MINI-COMPUTER PROGRAMMING SERV.
P. O. BOX 1943
ROCKY MOUNT, NORTH CAROLINA 27801

H. DAVID KEIM II PHONE: 442-4417

JANUARY 1980

TO PURCHASERS OF ASPTCH 2.0

Thank you for your order.

Micropute Software hopes that you will find many uses for your new copy of ASPTCH 2.0. In addition to the ASPTCH program, you will find your program cassette contains two complimentary programs that can be used with ASPTCH 2.0. Your ASPTCH package should contain the following:

1. One certified cassette tape with three programs: ASPTCH, SYSTPE, and BREAK
2. An instruction manual for ASPTCH and an appendix for each of the two complimentary programs
3. A letter from the program author
4. A questionnaire/permit page

When using your ASPTCH program for the first time, be certain to read through the instructions carefully and follow examples which have been provided as instructional aids. Tapes for this program are of high quality and usually load at about four and a half on those computers not having the cassette modification.

Any questions you may have can be answered by sending a self-addressed, pre-stamped envelope to James Williams, c/o Micropute Software at the above address.

NOTE: Enclosed with this package is a questionnaire/permit page which must be filled out, signed, and returned to Micropute Software immediately. Signing the agreement portion of the questionnaire will qualify you for automatic receipt of any update newsletters on ASPTCH and permit you to submit questions without charge for consultation. If you send inquiries, please include as much documentation as possible concerning your problems.

Sincerely,

H. David Keim II

H. David Keim II

January 1, 1980

DEAR FELLOW ASSEMBLY LANGUAGE PROGRAMMER:

Thank you for purchasing ASPTCH 2.0. I am very proud of ASPTCH. It is the product of 10 months of work and has evolved through ten distinct versions. Each version has been used to create the next one, as well as many other machine language programs. EDTASM/ASPTCH is by far my most used program. I load it almost every time I power up. Many TRS-80 programmers in the area used revisions of ASPTCH as they were developed. Version 2.0 is the sum total of their comments, suggestions, and needs.

Loading can be one of the most frustrating problems of any program, especially programs not made on one's own cassette player. For example, I have found EDTASM very difficult to load. If a program is to be used quite often, making a back-up copy on one's own cassette recorder is imperative. For this reason, I have included the source listing of the utility program, SYSTPE. SYSTPE permits you to copy any number of different areas of memory, and combine them into one system format tape. Because there are so many overlays, the combined tape of EDTASM and ASPTCH will load in about the same time as EDTASM alone.

The back-up capability lends itself to abuse. The copies you make are for your use only. If you give away or sell a copy of EDTASM/ASPTCH, SYSTPE, or BREAK and/or any of the printed instructions, you will be in flagrant and knowing violation of the copyright law. Please tell a friend about ASPTCH, the time spent in its preparation, the utilities, and instructions, and ask him to <u>purchase</u> a copy from Micropute Software.

The ability to edit a source listing and execute without using tapes is a very valuable and flexible debugging aid. I have included a second utility, BREAK, to help you utilize this feature to set as many break points in a program as you desire. Simply add a call to this subroutine in your program, dump it into memory, and execute. Your program will stop execution when the break point is reached, and all registers and flag conditions will be displayed. After a break, you may continue execution or return to EDTASM.

I sincerely want ASPTCH and the utilities to be as <u>useful</u> to you as possible. Please direct questions, comments, or problems to me by writing to me c/o Micropute Software. If your tape is defective or fails to load, please return it for another copy. Happy assembling.

Sincerely,

James Fuller Williams

James Fuller Williams

INSTRUCTION MANUAL
ASPTCH 2.0

by

James Fuller Williams

Micropute Software, P.O. Box 1943, Rocky Mount, N.C. 27801

NOTICE

ASPTCH 2.0

I. General Information

ASPTCH 2.0 was designed for cassette based machine language programmers using Radio Shack's EDITOR/ASSEMBLER 1.2. Without ASPTCH, EDTASM must be reloaded after every test of a machine language program. ASPTCH adds the following features, none of which require reloading of EDTASM or ASPTCH:

1. Reserve memory for machine language programs.
2. Dump assembled programs directly into memory, without using the cassette recorder.
3. Execute dumped program.
4. Display number of bytes left in text buffer.
5. Convert Hex to Dec and vice-versa. Also show contents of memory locations in Hex, Dec, and printable ASCII character.
6. Enter Level II BASIC's monitor mode (system prompt). Allows jumps to any location in memory and loading of system tapes.
7. Enter Level II BASIC's command mode (ready prompt). Use Level II commands that do not use variables (used mainly for calculator type functions and the POKE command).
8. Return to fully programmable BASIC with EDTASM/ASPTCH and optionally dumped programs protected in high memory. ASPTCH's key debounce routine may be re-activated while in BASIC to prevent keybounce and speed up BASIC execution.

You may return to EDTASM/ASPTCH after using any of the above options. EDTASM's text buffer stays intact for all but the last option. Usually, when options are added to a program, available memory is reduced. However, ASPTCH points EDTASM's references to copied ROM routines back to Level II ROM. By extending low memory usage, replacing copied ROM routines with ASPTCH code, and tightening up EDTASM's memory usage, the text buffer size is actually increased if no memory is reserved.

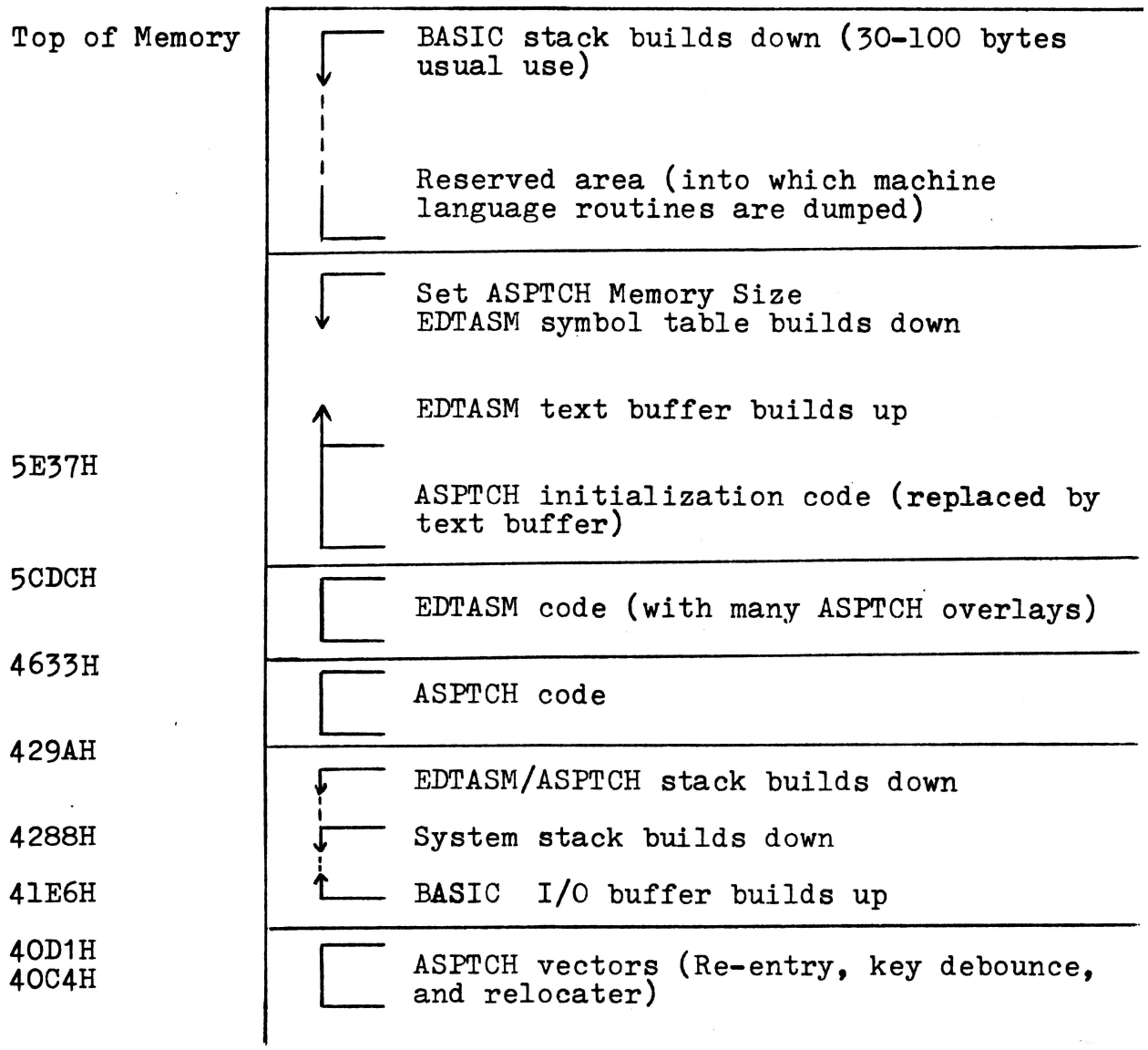
II. Loading Procedure

1. Load EDTASM but do not initialize.
2. Load ASPTCH.
3. Initialize with / [ENTER] .

III. ASPTCH Memory Size

Level II BASIC allows one to reserve high memory for machine

language programs (with automatic default to top of memory). ASPTCH adds this feature to EDTASM. Answer the ASPTCH Memory Size prompt with the beginning of the area you wish to reserve. Enter the address in Decimal or Hex (follow Hex entries with an "H"). The very top of memory is used by the Level II BASIC stack when the R command or reset button is used. Minimum usage is about 30 bytes (typing SYSTEM [ENTER] / [ENTER]). Maximum usage depends upon the complexity (levels of parentheses) of your BASIC command statement. 100 bytes is sufficient for most uses. Therefore, when reserving memory, it is safest to allow for 100 bytes at the very top of memory in addition to the area needed for machine language dumps. However, if you remember to redump your program after every use of the R command or reset button, you may run to the top of memory. The following memory map should serve to clarify:



IV. Entry to ASPTCH Command Mode

Enter any illegal command (X for example) while in EDTASM command mode.

V. ASPTCH Commands (Any non-command will return execution to EDTASM command mode.)

A ASPTCH reserved area set. Reset ASPTCH memory size (thus reset text buffer size). Enter in Hex or Dec the beginning of your reserved area (ENTER key only does not default to the top of memory here).

D Dump prompt control. Flip-flops Dump prompt from "DUMP TO CASSETTE" to "DUMP TO MEMORY". When set to "DUMP TO CASSETTE" machine language programs may be saved on tape as usual. When prompt is set to "DUMP TO MEMORY" assembled programs will be dumped directly into the area designated by the ORG statement(s) when the "READY CASSETTE" prompt is answered with [ENTER].

E Execute program that was dumped directly into memory. Execution starts at the entry point designated by the end statement.

C Convert and display contents of memory locations. Output format: Hex value, Decimal value, Hex contents of memory location, Decimal contents, and printable ASCII character (NP if graphics or not printable).

Sub Commands:

I Input address (or value to be converted). Converts value, then displays in output format on the next line.

↑ Display next higher memory location (hold for repeat).

↓ Display next lower memory location (hold for repeat).

R. Ready prompt (same as R command below). Used here for access to POKE command for memory modification.

Any other key causes return to EDTASM command mode.

R Ready prompt (Level II BASIC command mode). Use BASIC commands that do not use variables. Decimal calculator

type functions would be the most common usage (EX. PRINT(32+27)/5.6). Return to EDTASM by typing SYSTEM [ENTER] / [ENTER] or SYSTEM [ENTER] /16588 [ENTER].

S System prompt (Level II Monitor Mode). Allows jump to any location in memory or load of a system tape. Return to EDTASM with / [ENTER] or if you load a system tape, answer prompt with /16588 [ENTER].

M Memory size (sequence that leads to the Level II memory size prompt). First you are asked to enter the beginning of an area that you wish to reserve. This may be for a program already dumped there, or for a system tape that you intend to load after your return to BASIC. If you do not need to reserve any extra memory, default with [ENTER] key only (set to top of memory).

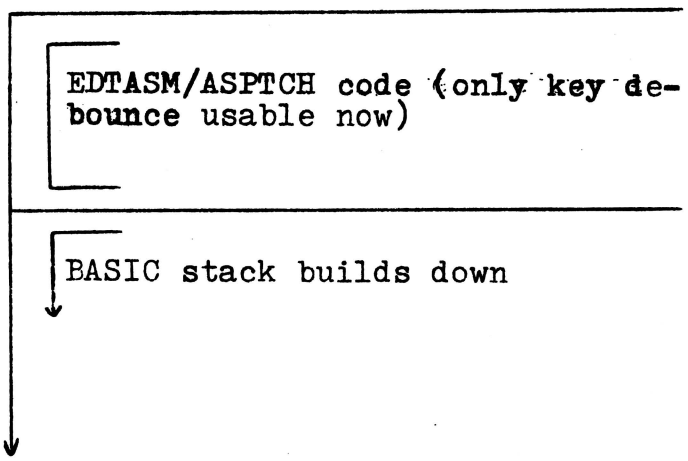
The next prompt will be the decimal number with which you must answer the BASIC memory size prompt when it appears. Memorize that number. Now hit any key (use break key if expansion interface is connected). The screen will go blank and the BASIC memory size prompt will appear. Answer that prompt with the memorized number. Now you are in fully programmable BASIC with EDTASM/ASPTCH and optionally, the extra area protected in high memory.

You may re-initialize the key debounce routine by answering the System prompt with /16585 [ENTER]. This will prevent key bounce and speed up BASIC execution.

You may return to EDTASM/ASPTCH by answering the system prompt with /16588. The text buffer will be destroyed.

The following two memory maps show 1) memory usage after returning to BASIC with no memory protected, and 2) memory usage with extra memory protected:

1) Top of Memory



Decimal address memorized and entered as answer to BASIC's Memory size prompt.

40D1H 40C4H	ASPTCH vectors remain intact. A jump to debounce initialization vector initialized key debounce and a jump to re-entry vector moves EDTASM/ASPTCH code back to original position.
2) Top of Memory	Reserved area (for programs dumped by ASPTCH or system programs to be loaded later).
Address entered immediately after M command	EDTASM/ASPTCH code (only key debounce usable now).
Decimal address memorized and entered as answer to BASIC's memory size prompt	BASIC stack builds down ↓
40D1H 40C4H	ASPTCH vectors

VI. Use of the Reset Button

The expansion interface forces a return to the power-up sequence when the reset button is pressed. This wipes out some of the ASPTCH code making reloading necessary. Therefore, if you use the expansion interface, you may not use the reset button except when you are in fully programmable BASIC with EDTASM/ASPTCH protected in high memory. When the memory size prompt appears, you must answer with the same decimal number you did before.

Since certain peripherals that attach directly to the E/I edge card have varying effect on the use of the reset button, you should consult the literature accompanying peripheral equipment.

If you do not have the expansion interface (or turn it off), you may use the reset button except when you are in EDTASM. The only time you will be tempted to use the reset button while in EDTASM is when you are trying to load a source tape and the machine "hangs up". You can usually get

out of the hang up by taking out the cassette you are trying to load and putting in the beginning of an object tape (system tape). This will usually cause a "BAD PARAMETERS" error after a few seconds. After that, however, you may have garbage in the text buffer (try P#:*) that will not delete with D#:*. To cure this, enter ASPTCH command mode, use S command , then type /18058 [ENTER] to re-initialize EDTASM. This may sound like a lot of trouble, but is is much better than having to reload EDTASM and ASPTCH.

After using the reset button, return to EDTASM by answering the System prompt with / [ENTER] or /16588.

VII. EDTASM "BUFFER FULL" Error

The "BUFFER FULL" error has been replaced with a jump to ASPTCH command mode. If you find yourself in ASPTCH unexpectedly, look at the "T-BUFFER BYTES LEFT" prompt. If this reads 00000, then the text buffer is full. You may use the ASPTCH A command to increase the text buffer size if it is not already set to the top of memory. (The same technique will work with the "SYMBOL TABLE OVERFLOW" error.)

VIII. Programming Procedure

1. Load EDTASM and ASPTCH.
2. Reserve area at the top of memory.
3. Write or load program as usual with EDTASM.
    a) ORG it so that it is contained within your reserved area.
    b) Make sure the last instruction executed will cause a return or jump to 16588.
    c) Make sure the END statement has the entry point.
4. Enter ASPTCH command mode.
5. Check Dump prompt. If it reads "DUMP TO CASSETTE", use the D command to change it to "DUMP TO MEMORY".
6. Assemble as usual with EDTASM A command.
7. Answer the "READY CASSETTE" prompt with the enter key. Do not turn on the cassette player.
8. After the * appears, the program has been dumped. Enter ASPTCH command mode.
9. Answer prompt with E command.
10. The program will execute and return to EDTASM. You may repeat execution as many times as desired with ASPTCH E command.

IX. Important Addresses

Combined load areas for EDTASM and ASPTCH:
40C4H-40D1H
429AH-5E37H
ASPTCH entry and EDTASM reset: 18058

ASPTCH/EDTASM re-entry: 16588
Key debounce initialization: 16585
Beginning of EDTASM text buffer: 5CDCH
ASPTCH input subroutine: 4523H
    Range: Hex 0000H-FFFFH
           Dec 00000-65529
ASPTCH Hex output subroutine: 4599H
    Range: 0000H-FFFFH
ASPTCH Dec output subroutine: 457CH
    Range: 00000-65535

The three ASPTCH subroutines use the HL register pair for I/O. These subroutines can be very useful for communicating with your machine language programs. If you need to input a value from the keyboard into your program, simply call the input subroutine. It accepts Decimal and Hex input and places the value in HL. If you need to display a value on the screen, put it in HL and call either the Hex or Dec output subroutine. You should save AF, BC, and DE for all of the subroutines.

X. Step by Step Example

The following example will show you how to use ASPTCH, keystroke by keystroke. You should refer to previous general discussions and the EDTASM manual so that you understand exactly what you are doing. The enter key is designated by [ENTER].

1.	Load EDTASM.	Ready Cassette Type SYSTEM [ENTER] EDTASM [ENTER]
2.	Load ASPTCH	Ready Cassette Type ASPTCH [ENTER] Type / [ENTER]
3.	Initialize. The ASPTCH initialization prompt will appear. Notice that the default value is set to the top of memory.	
4.	Reserve memory If you have more than 16K, this will be a gross waste of memory, but the purpose of this example will still be served. The prompt will show (top of memory-7E00H) in Dec.	Type 7E00H [ENTER]
5.	Continue You are now in EDTASM command mode. This is designated by the * prompt.	Hit any key

6. Enter program.
   This short program prints a "Z" on the screen at location 3E00H.

Type I [ENTER]
→ ORG → 7E00H [ENTER]
Entry LD → HL, 3E00H [ENTER]
→ LD → (HL), 'Z' [ENTER]
→ RET [ENTER]
→ END → ENTRY [ENTER]
(BREAK)

7. Enter ASPTCH command mode.
8. Change Dump prompt from "DUMP TO CASSETTE" to "DUMP TO MEMORY". Notice that ASPTCH commands do not require the enter key.

Type X [ENTER]
Type D

9. Assemble program
10. Dump it into memory
11. Enter ASPTCH command mode
12. Execute the program.
    Notice that the program prints a "Z" on the screen, then returns to EDTASM. Next we will demonstrate the use of the C command.

Type A [ENTER]
Hit [ENTER]
Type X [ENTER]
Type E

13. Enter ASPTCH command mode.
14. Enter C sub-command mode.
15. Input sub-command.
    Notice the →. This indicates that ASPTCH's input subroutine is expecting input. Input may be in Hex or Dec.

Type X [ENTER]
Type C
Type I

16. Input start of memory examination area.
    Conversion takes place on the same line as entry. The next line is in the output format. (see C command).

Type 7E00H [ENTER]

17. Examine next few memory locations.
    This is the object code of our program. For demonstration purposes, we will modify it directly in memory. See the character "Z" in location 7E04H? We will change it to a "Y" (ASCII 89).

Hold down ↑ until about 6 locations are displayed.

18. Enter Level II BASIC command mode.
19. Change location with POKE command.
    Notice that we must use decimal numbers for the BASIC POKE command. That is why the decimal conversions are there for you.

Type R
Type POKE 32260,89 [ENTER]

20. Return to EDTASM

Type SYSTEM [ENTER]
/ [ENTER]

21. Enter ASPTCH command mode	Type X [ENTER]
22. Execute program. Notice a "Y" was printed rather than a "Z". Our source code in the text buffer still has a "Z". We will re-dump it and execute.	Type E
23. Assemble.	Type A [ENTER]
24. Dump.	Hit [ENTER]
25. Enter ASPTCH command mode.	Type X [ENTER]
26. Execute and return to EDTASM. Notice that the "Z" was printed again. When we re-assembled and dumped, we over-wrote the whole program, including the character we modified. Next we will save EDTASM/ ASPTCH and our program and return to fully programmable BASIC. We can then execute the program from BASIC with the USR command. However, without the expansion interface, we have to convert the two halves of the entry address into decimal so we can POKE them into the USR locations (see Level II manual). 7E00H breaks down into 7EH and 00H. The 00H conversion is not much of a problem, but we will use the C command to convert 7EH.	Type E
27. Enter ASPTCH command mode.	Type X [ENTER]
28. Enter C sub-command mode.	Type C
29. Input value to be converted. Memorize or write down decimal equivalent. Ignore the rest of the display. We are not interested in the contents of location 7EH.	Type I 7EH [ENTER]
30. Leave C sub-command mode.	Hit any key other than I, ↑, ↓, or R.
31. Enter ASPTCH command mode.	Type X [ENTER]
32. Enter return to BASIC sequence.	Type M
33. Enter beginning of reserved area. Memorize the given decimal number.	Type 7E00H [ENTER]
34. Back to BASIC.	Hit any key (use break key if expansion interface is on)

35. Protect high memory.
    Now we are in fully programmable BASIC. At this point we can load and execute basic programs (or system programs if they do not interfere with the protected area).
    Type 25533 ENTER
36. Re-initialize key debounce.
    Type SYSTEM ENTER
    /16585 ENTER
37. Enter BASIC test program.
    Type 10 CLS:POKE 16526,0:
    POKE 16527,126 ENTER
    20 X%=USR(X%) ENTER
38. Execute.
    Type RUN ENTER
    Notice that the "Z" was printed, indicating that the program is intact.
39. Return to EDTASM.
    Type SYSTEM ENTER
    /16588 ENTER
    Notice the BASIC program and the text buffer of EDTASM have been destroyed, but the machine language program is still intact.
40. Enter ASPTCH command mode.
    Type X ENTER
41. Execute.
    Type E
    Program is still there.

Remember to save source listings and BASIC programs on tape when going back and forth between EDTASM and fully programmable BASIC. In fact, it is wise to make periodic source tapes while debugging, in case of a system crash.

SYSTPE

by
James Fuller Williams

SYSTPE is a source tape that when loaded into EDTASM/ ASPTCH 2.0, assembled, dumped, and executed will allow you to copy any number of distinct memory areas and combine them into one System format tape. It also allows you to cause your program to execute immediately after loading, (without keyboard initialization) if desired.

Loading

SYSTPE must be loaded into EDTASM/ASPTCH 2.0

1. Load EDTASM
2. Load ASPTCH
3. Initialize
4. Reserve memory. (SYSTPE is originally set to 7E00H, but this can be changed if desired.)
5. Load SYSTPE (use EDTASM "L" command)

ORGING

SYSTPE is set for loading near the top of 16K. However, if you have more than 16K, or 7E00H is not a convenient place for SYSTPE, then simply change the ORG statement to an address that suits your needs.

Making a System tape of SYSTPE

If you desire, you may make a system tape of SYSTPE for separate loading by assembling and dumping on tape as usual. However, the first half of SYSTPE uses ASPTCH routines, so EDTASM/ ASPTCH must be in memory (in their original position) for the initialization part of SYSTPE to work.

Dump to Memory

Set ASPTCH Dump prompt to "DUMP TO MEMORY", assemble program and dump.

Execution

Execute with ASPTCH E command, or a SYSTEM command jump to entry point.

File name

Enter up to 6 letters for the file name with which you wish this system tape to load.

Immediate Execution

Sometimes it is desirable for programs to execute immediately

after loading, without manual initialization (typing / ENTER). (For example, if you had a fancy display that was loaded directly from tape, the prompt could mess it up.) SYSTPE allows you to utilize this function. Simply include as one of your memory areas to be copied, 41E2H-41E4H. This will create one problem: the BASIC 'SYSTEM' command will cause an immediate jump to your entry point every time you try to use it. To fix it from BASIC, type POKE16866,201 ENTER , or better yet, have your machine language program fix it first thing (LD HL, 41E2H LD (HL),201).

Number of ORGS

SYSTPE asks for the number of distinct memory areas you wish to copy. (For example, if you wanted to copy the screen (3C00H-3FFFH) and 6000H-7000H, you would answer 2. If you wanted to have immediate execution, you would answer 3.)

Start Address-End Address

SYSTPE now will loop for n times, asking for the start address then the end address for each area you wish to copy. (n=number of ORGS). Answer in Hex or Dec.

Entry Address

After the last End Address is entered, SYSTPE asks for the Entry Address. Enter the entry point in Hex and Dec.

Re-entry

SYSTPE now gives the re-entry address and goes into Level II BASIC monitor mode (System prompt). The programs or data you wish to copy do not even have to be in memory yet. If they are in memory, you may proceed directly to making the tape. If not, return to EDTASM with / ENTER . From ASPTCH you can dump your data or programs, or move EDTASM/ASPTCH and return to BASIC. If you go back to BASIC, be sure to protect memory starting at the re-entry point or less.

Making the Tape

After the data or programs are loaded into memory, you must ready the cassette recorder for recording and jump to the re-entry point. The easiest way to do this is answer the System prompt with / re-entry point . You can also use the BASIC USR command if you wish. You may repeat the jump to re-entry as many times as desired for as many copies as desired.

Step-by-Step Example

The following is a keystroke by keystroke example of how to use SYSTPE to make a copy of EDTASM and ASPTCH for a single loading. If you wish to use the program for something other than this, it is important that you try to understand exactly why and what you are doing.

1.	Load EDTASM	Ready cassette Type SYSTEM [ENTER] EDTASM [ENTER]
2.	Load ASPTCH	Ready cassette Type ASPTCH [ENTER]
3.	Initialize	Type / [ENTER]
4.	Reserve memory	Type 7E00H [ENTER]
5.	Continue	Hit any key
6.	Load SYSTPE	Ready cassette Type L [ENTER] [ENTER]
7.	Enter ASPTCH command mode	Type X [ENTER]
8.	Change prompt to "DUMP TO MEMORY"	Type D
9.	Assemble	Type A [ENTER]
10.	Dump	Hit ENTER
11.	Enter ASPTCH command mode	Type X [ENTER]
12.	Execute	Type E
13.	Answer File Name prompt	Type EDTASM [ENTER]
14.	Answer ORGS prompt	Type 2 ENTER
15.	Answer Start Address prompt	Type 40C4H [ENTER]
16.	Answer End Address prompt	Type 40D1H [ENTER]
17.	Answer Start Address prompt	Type 429AH [ENTER]
18.	Answer End Address prompt	Type 5E37H [ENTER]
19.	Answer Entry prompt	Type 18058 [ENTER]

Now initialization is complete. We must re-load ASPTCH because the text buffer destroys the initialization code. EDTASM is alright.

20.	Load ASPTCH After loading, we do <u>not</u> initialize ASPTCH. We want to copy memory exactly as it is now, before initialization.	Ready cassette Type ASPTCH [ENTER]
21.	Make Tape Tape recorder will turn on and make a tape of combined EDTASM/ASPTCH 2.0. It will load under file name EDTASM. To make more copies, repeat step 21.	Ready cassette Type /32370 [ENTER]

BREAK

by
James Fuller Williams

BREAK is a source tape, for use with EDTASM/ASPTCH 2.0. After dumping it into memory, it can be called by a program being debugged to stop execution and display flag conditions (a binary representation of flag registers) and register contents in Hex. After a break, you have the option of continuing execution or returning to EDTASM. If execution is continued, all registers are reset with their original contents.

Setting and removing break points is simply a matter of adding or deleting lines that contain a call to the entry address of the BREAK subroutine. After break points are added or deleted from a program being debugged, it is assembled, dumped and executed. All of this can be done very easily with ASPTCH.

Be sure to give some consideration where to ORG and dump the BREAK subroutine. It must be in the reserved area and it must not conflict with the program you are debugging. Also remember to keep the end of the BREAK subroutine at least 100 bytes from the top of memory so that the ASPTCH R command will not wipe it out.

If you use this program often, you will want to add it as a 3rd ORG to your combined EDTASM/ASPTCH tape (see instructions below). It would then load automatically every time you loaded EDTASM/ASPTCH.

The following is a step-by-step example of how to use BREAK. For demonstration we will use the SYSTPE program.

1. Load EDTASM and ASPTCH and initialize.
2. Reserve memory at 7D00H
3. Load the BREAK program with EDTASM's "L" command.
4. Change BREAK's ORG statement to 7D00H
   We will load BREAK beneath SYSTPE.
5. Enter ASPTCH command mode and set the Dump prompt to "DUMP TO MEMORY" with the D command.
6. Assemble and dump BREAK into memory.
7. Delete EDTASM's text buffer (D#:*)
8. Load SYSTPE program with EDTASM's "L" command.
9. Now choose any spot in the program and insert the line CALL 7D00H.
10. Assemble and dump it into memory.
11. Execute with ASPTCH E command.

The program will execute as usual until the break point is reached. At that point, flags and registers are displayed, and a prompt asks what you want to do next. If you hit C, execution will continue from that point. If you set your break point in a loop in the program, it will break every time it is encountered. If you hit E, you will return to EDTASM.

12. Return to EDTASM and add another break point and remove the old one, if desired. You may have as many break points in a program as you wish. Use the EDTASM "F" command to help you find break points. Go back to step 10.

How to make BREAK load automatically with EDTASM/ASPTCH 2.0.

If you combine BREAK with EDTASM/ASPTCH, it will add only a second or so to the total loading time. It can be ignored and written over if not needed. If you plan to use it, all you need to do is remember where you put it, reserve enough memory for it and the program with which you will be working and be careful that your expanding program does not "eat" BREAK. The original ORG for BREAK is 7EBOH. This puts the end of it about 100 bytes from the top of 16K. If you have more than 16K, you should put it in the same relationship to the top of memory. (BEBOH for 32K and FEBOH for 48K).

1. Load EDTASM/ASPTCH and initialize.
2. Reserve memory (7DOOH will work).
3. Load BREAK with the EDTASM "L" command.
4. Enter ASPTCH command mode and change Dump prompt to "DUMP TO MEMORY".
5. Change ORG statement if you have more than 16K (see suggested ORG's above).
6. Determine the Start and End points of BREAK
   a. Assemble with EDTASM command A/NS
   b. Make a note of the location of the last byte in the assembly (DEFB 0).
   c. The Start address (Entry point) is to the left of the END statement. Write it down.
7. Dump it into memory hit ENTER.
8. Delete the text buffer (D#:*) and load SYSTPE.
9. If you have 16K, SYSTPE will conflict with our dumped BREAK. Therefore change the ORG statement to 7DOOH. (If you have more than 16K, you should have dumped the BREAK subroutine at a far higher ORG, eliminating the conflict over the same memory area, thus making this ORG change unnecessary.)
10. Assemble, dump and execute.

10a. Answer filename prompt with EDTASM

11. Answer the SYSTPE NUMBER OR ORGS prompt with 3.
12. Answer the 3 sets of START-END ADDRESS prompts with:
    40C4H
    40D1H
    429AH
    5E37H
    BREAK Start address
    BREAK End address
13. Answer ENTRY ADDRESS with 18058.
14. Make a note of the re-entry address.
15. Load ASPTCH, but do not initialize.
16. Ready the Cassette player for recording.
17. Jump to the re-entry address (answer System prompt with /re-entry address). Repeat step 17 for as many copies as desired.