TX-0 COMPUTER
Massachusetts Institute of Technology
Cambridge 39, Massachusetts

M-5001-23

FLIT--Flexowriter Interrogation Tape: A Symbolic Utility Program for TX-0

July 25, 1960

TABLE OF CONTENTS

FLIT--Flexowriter Interrogation Tape: A Symbolic Utility Program for TX-0

INTRODUCTION

A new utility program, FLIT, has been prepared for the TX-0 which allows the user to test a program in terms of symbolic addresses. FLIT is designed to replace UT-3 and has taken advantage of the recent increase in memory size to provide the user with many new features as well as improved versions of the old ones. The largest single advantage is that the operator may now type in and out of memory using, whenever he desires, any of the three character tags in his symbol table as well as any additional ones he may wish to define.

For clarity when typing examples are given herein, the typing done by the user of FLIT is underlined. Also, when needed the following symbols are assigned to the invisible flexo characters:

carriage return	↲
tabulation	⇥
space	▨
backspace	←
delete	$\boxed{\text{delete}}$

Color changes will be indicated explicitly.

USING FLIT

A. Preparations

1. FLIT occupies the end of memory from approximately register 13300 to register 17777 including the standard end of memory input routine. Obviously the program to be tested by FLIT may not occupy or use any of these registers.

2. To use FLIT turn on the flexowriter with the punch switch on and depress the start read button. Place FLIT in the PETR and press read in. Turn on the type in switch. When FLIT has finished reading in, the user may type immediately; it is not necessary to press restart.

3. To inform FLIT of the meaning of the symbols used in your program, place your binary symbol tape, which was prepared by MACRO SYMBOL PUNCH, in the PETR and type

table ↵

FLIT will then read in your symbol table 100 registers at a time. After this FLIT is ready for use and will be able to interpret constants and instructions typed either symbolically or numerically or both.

It is important to recognize that after the symbol table has been read in, FLIT occupies more memory than that implied in paragraph 1. How much more naturally depends upon the length of the symbol table. Thus for convenience a provision has been made such that symbol F is always assigned a value equal to the address of the last register not occupied by FLIT. To determine this address simply type

F =

and FLIT will immediately type out the address of the last free register.

4. To read in the binary tape containing your program or data place the tape in the PETR and type

read ↵

Then typing

```
                    begin ↲
```

will transfer control to your program at the address indicated on the last binary program tape read. To transfer control to a specific register such as loc, type

```
                    begin loc ↲
```

Additional features of the pseudo instructions table, read and begin are given in section J of this memo.

5. To return control to FLIT after using a begin command, transfer to register 14000.

B. Examination and Modification of Stored Information

1. Opening a register--In using FLIT a fundamemtal idea is that of opening a register so that its constants may be examined and/or changed. This is accomplished by typing the address of the register to be opened, either symbolically or as an absolute constant, followed by a vertical bar. For example:

```
reg+2|

 or

2467|
```

When this is done FLIT will immediately print a tabulation, then the contents of the register followed by another tabulation. Continuing the example:

$\underline{\text{reg+2}|}\longrightarrow\!|$ add loc+3$\longrightarrow\!|$

2. Modifying and closing a register--Once a register has been opened in this manner its contents may be modified, if desired, by typing the change either symbolically or as a constant. For example:

$\underline{\text{reg+2}|}\longrightarrow\!|$ add loc+3$\longrightarrow\!|$ $\underline{\text{add loc+5}}$

However, the modification is not placed in memory until the user types one of three terminating characters--tabulation, carriage return, or backspace. The effect of each of these characters is given in the following table.

Terminating Character	Action
$\longrightarrow\!|$	Moves carriage to next tabulation stop and modifies the contents of the open register if a modification has been typed. The register is left open.
↲	Returns carriage and modifies the contents of the open register if a modification has been typed. The register becomes closed.
←	Same action as carriage return except in addition the next register in sequence is opened automatically. If no register is open, this character may still be used to open the next register in sequence.

Once a particular register has been closed by use of either the carriage return or the backspace, further modification of that register is impossible until it is opened again.

During the process of register modification, it is often desirable to be able to punch the new contents. This may be accomplished at any time that a register is open simply by typing the upper case character

P

Upon subsequent typing of any of the three terminating characters listed on the previous page, the current contents of the register in question will be punched as a one register block including check sum. The following example illustrates the use of the punch indicator, backspace and carriage return.

```
reg|———→| cla ———→| P← ↲
reg+1———→| add t ——→| add ti P ↲
```

When the backspace was typed, FLIT punched the contents of reg as a one word block and opened the following register. Then the contents of reg + 1 were modified and punched.

3. Modification color code--FLIT is arranged so that all typing done by the user while a register is open will appear in red. This feature allows all modifications made to be easily spotted in a long list of FLIT typings, since no modifications of memory can result unless the user types while a register is open. The color feature also warns against possible accidental modification of a register.

4. Additional interpretation of register contents--If, while a register is open, any one of the following characters is typed, the contents of that register will be reprinted in the indicated manner.

Interpretation Characters	
Character	Interpretation Type as:
=	a constant*
I	an instruction
D	a signed decimal constant
O	a signed octal constant
o } super-	an unsigned decimal constant
8 } script	an unsigned octal constant

To illustrate the use of these interpretation characters, consider the following example:

```
reg+100|—→| sto 144→| =144———→| D+100——→| ←⤶
reg+101|—→| 777767—→| O-10———→| D-8———→| °262135⤶
———————→| ⁸777767→|
```

Here we see that register reg+100 contains the instruction sto 144 which is 144 as an unsigned octal number and + 100 as a signed decimal number. The contents of the following register was initially typed as the unsigned octal constant 777767 because no reasonable interpretation as an instruction was possible.** This number is -8 in signed decimal and 262135 = $[2^{18} - 1) - 8_{10}]$ in unsigned decimal.

* Although this character will usually evoke interpretation as an unsigned octal constant, it will be seen later that this is not always so.

** See appendix I on operate class commands.

5. Examination and modification of deferred register--Once an instruction has been typed out by FLIT, it is frequently desired to know the contents of the register address by the instruction either as an instruction or as a constant. The following control characters provide this facility:

Deferred Register Examination	
Character	**Meaning**
(	Type the contents of the register addressed by the preceding word as a constant.
/	Type the contents of the register addressed by the preceding word as an instruction.

If a register has been opened, and the deferred register has been examined by typing one of the above characters, its contents may be modified by typing the desired new contents and one of the terminating characters. If the punch indicator is typed before the terminating character, the final contents of the deferred register will be punched as a one word block. These features are illustrated by the following example:

```
reg+40 ⇥ add loc ⇥ add 500P⇥  (0      ⇥ add tabP←↲
reg+41 ⇥ sto t   ⇥ ↲
reg+40 ⇥ add 500 ⇥ /add tab⇥ =200205 ⇥ (1025↲
       ⇥
```

Here the user has modified register reg + 40 and then has deferred to examine the constant in register 500. He has changed this to add tab and has terminated with a backspace which has caused location reg + 41 to be opened. Both modifications have been punched by use of the punch indicator. The third line verifies the changes and demonstrates the reinterpretation of the word in register 500 as a constant. It also shows how the defer characters can be repeated to show that register tab (205) contains the octal constant 1025.

6. Typing comments--The character

|

when used alone informs FLIT that a comment is about to be typed. All comments will be ignored by FLIT. Comments will be terminated by a carriage return or tabulation.

C. Typing Instructions, Constants, and Locations

1. Instructions, constants, and locations, which collectively may be referred to as words, may be typed by the user at any time using any combination of numbers and/or defined three character symbols separated by appropriate connectives such as plus and minus signs. These connectives are listed in the following table along with their meanings.

Connectives	Meanings
▨	a space means addition
+	a plus means addition
-	a minus means the complement of what follows will be added
U	unite (logical sum) (inclusive or)
A	intersect (logical product) (and)
S	distinguish (partial add) (exclusive or)
X	multiply as integers NOTE: Multiply as integers obeys all the usual rules of sign except in this case: (-0)(-0) = -0

When these connectives appear together in a single word, all of the unions, intersections, partial additions, and multiplications are done first, after which the additions and additions of complements (subtractions) follow.
For example:

Typing	Yields
add 10	200010
10X10X10	1000
-10X-10X-10	-1000
1X-2+-1X4	-6*
070070A000777S000007	77*
claUlro	cal
ladS10	lpd
100+10-110	-0
aSb+b	b carried into a*

D. <u>Break point testing</u>

In testing a large program it is frequently convenient to interrupt the computation so that partial results may be examined or the state of the program determined. Two psuedo instructions have been included in FLIT which simplify this technique. Typing

<u>break bpt 2</u>

causes FLIT to set up a <u>break point</u> at location bpt in the user's program. If the user transfers control to his program, and the instruction in register bpt is reached, the computation will cease and FLIT will print the contents of the

---

* Order of operation matters in these three cases. Otherwise the results might be -12, 70, and aS2b respectively.

accumulator and live register as they are before execution of the order in register bpt.

For example:

```
begin beg ↲
bpt|       ac  77777   lr  15
```

At this point the user may examine registers and make modifications as he pleases. Then, typing

```
proceed ↲
```

causes FLIT to continue operation of the user's program as though the break point had not been inserted.*

As many as four separate break points may be set up in the user's program by typing

```
break  bp1, bp2, bp3, bp4 ↲
```

Each time a break psuedo instruction is typed, all previously specified break points are reset to normal. In particular, typing simply

```
break ↲
```

will reset all break points.

---

* If the user wishes to proceed with a modified accumulator and/or live register he must use the begin instruction as described in section J7 of this memo.

The pseudoinstruction proceed may be used even though the last breakpoint encountered has been reset. For instance, the following sequence of testing might be carried out:

```
break bp1 ↵
begin beg ↵
bp1|——————→| ac 215677——————————→| lr 355670↵
loc|——————→| add t————→| add t1 ↵
break bp2 ↵
proceed ↵
bp2|——————→| ac   0————→| lr 1001↵
```

Control is returned to FLIT when the first breakpoint is reached. The user then modifies the instruction in register loc and changes the location of the breakpoint before giving the proceed command.

CAUTION The locations selected as breakpoints must not be registers whose contents are modified by the program under test, since FLIT transplants their contents and substitutes specific transfer commands.

E. Defining New Three Character Symbols

1. There are three different ways of adding to the list of three character symbolic address recognized by FLIT.

a. A binary symbol tape may be prepared by MACRO SYMBOL PUNCH and entered into FLIT as described in section A-3 of this memo.

b. Symbols may be defined directly by means of the equivalence sign as in the following example:

```
        sym:2475↲
```

These symbols may contain one, two, or three letters or digits at least one of which must be a letter. Symbols may be redefined in this manner unless they are in FLIT's permanent vocabulary as given in Appendix II. Of course, the previous definitions are completely lost.

c. Symbols may be defined while a register is open simply by typing the symbol followed by a comma. For example:

```
106 ——→| hlt ——→| a, add 203 ————→| ∠hlt ——→| b,+1 ↲
 a| ——→| add b —→| ( 1 ——————————→|
```

Here, location 106 has been assigned the symbolic address a, and add instruction inserted. The location addressed has been filled with the constant +1 and given the name b. Symbols may not be redefined in this manner. The second line verifies the change.

F. Characters with Special Values

The characters in the following table have been given the special meanings indicated:

Character	Meaning
F	The last address not occupied by FLIT
W	The last word typed either by the user or by FLIT
L	The address of the last register opened

As can be seen by the nature of the meanings of these characters, their actual numerical values are constantly being reevaluated during the use of FLIT. The value of the first is reduced by two for each new three letter symbol defined. (See section A-3 of this memo.) The second special character finds use mainly in revising a word when its present value is substantially as desired. The third is used in reopening a register accidentally closed or to refer to registers near the one presently open.

As an example of the use of these characters consider the following:

```
F = 13162↲
W|———————→| hlt————→| -1 ↲
L-1|—————→| hlt ————→+1 ↲
```

Here the user desires to place two of his constants at the very end of available memory. Note that he had stored halts throughout unused memory before starting troubleshooting activities.

G. <u>Evaluation of Words</u>

1. Often it is desirable to be able to evaluate a word that is to be used in a program without actually affecting memory. This may be done at any time without opening a register by simply typing the word to be evaluated followed by the appropriate interpretation character (See section B-4). When this has been accomplished, FLIT will automatically type out the appropriate interpretation of the word followed by a carriage return. Additional interpretations of the same word may then be obtained by typing

W

followed by another interpretation character. Consider the following example:

```
abc = 777777↲
W° 262143↲
WD -0
```

Here the user has determined that abc is equal to the unsigned octal constant 777777 which equals the unsigned decimal constant 262143 which equals the signed decimal constant -0.

The "interpret deferred register" characters may also be used when a register is not open. This permits examination of a register without destroying FLIT's knowledge of location. For instance

```
reg|————→| llr con——————→|↲
t(117↲
loc/add x ↲
⟵ reg+1|————————→| slr t——→|
```

Of course, the contents of a register interrogated by this procedure cannot be altered, since no register is open at that time.

H. Control of Printout Modes

1. Although it has been assumed so far that FLIT normally prints out the contents of registers as instructions with symbolic addresses and normally interprets constants as unsigned octal numbers, a provision has been made to alter this state of affairs with a considerable degree of flexibility.

2. Print mode control -- By typing one of three pseudoinstructions *
as listed in the following table, the normal mode of printout may be controlled.

Pseudoinstructions	Resulting Action
instructions ↲	set normal print mode to instructions
constants ↲	set normal print mode to constants
constant a,b ↲	set normal print mode to constants for locations a through b inclusive and to instructions elsewhere

3. Instruction print control -- By typing one of the pseudoinstructions as listed in the following table, the normal mode for the printout of instruction addresses and locations may be controlled.

Pseudoinstruction	Resulting Action
absolute ↲	print all addresses and locations as octal numbers
symbolic ↲	print all addresses and locations symbolically **
symbolic a,b ↲	print all addresses and locations whose values lie in the range a through b inclusive symbolically and as octal numbers otherwise

* Pseudoinstructions must always be typed starting at the extreme left-hand margin. FLIT identifies pseudoinstructions by comparing the first three characters with a list, once four characters have been typed. Hence, it is only necessary to type the first four letters of a pseudoinstruction, and the fourth letter need not be correct.

** For a detailed discussion of the symbolic print mode see Appendix I.

4. Constant print control -- By typing one of the four pseudoinstructions as listed in the following table, the normal mode for the printout of constants may be controlled.

Pseudoinstructions	Resulting Action
octal ↲	all constants will be printed as octal numbers
decimal ↲	all constants will be printed as decimal numbers
unsigned ↲	all constants will be interpreted as unsigned 18 bit numbers
signed ↲	all constants will be interpreted as signed 17 bit numbers

Locations and addresses typed by FLIT will always be in octal regardless of the current constant print mode.

5. Utilization of printout modes -- The versatility of operation offered by the facilities mentioned above is self-evident; however, a few additional explanatory remarks are appropriate.

The usual mode of operation of FLIT will be that best suited to program troubleshooting. For this purpose it is most advisable to set the print mode control to instructions except in the range where the constants are located. This can be done very simply by typing

constants a,b ↲

Then the instruction print mode should be set for absolute except for within the range of memory that the program itself resides. This is done by typing

symbolic beg, end ↲

The constant print mode may then be selected as desired.

This arrangement allows the following conveniences. If a register containing a constant is opened, its contents will be typed as a constant without further thought on the part of the user. If an instruction not referring to the program in question is typed, this will be implied by the absolute address. This feature also avoids the typing of symbolic addresses with large numeric augmentation; for example

<u>reg+2</u>|——→| add end+2674————→|

is less meaningful than

<u>reg+2</u>|——→| add 3241————→|

I. <u>Input Radix Control</u>

The pseudoinstructions octal and decimal discussed in the preceding section also control the radix used by FLIT to interpret <u>all</u> numbers typed in by the user. The two characters given below may be used to force the interpretation in either octal or decimal regardless of the current printout radix.

<u>Character</u>	<u>Meaning</u>
.	Immediately preceding number typed by the user will be interpreted as decimal regardless of the current radix.
)	Immediately preceding number typed by the user will be interpreted as octal regardless of the current constant radix.

J. <u>Utility Subroutines</u>

Several special features have been incorporated in FLIT which allows the user to perform program testing more rapidly and intelligently. Some of these are similar to those which were available in UT-3, and some of them are new. These features are provided by the inclusion of a number of subroutines in the FLIT tape. Their purposes are described in the following paragraphs.

1. CLEAR -- The "clear" subroutine is used to clear portions of memory as follows:

clear ↵	clears available memory
clear a,b ↵	clears all memory from "a" to "b" inclusive
clear a,b,w ↵	stores the word "w" in all memory from "a" to "b" inclusive

Typing clear 0, F, hlt

will store halt instructions in all available memory registers.

2. PRINT -- The print subroutine prints the current contents of memory according to the print mode settings of section H in a four column horizontal format as follows:

print a,b ↵*	prints the contents of registers "a" through "b" inclusive such as

```
a|────────→| add abc→| sto abd→| cla──────→| add def↵
a+4|──────→| trn efg→| hlt─────→| +1───────→| -1↵
a+10|─────→| 2000───→| -2000───→| 77───────→|↵
```

3. WORDS and ADDRESS -- The "word" and "address" subroutine searches memory for given word or words with given addresses as follows:

* Assumes signed, constants a+6, b

word w	prints all registers containing the word w except those in FLIT.
word w,a,b	prints all registers containing the word s that lie in the range a to b inclusive.
word w,a,b,m	prints all registers containing words that agree with the word w where the mask m has ones, provided they lie in the range a to b inclusive.

address l ↲	
address l,a,b ↲	same as above except only bits 5-17 of l are compared with memory.
address l,a,b,m ↲	

4. SURPRISE -- The "surprise" subroutine compares the contents of memory with a binary tape and prints the registers which are different. The tape contents are printed in the opposits color. Surprise will not change the contents of memory * and cannot be used with tapes containing blocks longer than 100_8 registers.

surprise ↲	compare entire tape with memory and print discrepancies.
surprise a,b ↲	compare only for location in the range "a" to "b" inclusive.

5. PUNCH and other tape preparation subroutine -- If a punched tape is to be prepared by FLIT the following pseudoinstructions are used:

* As a result the following undesirable situation can arise. If the contents of a register are specified twice on the tape being compared, both words on tape will be compared with the same word in memory, and any disagreement will be typed by FLIT in both cases. If the second word on tape agrees with memory but the first does not, the printout will be misleading.

input ↲	punch a standard input routine occupying registers 17741 - 17777.
punch a,b ↲	punch the contents of registers a to b inclusive in blocks of 100_8 registers. (same as MACRO)
start 1 ↲	punch a start block of the form add 1.
start add 1 ↲	punch a start block of the form trn 1. (Automatic start)
feed ↲	punch three inches of blank tape.

6. Symbol table subroutine -- The following pseudoinstructions control the list of three character symbols in FLIT:

table ↲	add the symbols in a binary symbol table as prepared by MACRO SYMBOL PUNCH to FLIT's symbol table.
table a,b ↲	read in symbol table as above, but, of symbols with values tetween 0 and 17777, take only those in the range a to b inclusive.

Attempts to redefine symbols in FLIT's permanent vocabulary will be ignored. For other symbols the new definition will be substituted. Two additional registers are occupied by FLIT for each new symbol defined.

reset ↲	erases FLIT's symbol table leaving only the permanent vocabulary which is identical to MACRO's initial symbol list.

The "reset" subroutine causes more memory space to be made available to the user; however, this space will not be cleared.

7. Read in and testing subroutines -- While testing a program it is often desirable to be able to read in a new tape and enter the program at arbitrary points with desired quantities in the accumulator. The following pseudoinstructions provide these features.

read ↵	causes a binary tape to be read into memory. The tape must be punched in blocks of 100_8 registers. FLIT will not read over itself.
read a,b ↵	as above, but only words located between address a and b, inclusive, are stored in memory.
begin ↵	transfers control to the starting address of the last binary tape read in by means of the "read" pseudoinstruction with the accumulator and live registers set to zero. (C(AC) = 0, C(LR) = 0).
begin l ↵	transfers control to register l with C(AC) = 0.
begin l, ac ↵	transfers control to register l with C(AC) = ac, C(LR) = 0.
begin l,ac,lr ↵	transfers control to register l with C(AC) = ac, C(LR) = lr.

K. Error Indication and Correction

1. FLIT has five error alarms associated with its use. They are all typed by FLIT in red and have the following general meanings:

Identification	Meaning
error	An uncorrectable error has been made: perform the previous operation again.
X	The immediately preceding typed character is improperly used and has been ignored; type the correct character without starting over.
indef	The immediately preceding word contains an undefined three character symbol; take required action.
flit	The preceding operation has involved memory space currently occupied by FLIT, or, an attempt was made to redefine a symbol in FLIT's permanent vocabulary.
sumchk	A sum check error occurred in reading a binary program or symbol tape.

2. When the user of FLIT realizes that he has made a typing error, he may delete all that he has typed since the last carriage return or tabulation by pressing the delete button. For example:

$\underline{\text{address a } \boxed{\text{delete}}}$ xxxx↲

3. The alarm "flit" is also used to warn against an attempt to redefine a three letter symbol more than once by means of the comma. For example:

$\underline{\text{a:30↲}}$

31|———→| trn20——→| $\underline{\text{a}}$,flit——→|

APPENDIX

I. The Symbolic Printout Mode

When FLIT types instructions that are not operate commands in the symbolic mode, it types the operation code (add, sto llr; etc.) followed by a space followed by the largest three character symbol in its current symbol table which is less than or equal to the address, followed by a plus sign and the difference needed to make up the true address value. For example:

```
add a
add a+2
```

In the case of operate class commands, an instruction mnemonic will be typed if FLIT can find one in its vocabulary whose value agrees exactly with the word to be printed. Otherwise, it types opr n, where n is the difference between opr and the word to be printed. Thus:

```
600000Iopr↵
630000Ihlt↵
630001Iopr 30001↵
```

II. Resumé of Pseudoinstructions and Character Meanings

A. Pseudoinstructions:

*	instructions	type as instructions
	constants	type as constants
	constants a,b	type as instructions except between a and b
	absolute	type addresses as numbers
*	symbolic	type address symbolically
	symbolic a,b	type addresses as numbers except a and b
*	octal	interpret constants as octal
	decimal	interpret constants as decimal
*	unsigned	interpret constants as unsigned
	signed	interpret constants as signed
	clear	clear available memory
	clear a,b	clear from a to b
	clear a,b,w	insert w from a to b
	print a,b	print registers a to b horizontally
	word w	search for w
	word w,a,b	search for w from a to b
	word w,a,b,m	search for w from a to be masked by m
	address l	search for address l
	address l,a,b	search for address l from a to b
	address l,a,b,m	search for address l from a to b masked by m

* condition of FLIT when read in

<u>Pseudoinstructions cont'd</u>:

surprise	compare tape with memory
surprise a,b	compare tape from a to b
feed	feed three inches blank tape
input	punch input routine
punch a,b	punch memory from a to b
start l	punch start block
start add l	punch automatic start block
table	read symbol table tape
table a,b	read symbols if between a and b
reset	reset symbol table
read	read program
read a,b	read program between a and b
begin	start program
begin l	start program at l
begin l,ac	start program at l with ACC = ac
begin l,ac,lr	start at l with C(ACC) = ac and C(LR) = lr
break bp1, bp2, bp3, bp4	stop when breakpoint is reached
proceed	proceed from last breakpoint
break	erase all breakpoints

B. Characters:

→|	make modification if register is open
2	close register and make modification
←	close register, open next
=	equals as a constant
I	equals as an instruction
D	equals in signed decimal
Θ	equals in signed octal
o	equals in unsigned decimal
8	equals in unsigned octal
/	register referred to contains the instruction
(	register referred to contains the constant
▨	plus
+	plus
-	minus
U	unite (logical sum; inclusive or)
A	intersect (logical product; and)
S	distinguish (partial add; exclusive or)
X	times (integer multiply)
F	address of last free register
W	last word typed
L	last register opened
:	is now defined as
.	interpret as decimal
)	interpret as octal

|	open register specified; or allow comment
P	punch new contents of register last examined
$\boxed{\text{delete}}$	delete all typing since last tab or carriage return
,	separates pseudoinstruction arguments; or defines three letter symbol as present register address.