROCESSOR D.M. (DM-02)
TMS 9901 PROGRAMMABLE SYSTEMS INTERFACE (MP003)
TMS 9902A ASYNC. COMMUNICATIONS CONTROLLER (MP004A)

OSBORNE 16-BIT MICROPROCESSOR HANDBOOK has two or three GREAT tables. One, table 3-3, 9900 instruction set object codes, was the single most useful reference I had when I wrote X_DISASM. However, this book will not help you write programs -- don't expect it to do more than place the TMS 9900 in some perspective.

all save Osbourne:
Texas Instruments
POBox 3640, M/S 54
Dallas, Texas
75285

Data manuals:
Texas Instruments
POBox 225012, M/S 308
Dallas, Texas
75265

OSBORNE:
OSBORNE/McGraw-Hill
630 Bancroft Way
Berkley, Calif.
94710

odds and ends

X_DISASM was written for a color system. It's always struck me as a mystery why utility programs don't all use at least some of the color capability. The choices for the colors was easy: they were the colors I was using when I wrote that particular part of the program.

A typical piece of coding that you might use:

```
4 !CALL SCREEN(14)::FOR A=0 TO 14::CALL
COLOR(A,16+11*(A=1)+14*(A>4)+(A>8)*(A<13),5-10(A>0))::NEXT
A::ACCEPT AT(5,24):A$
```

The exclamation prevents execution. To use, EDIT 4, ENTER, REDO, space over the "4 !", ENTER, and finally CLEAR. Until you execute an immediate command, RUN, or create a syntax error (usually mismatched quotes) your screen will stay as you set it. LIST, NUM, OLD and SAVE have no effect.

warranty

Aside for defective materials, for which a replacement will be made in exchange for the original within 15 days of purchase, this program is offered only in as-is condition, owing chiefly to the popularity of user group duplication. I will not be liable for implied merchantability or usefulness, nor will I guarantee X_DISASM's ability to run on every version of the TI 99/4A.

I will, within limits of patience and goodwill, respond to queries and suggestions about X_DISASM and its documentation.

Replacement disks are available at $6.00 in exchange for the original. Copies of the documentation are available at $2.50.

Address all correspondence, orders and the like:

Frederick Hawkins
1020 N. 6th St.
Allentown, Penna. 18102

trademarks

X_DISASM

quick reference

menu #	description	prompts	
1)	DISASSEMBLY	start hex address ['H'=hex to dec routine]	
		# of words (decimal)	
2)	MEMORY DUMP	start hex address ['H'=hex to dec routine]	
		# of bytes (decimal)	
3)	FILE CONTROL	current output file	#0: standard, monitor
		validate filename	#1: assembly listing
		ok to proceed	#2: standard screen
		DELETE or CLOSE	#3: compilable source
4)	MULTIPLE OUTPUTS	accept/change filelist	
		accept, redo, OPEN	
5)	DISK DIRECTORY	proceed, INIT and proceed, (exit)	
		LOAD, redo, exit	
		ok to LOAD	
6)	INTERRUPT PGM	> immediate mode, CON to resume	
7)	QUIT PROGRAM	ok to quit	
8)	REPEAT MENU PROMPTS		

important addresses:

>0CFA beginning of eight pointers to XMLLNK tables

>2002 first available low memory address

>2004 REF/DEF table address pointer
Each REF/DEF entry is made of a 6-character name & that program's starting address. (6 bytes plus 2 bytes, total: eight byte REF)

miscellaneous files:

DSK1.CLOCK --AL uncompressed file, demonstrates user interrupt

All X_DISASM-created files are DISPLAY,VARIABLE 80.

June 21

Dear Mr. Nomina

Here's your copy of X-DISASM, I hope you find it worthwhile. Also you will find what passes for X-D's user response form.

The file DIRECTORY details the disk's contents— CAT, CLOCK, LOAD and COLOR are the nominal X-DISASM files, the others may be deleted freely.

CAT, more usually called LOAD, will ~~crash~~ apparently fail on the first pass; simply hit "M" at the bottom prompt. CAT must be edited from a MERGEd version because the 5)keeleton option will render the edited pgm useless. In any case, delete the 5)keeleton option's code from the version you create.

You may be amused to know that you have the 40th X-DISASM; breaking even seems likely now.

Have fun with it —

Frederick Hawkins