INTEL CORP. 3065 Bowers Avenue, Santa Clara, California 95051 • (408) 246-7501

MICRO COMPUTER SYSTEMS

SIM4-02 Hardware Simulator

Nine PROMs (A0750 to A0758) plug into the SIM4-02 prototyping board, enabling your micro computer prototype to simulate and debug its own program. The Simulator when used in conjunction with the SIM4 Hardware Assembler and the MCB4-20 System Interface and Control Module provides complete program assembly, simulation and debugging.

DECEMBER 1972

CONTENTS

		Page
1.0	**Introduction**	1
2.0	**Number Systems**	1
3.0	**Description**	1
4.0	**Directives**	2
5.0	**Error Messages**	4
6.0	**Operating Instructions**	4
	6.1 Assemble Program	4
	6.2 Prepare SIM4-02 Hardware	4
	6.3 Load Program	4
	6.4 Execute Program Simulation	5
	6.5 Edit Program	5
	6.6 Punch New "BNPF" Tape	5
	6.7 Simulation of Segmented Programs	5
7.0	**Jumps to Page 0**	5
8.0	**RAM Usage**	7
9.0	**Examples**	8

SIM4-02 HARDWARE SIMULATOR

1.0 INTRODUCTION

The SIM4-02 Hardware Simulator is a program written for the MCS-4 ™ series Micro Computer System. This program will provide interactive control over the debugging of other MCS-4 ™ programs.

The minimum configuration required is a SIM4-02 prototype card with three 4002 RAMs and a Teletype. When fully stuffed with 16 RAMs, test programs up to 512 bytes (locations) in length may be accomodated. The hardware simulation program itself occupies nine full ROMs.

The Hardware Simulation Program has two basic functions:

1. To simulate the execution of a test program, tracing its progress, and apprehending gross errors.
2. To allow the user to dynamically interact with and/or modify his test program, in order to facilitate the debugging process.

These two functions are implemented by means of a set of directives or commands which the user types in at the teletype keyboard. Some of the directives call for typeouts by the simulator program, some of the directives signal the input of data or program modifications, and some of the directives involve both typeouts and input response or data.

A directive is identified by a single letter of the alphabet (except the arithmetic conversion directives = and "). If the directive is associated with output only, the typing (or punching) will commence immediately. If input is allowed or required with the directive, the simulation program will enable the paper tape reader control, and wait for valid input data.

2.0 NUMBER SYSTEMS

Two number radices are standard with the hardware simulation program: binary and decimal. Index register values, program counter and instruction location values, chip numbers, and some pointers are handled in decimal for convenience. ROM instructions, the accumulator value, and one-bit indicators are handled in binary. Any input number may be entered in either radix by prefixing it with a suitable indentifier ("D" for decimal, "B" for binary), regardless of the expectations of the program. Unless so identified, however, all input should be in the radix used in the corresponding typeout.

To facilitate working with program tapes in the "BNPF" format, the hardware simulation program will accept binary numbers coded either as strings of ones and zeroes, or as strings of "P"s and "N"s, where the letter P is interpreted as a zero, and the letter N is interpreted as a one.

All input numbers are right-justified into the receiving register or field. If the number is smaller than the receiving field, leading zeroes are implied as necessary. If the number is larger than the receiving field, the excess bits are lost from the most-significant end of the number. Thus, if it is attempted to load an index register with the value 20, the result will be 4 in the register. This may be used to advantage in the event of an inadvertant error typein, by typing in as many zeroes as there are bits in the receiving field, then re-typing the number, all as one string of digits. A number typed in may end with a carriage return, a comma, a space, or the letter "F", or in the case of the = directive, with plus or minus sign. Any other characters will give unpredictable results, and should be avoided. Rubouts are the only non "numeric" characters which may be imbedded within the input number strings with no adverse effects. Rubouts are ignored in all cases.

3.0 DESCRIPTION

The hardware simulation program allocates a user-selected block of RAM main memory locations to hold the ROM instructions to be simulated, assigning two RAM locations for each simulated ROM location. Thus, to simulate 512 locations of ROM, all 16 RAMs must be used. Any RAM locations not allocated for program storage may be accessed in the normal way by the test program. In addition, the hardware simulation program uses the status characters in twelve consecutive RAM registers (equivalent to three RAM chips) to hold simulation parameters. RAM is assumed to be organized as four consecutive banks (with wraparound) of sixteen registers each, so that if less than 16 RAMs are used, those allocated to program and parameter storage must be in one block of contiguous banks and registers within banks.

The program to be tested may have an address anywhere in the 4096 locations of addressable ROM, since the hardware simulator program adds a bias value to all addresses which reference the simulated ROM. If the program attempts to jump or increment to outside the range of the simulated ROM, an error interrupt occurs.

Another error interrupt occurs in the event of an illegal instruction op code during simulated execution. The op codes which cause this interrupt are: 11111110, 11111111, 11100011, and all instructions with OPR = 0000 except for 00000000 (NOP).

A breakpoint register is associated with the simulated execution mode of operation, allowing the user to pre-set a location which will cause an interrupt before execution. The BREAK key on the teletype may also be used to interrupt execution and some other types of output.

During simulated execution, a count is kept of the number of simulated machine cycles (i.e., sync pulses) used by the test program, to assist in checking out programs with critical timing problems.

4.0 DIRECTIVES

"n **Binary conversion**
This directive accepts a single (decimal) number, and types out its equivalent in binary. A maximum of 12 bits (numbers to 4095) may be accomodated at once.

=n **Decimal adder**
This directive accepts a string of (decimal) numbers separated by plus and minus signs, and types out the algabraic sum, modulo 4096. If the algebraic sum is negative, 4095 is logically added to it to give a positive result. This directive may be used to perform binary to decimal conversion thus: =Bnnnn

Qr,s,e **RAM/ROM chip assignment**
This directive must be entered before any other letter directive. If the first character after the Q is a space or comma, the current values are typed out. Then, or immediately after the Q, three parameters separated by commas or spaces are required. If any of the three parameters is omitted, or if a RETURN is typed instead of the first parameter, the current values will be unchanged. **r** is the (decimal) RAM register number (0-63) which is used as the lowest in the block allocated to ROM. The simulation program has no way of preventing the test program from accessing RAM locations allocated to simulated ROM, so the user must use care in selecting a value for this parameter which will reduce the likelihood of improper access. If **r** is greater than 63, the previous value is used. **s** is the starting address of the ROM segment to be simulated. Any attempt to execute an instruction with an address less than this number will result in an out-of-bounds interrupt. **e** is the (decimal) ending address of the ROM segment to be simulated. Any program access to ROM locations greater than this address will result in an out-of-bounds interrupt. This directive clears the option word to zeroes.

Z **Zero**
This directive simulates the hardware reset function, and clears to zero all simulated registers, counters, and all RAMs not allocated to program. The Q directive executes a Z each time the parameters are changed.

In **Input**
This directive accepts a sequence of (binary) numbers and stores them in consecutive simulated ROM locations, beginning with location **n**. Spaces, commas, returns, and linefeeds may occur with any frequency or pattern between the individual numbers. Input is terminated by a free-standing letter F in the sequence. An ASCII SOH (control A) may be used to introduce a (decimal) number which, like **n**, becomes the new starting address for subsequent instruction bytes. The program counter in the current stack level is altered by this directive.

Pn,m **Punch**
This directive will punch out, in "BNPF" format with location numbers, the contents of the simulated ROM beginning at location **n** (decimal), and ending with location **m**. The currently selected program counter, and the breakpoint register are altered by this directive. Four inches of leader and trailer are punched on the tape, with an "F" after the last location. If the BREAK key on the teletype is depressed between locations, the typeout will be aborted, with no trailer. Both the breakpoint register and the program counter in the current stack level are altered by this directive.

Mn,i **Memory input**
This directive accepts a sequence of (decimal) numbers (0-15) and stores them sequentially in consecutive RAM locations, beginning in register **n** (decimal, 0-63) and location **i** (decimal, 0-19). If the starting location is in main memory (**i** less than 16), only main memory locations are filled. The next number after the one which goes into register **r**, digit 15, goes into register **r** + 1, digit 0. If the starting location is a status character (**i** greater than 15), only status characters are filled. The sequence ends with a carriage return following the terminal delimiter of the last number.

Dr, n **Dump RAM**
This directive types out in decimal the contents of each RAM location (both main memory and status) of **n** registers, beginning with register **r**. The typeout may be ended prematurely by depressing the BREAK key.

A **Accumulator**

This directive may be used to display and/or alter the contents of the simulated accumulator. A space or comma following the A will display in binary, the contents of the simulated accumulator. This or the A may be followed either by a new value to be entered, or by a carriage return to end the directive.

C **Carry/Link**

This directive may be used to display and/or alter the contents of the simulated Carry/Link bit. The use of this directive is the same as for A.

Xn **Index**

This directive may be used to display and/or alter the contents of any one or pair of the simulated index registers. If a space or comma follows the X, all 16 index registers are displayed (in decimal). Otherwise, if the (decimal) number **n** is followed by a space, index register **n** may be displayed and/or altered as in A. If the number **n** is followed by a comma, slash, or period, index **pair** number **n** may be displayed (in decimal) and/or altered as a unit. (Index registers 2n and 2n + 1 are handled together as a single 8-bit number.)

S **Stack pointer**

This directive may be used to display and/or alter the contents of the subroutine stack pointer. The current location counter is the one pointed to by this pointer. This pointer is incremented by JMS instructions and decremented by BBL instructions.

L **Location Counter**

This directive may be used to display and/or alter the contents of the current location counter. Note that altering the value of the stack pointer will cause a different register to be current location counter.

E **Examine Everything**

This directive combines the display functions of the C, A, S, L, R, and X directives. All four program counters in the stack are displayed. No modification is possible.

R **RAM/ROM selection**

This directive may be used to display and/or modify the simulated memory chip/location selection. A space or comma after the R types out an 11-bit binary number, of which the most-significant 3 bits represent the command line selection effected by the last DCL instruction, and the least-significant 8 bits represent the contents of the index pair as last used by an SRC instruction.

B **Breakpoint**

This directive may be used to display and/or modify the contents of the breakpoint register. The simulated execution will always be interrupted before processing the instruction pointed to by the breakpoint. If the breakpoint points to the second byte of a two-byte instruction, no breakpoint action will occur during instruction simulation.

W **When**

This directive may be used to display and/or alter the contents of the simulated sync cycle counter. This is a 12-bit (decimal) counter used to tally the number of instruction cycles used during instruction simulation.

T **Trace**

This directive causes the simulation program to begin simulated execution of the test program, beginning at the address in the current location counter. If instruction execution simulation is interrupted by a breakpoint, the keyboard BREAK key, or an illegal instruction, the T directive will cause the program to resume where it left off, just as if the program was never interrupted, except insofar as program parameters or registers were modified while interrupted. The basic format of the trace listing is

pppp:iiiiiiii **c** aaaa rr

where **pppp** is the decimal location counter; **iiiiiiii** is the binary representation of the (first byte of the) instruction being simulated; **c** is the resultant carry/link bit; **aaaa** is the resultant accumulator value; and **rr** is the resultant (decimal) index or index pair used in the instruction. (value, not index number) Any or all of the last three numbers may be omitted on instructions which do not reference their respective registers.

N **Non-trace**

This directive is identical to the T directive, except that execution proceeds without tracing.

O **Options**

This directive may be used to display and/or alter the current option status bits. This is a 4-bit binary number with the following significance:

1	Input, Output, and CPU test instructions are executed directly when this bit is on, instead of typing out on the teletype the port number and then typing or accepting the data.
10	No interrupt for subroutine stack overflow or underflow will occur when this bit is on.
1000	Unconditional jumps and subroutine jumps to ROM page 0 (chip 0) are executed directly instread of interpretively, permitting direct byte I/O during checkout.

5.0 ERROR MESSAGES

Most of the errors which can be detected by the simulation program are identified by a single character typeout, followed by ringing the bell once. Six different types of errors are identified this way:

CODE	SIGNIFICANCE
?	This is not a valid directive. Any printed graphic normally generated by the ASR33, which is not a valid directive, evokes this response. A question mark-bell combination also calls attention to a simulated input request.
#	Break condition recognized. This occurs normally, either when the location counter reaches the value in the break register in execution simulation, or when the BREAK key is depressed in simulation or ROM or RAM dumping.
>	Location counter out of range. This error occurs in simulation or ROM punching, if an attempt is made to access an address out of the range specified in the most recent Q directive.
!	Invalid op code. This error occurs and is recognized during execution simulation, after the instruction byte is typed out, but before the location counter is incremented, so that if it occurs under the control of the N directive, the T directive may be entered to examine the error by trying to execute it again.
%	Location counter stack overflow or underflow. This error is unique in that the interruption occurs after the instruction has been executed in simulation. A T or an N directive will resume execution with the next instruction (jumped or returned to).
↑	Cancel. This is the program response to a Cancel (Control "X" or ESCAPE) typein, during data input. Except for I, M, T, and N directives, it cancels the entire directive. If used in the I or M directives, only the current datum is canceled, and the directive is terminated at that point. Previous values, if any, have already been stored in memory. If used while the simulation program is requesting input data from a simulated ROM port or the simulated CPU Test line, it is equivalent to a break at the beginning of the instruction. In each case the simulation program returns to accept another directive.

6.0 OPERATING INSTRUCTIONS

6.1 Assemble Program

First assemble the test program on the Hardware Assembler (A0740 to A0743). *Important:* the Hardware Simulation Program will not accept ROM program tapes created by the FORTRAN assembler, ASM4, as these tapes have bit patterns and addresses together, with no identifier for the addresses. It is not necessary to assemble the program in one contiguous block of ROM locations, since the I directive in the simulation program is able to recognize the address fields (by the Control "A", SOH preceding them), and place the instruction patterns into the proper simulated ROM locations.

6.2 Prepare SIM4-02 Hardware

Remove ROM chips A0740 through A0743 and plug in the Hardware Simulation Program chips, A0750-A0758. Press RESET. The teletype should type out a carriage return-linefeed, and an asterisk to show that the simulation program is ready to accept a directive.

Determine how much simulated ROM is needed to test the program, and which RAMs are least likely to be accessed by the test program, using if necessary the = and " directives. Then type in the Q directive for this program. From now until the testing of this program has been completed and the amended program tape has been punched out, *DO NOT touch the RESET button. If the RESET button is pressed, the simulation parameters and any program in the simulated ROM will be destroyed.*

6.3 Load Program

Place the object tape in the teletype reader, and type in I. The simulation program will read both the addresses and the instructions from the tape and store them in the proper locations. Reading will be terminated by any error or by the terminal F punched by the assembler. Note that any instructions or data which fall outside the limits defined in the Q directive will be ignored. Note also that if the Q directive defines the ROM limits to be more than 512 bytes, wraparound overlap is possible. Thus, location 100 would be overwritten by instructions going into location 612. When the program is loaded, a simulated RESET may be effected by the Z directive. If starting at other than location zero, or with registers preloaded with data, the appropriate directives may be used to set these up. A breakpoint may be set if desired, and RAM may be loaded up with data if desired. If a subroutine or a part of a subroutine is being tested, the stack may be loaded with a

return address using the L directive. That may then be pushed down with the S directive, so that the starting address may be loaded into the first subroutine level, or the process may be repeated up to three times. If it is desired to force an interrupt at the first occurrence of a JMS instruction, the stack pointer may be set to 3 initially, so that the first JMS instruction causes a stack overflow. If it is desired to achieve more than one breakpoint, illegal instructions may be assembled or inserted into the program at the desired points. When the simulation attempts execution of one of these locations, an interrupt occurs, and the instruction may be replaced, or the program counter incremented around it to proceed.

6.4 Execute Program Simulation

To start the execution simulation, type a T (for Trace mode) or an N (for non-Trace mode). If at any time it is desired to stop execution simulation, whether because of program errors, to examine register contents, or to make corrections, the BREAK key may be depressed, and the simulation will be interrupted at the completion of the current instruction. Execution will resume as if uninterrupted, if the T or N directive is typed in after a break.

6.5 Edit Program

To make corrections to the program, the I directive is used, giving an address, and the value(s) to be entered. The I directive alters the contents of the current location counter. Thus, it should either be noted and restored, or the stack pointer may be incremented first and decremented afterwards – (unless of course, the simulation is interrupted at subroutine nest level 3).

6.6 Punch New "BNPF" Tape

After the program works correctly, an amended ROM tape may be punched in the "BNPF" format using the P directive. Four inches of leader and trailer are punched by this directive. If more is needed, rubouts or nulls (shift-control-P) may be punched while the simulation program is waiting for a directive. This will not in any way interfere with normal operation of the program. The user should remember to turn on the paper tape punch after typing in the second address in the P directive if a tape is to be made. If it is desired only to examine the contents of a simulated ROM location, this is not necessary.

6.7 Simulation of Segmented Programs

If a program is not very large, but is scattered over a wide range of addresses, it may be possible to accomodate the program in segments. Suppose the program occupies the first 32 locations in each of four ROMs. 128 locations must be reserved by the Q directive to hold all of this. Suppose further that the program accesses only bank zero in RAM. The Q directive would be something like this:

```
Q16, 0, 127
```

Then the first 32 locations of the program tape are read in using the I directive. The entire tape may be read with no deleterious effects, if that is convenient, or an F may be typed in manually at the end of the first 32 locations' worth of data. Then the Q directive is used again, to re-assign the same locations to the next block of addresses:

```
Q99, 224, 355
```

Note that the address limits have been offset by 32, to prevent the obliteration of the first 32 locations. The object tape may be read in again, or at least that part of it which includes the next block of data or instructions. Then the area is reassigned again:

```
Q99, 448, 575
```

The process is repeated until the whole program is loaded. To execute, the Q directive for the starting block of code is typed in again. If the segments are placed correctly, each time a jump is made to another segment, an out-of-range interrupt occurs. The Q directive for the segment jumped to is entered, and the program may proceed. This technique may also be used to relocate a program in ROM: for example, the following sequence of commands will effectively move (shift) a program up one position in ROM:

```
Q0, 0, 255
I0 (program)
Q0, 1, 256
P1, 256
```

7.0 JUMPS TO PAGE 0

Because of the nuisance of doing serial-to parallel conversion, and properly timing the bit frames in teletype input and output, the simulation program is provided with an option to perform subroutine calls and unconditional jumps to ROM page 0 directly, returning to simulation mode upon return. ROM page 0 contains subroutines to perform teletype reader and keyboard input, 7 bits wide (the parity bit is ignored), teletype output 8 bits wide, binary to decimal conversion and output,

the typing of some specialized sequences of characters, partial decimal to binary conversion on input, and 6-bit teletype character input with control character checking. A test program may use these subroutines to facilitate checkout of complex programs, or the ROM may be included in the final program if teletype interface and the same ancillary routines are needed.

The following is a summary of the subroutines and their calling parameters:

NAME	ADDRESS (X)	FUNCTION
KEY	120	(11-15) This routine inputs one 7-bit character from the teletype keyboard, and returns it, left-justified, in index registers 14 and 15. Index registers 12 and 13 are cleared to zero. The least significant bit of register 11 determines whether the character is echoed back (0 = yes, 1 = no). The carry is set if the character typed in is printable.
TTI	117	(11-15) This routine inputs one 7-bit character from the teletype paper tape reader or keyboard (the reader control is enabled), and is otherwise exactly the same as KEY.
TXX	234	(10, 11, 14, 15) This routine examines the carry bit set by KEY or TTI as well as the accumulator value and the character in registers 14 and 15 to determine if one of the following conditions obtains: 1. The character is some printable graphic between "Ø" and "←". If so, the character is biased to a six-bit value, centered in the byte. The carry is turned on if it is a letter. A normal return is taken, acc=0. 2. The character is a control character between null and ETB (control-Y) or a printable graphic between space and slash. An indirect jump to the address in ROM page 1 contained in index registers 10 and 11 is taken. 3. The character is a control between CAN (Control-X) and US (Shift-Control-0). An unconditional jump to location 256 is taken. 4. The character is one of those not generated by a KSR33 teletype, or a rubout. A normal return is taken, with the accumulator non-zero.
T6R	205	(10-15) This routine combines TTI and TXX such that normal return occurs only on characters "0" through "←". Characters in group (4) above are ignored, and the alternate exits are taken for control characters and delimiters. Note that if the address to which the delimiter return is to be made is odd, no echo occurs. On normal return, the character is right-justified in registers 14 and 15, and the carry is set if the character is a letter or higher.
T6L	220	(10-15) This routine is the same as T6R, except that on normal return the carry is always zero, and the character is left-justified in registers 14 and 15, leaving the lower two bits zero. Both T6R and T6L contain subroutine calls nested three deep, and may only be called from the main program, except during simulation.
D10	183	(4-7, 10-15) This subroutine multiplies the 12-bit binary number in registers 5-7 times ten, and adds the number in register 15 to the product, then goes to T6R to input another digit. This routine may be called repeatedly to input and convert to binary a decimal number. A terminal delimiter takes the alternate exit in registers 10 and 11. Register 4 is used for scratch.
Z47	6	(4-7) This routine clears registers 4 through 7 to binary zero in preparation for D10.
PUN	80	(11-15) This routine prints or punches the character in registers 14 and 15 out on the teletype. Registers 11 through 15 are cleared to zero on return.
IPR	70	(11-15) This routine does the same as PUN, except that if register 11 is initially even (echo mode on input), a 15 ms delay occurs to allow the teletype printer to settle.
RETN	107	(11-15) This routine types out a carriage return, a null, and a linefeed. It may only be called from the main routine.
MSG	66	(11-15) This routine types out the character in registers 14 and 15, then follows it with a bell.
SPACE	63	(11-15) This routine types one space.
DIGIT	53	(10-15) This routine types an ASCII digit corresponding to the BCD number in the accumulator. If it is zero, and the register 10 contains 15, a space is typed instead. Unless a space is typed, register 10 is incremented.
PDN	40	(4-7, 10-15) This routine will print, with zero suppression, the four-digit decimal number in registers 4 through 7.
BCD	11	(1-7, 10-15) This routine converts the 12-bit binary number in registers 1-3 into decimal, and prints the four digits with zero suppression.

8.0 RAM USAGE

The simulation program, to facilitate full usage of the RAM, has organized it into a nominal block of 64 registers, each containing 16 main memory locations and four status locations. Directives which reference RAM as such (i.e., Q, M, and D), always address it by a register number, and sometimes by a character position within the register. The following chart illustrates this addressing scheme:

	REGISTER (selected by even index in SRC instr.)	DIGIT (selected by odd index in SRC instr.)															[Status]			
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Bank 0 RAM 0 (Port 0)	0																				
	1																				
	2																				
	3																				
Bank 0 RAM 1	4																				
	5																				
	6																				
	7																				
	8																				
	9																				
	•																				
	•																				
	•																				
Bank 1 RAM 0 (Port 4)	16																				
	17																				
	18																				
	19																				
	20																				
	•																				
	•																				
	•																				
Bank 2 RAM 3 (Port 11)	44																				
	45																				
	46																				
	47																				
Bank 4 RAM 0 (Port 12)	48																				
	49																				
	50																				
	51																				
	52																				
	•																				
	•																				
	•																				
Bank 4 RAM 3 (Port 15)	60																				
	61																				
	62																				
	63																				

The bank number in the chart above is the value of the accumulator during a DCL instruction, needed to address that bank of RAMs. The port number given corresponds to the number typed out during simulation of the WMP instruction. Register positions 16-19 (i.e., the status locations) are normally addressed in the program by the RD0/WR0, RD1/WR1, etc., instructions, respectively.

The Q directive is used to define and set aside some part of RAM for use by the simulation program as simulated ROM and other registers and parameters. Whole RAM registers are taken by the Q directive, beginning with the register identified in the first parameter. The status locations from exactly 12 registers are used by the simulator. The number of main memory locations used is determined by the difference between the second and third parameters of the Q directive. Where **s** and **e** represent the values in the second and third parameters of the Q directive, the number of registers used is determined by the formula: $n = (s + e)/8 + 1$
This value may be more or less than 12, the number of registers whose status locations are used, with no ill effects.

RAM main memory locations reserved by the Q directive are used solely for program storage. The instruction with an address equal to the second parameter of the Q instruction is loaded into digits 0 and 1 of the register designated by the first parameter of the Q directive; subsequent instructions are loaded into the following digit pairs, according to the addresses.

The RAM status locations reserved by the simulation program are allocated to the following functions:

Relative REGISTER	LOC'N.	FUNCTIONS
r + 0	0	Simulated Accumulator
0	1-3	Low ROM address limit
1	0	Option word
1	1-3	High ROM address limit
2	0	Execution parameters
2	1-3	Breakpoint address
3	0	Simulated Carry
3	1-3	Simulation Cycle counter
4	0	Simulated Stack pointer
4-7	1-3	Simulated Stack
5	0	Simulated Command line selection
6,7	0	Simulated ROM or RAM chip selection
8-11	0-3	Simulated index registers

9.0 EXAMPLES

Figures 9.1, 9.2 and 9.3 are annotated listings generated during actual simulation. Figure 9.1 is a simulation of the "AND" program described in figures 9.4 and 9.5. Figures 9.2 and 9.3 represent the simulator's response to various directives entered via a TTY keyboard.

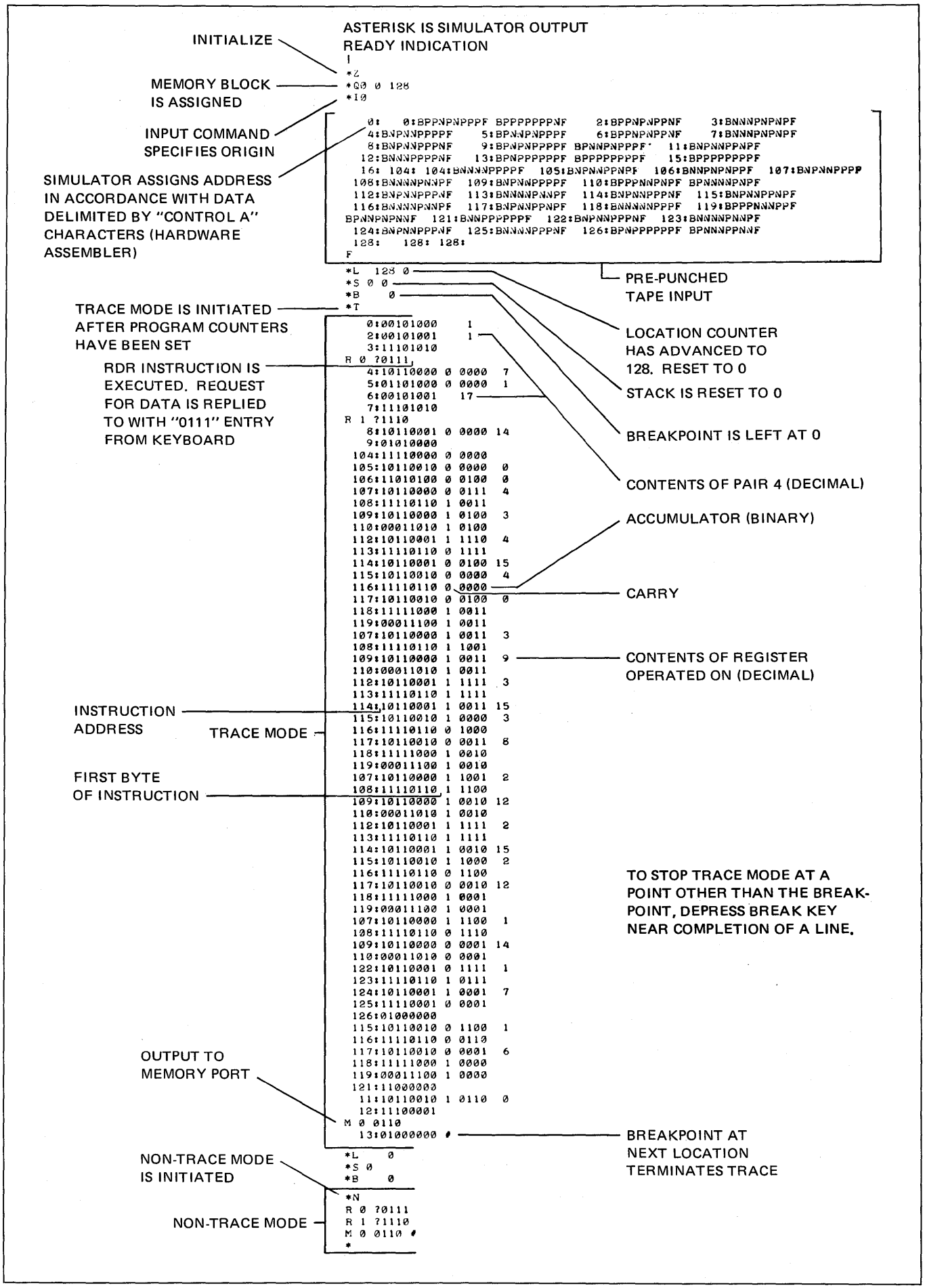

Figure 9.1. Trace and Nontrace Modes.

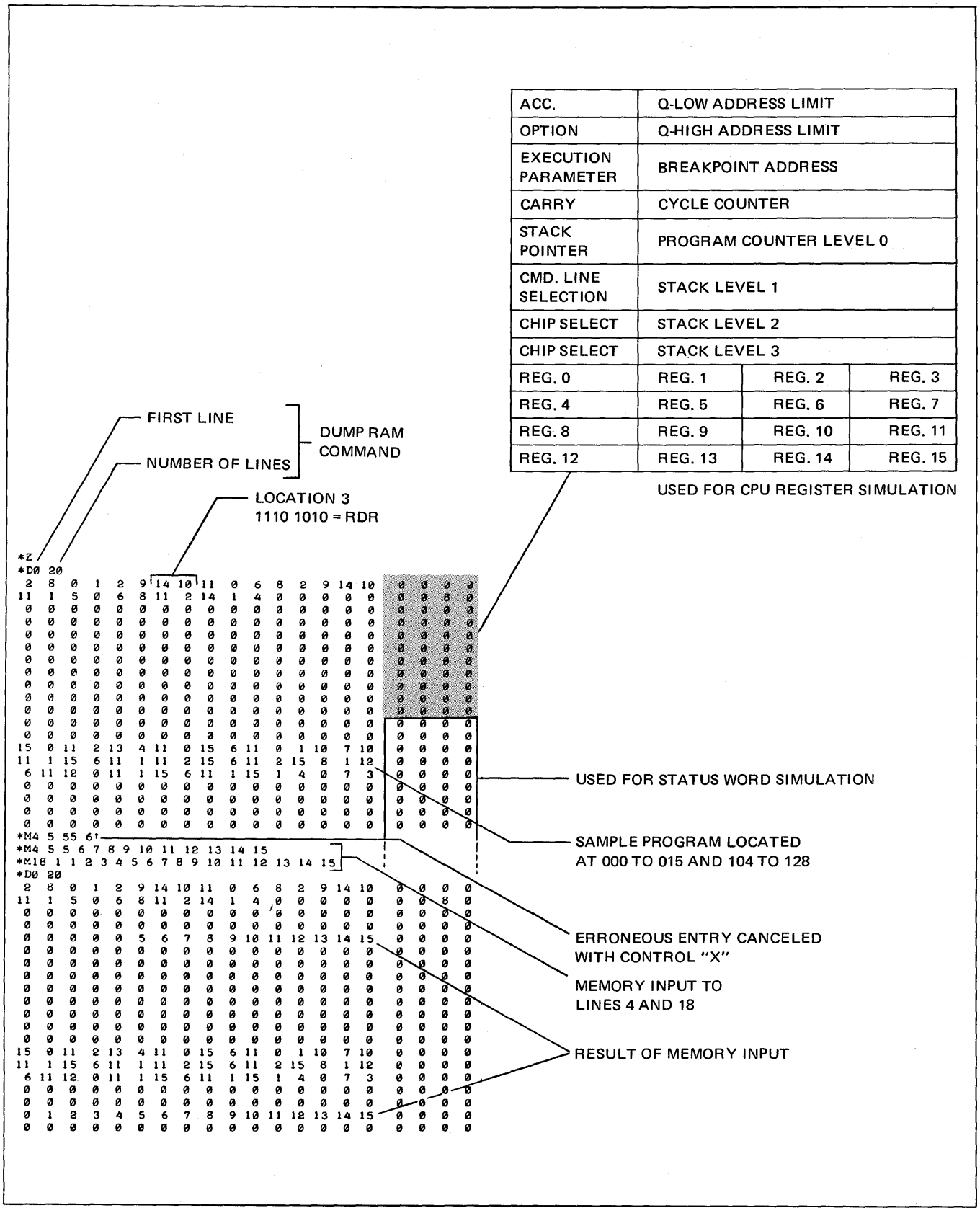

Figure 9.2 Memory Dump/Input.

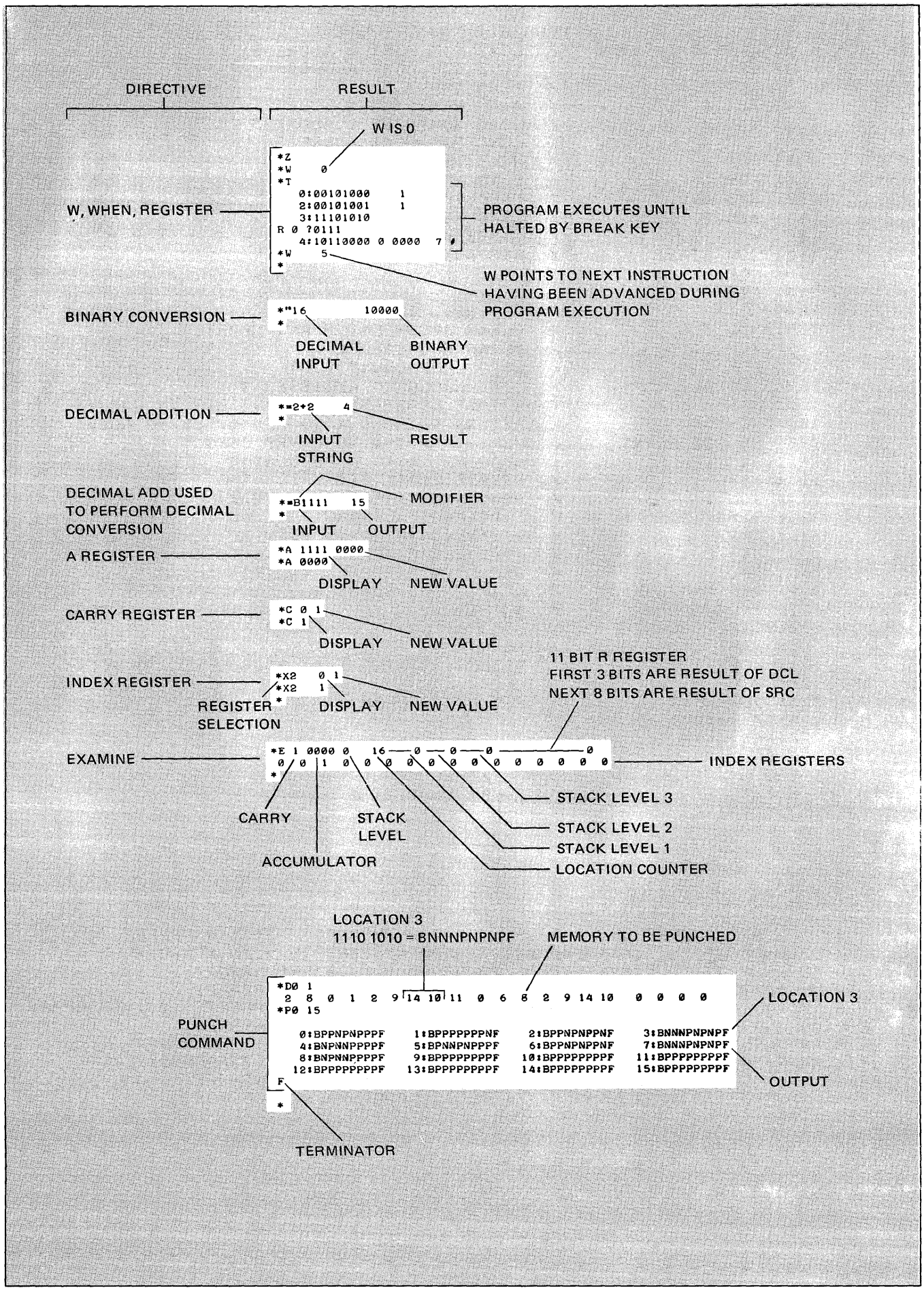

Figure 9.3 Miscellaneous Directives.

```
   0:/                    FOUR BIT "AND" ROUTINE
   0:START, FIM 4P 0       /  LOAD ROM PORT 0 ADDRESS
   2:  SRC 4P              /  SEND ROM PORT ADDRESS
   3:  RDR                 /  READ INPUT A
   4:  XCH 0               /  A TO REGISTER 0
   5:  INC 8               /  LOAD ROM PORT 1 ADDRESS
   6:  SRC 4P              /  SEND ROM PORT ADDRESS
   7:  RDR                 /  READ INPUT B
   8:  XCH 1               /  B TO REGISTER 1
   9:  JMS AND             /  EXECUTE "AND"
  11:  XCH 2               /  LOAD RESULT C
  12:  WMP                 /  STORE AT MEMORY PORT 0
  13:  JUN START           /  RESTART
  15:  NOP
  16:  =104
 104:/                    "AND" SUBROUTINE
 104:AND, CLB              /  CLEAR ACCUMULATOR AND CARRY
 105:  XCH 2               /  CLEAR REGISTER 2
 106:  LDM 4               /  LOAD LOOP COUNT (LC)
 107:  XCH 0               /  LOAD A, LC TO REGISTER 0
 108:  RAR                 /  ROTATE LEAST SIGNIFICANT BIT TO CARRY
 109:  XCH 0               /  RETURN ROTATED A TO REG 0, LC TO ACC.
 110:  JCN CZ ROTR1        /  JUMP TO ROTR1 IF CARRY ZERO
 112:  XCH 1               /  LOAD B, LC TO ACCUMULATOR
 113:  RAR                 /  ROTATE LEAST SIGNIFICANT BIT TO CARRY
 114:  XCH 1               /  RETURN ROTATED B TO REG. 1, LC TO ACC.
 115:ROTR2, XCH 2          /  LOAD PARTIAL RESULT C, LC TO REGISTER 2
 116:  RAR                 /  ROTATE CARRY INTO PARTIAL RESULT MSB
 117:  XCH 2               /  LOAD LC, RETURN C TO REGISTER 2
 118:  DAC                 /  DECREMENT THE ACCUMULATOR (LC)
 119:  JCN ANZ AND+3       /  LOOP IF LC NON ZERO
 121:  BBL 0               /  RETURN
 122:ROTR1, XCH 1          /  LOAD B, LC TO REGISTER 1
 123:  RAR                 /  ROTATE B
 124:  XCH 1               /  RETURN ROTATED B TO REG. 1, LC TO ACC.
 125:  CLC                 /  CLEAR CARRY
 126:  JUN ROTR2           /  RETURN TO LOOP
 128:CZ  =10
 128:ANZ  =12
 128:$
```

Figure 9.4. Pass 1 Listing.

```
   0:    0:BPPNPNPPPF BPPPPPPPNF     2:BPPNPNPPNF     3:BNNNPNPNPF
   4:BNPNNPPPPF     5:BPNNPNPPPF     6:BPPNPNPPNF     7:BNNNPNPNPF
   8:BNPNNPPPNF     9:BPNPNPPPPF BPNNPNPPPF    11:BNPNNPPNPF
  12:BNNNPPPPNF    13:BPNPPPPPPF BPPPPPPPPF    15:BPPPPPPPPF
  16:  104:  104:BNNNNPPPPF   105:BNPNNPPNPF   106:BNNPNPNPPF   107:BNPNNPPPP
 108:BNNNNPNNPF   109:BNPNNPPPPF   110:BPPPNNPNPF BPNNNNPNPF
 112:BNPNNPPPNF   113:BNNNNPNNPF   114:BNPNNPPPNF   115:BNPNNPPNPF
 116:BNNNNPNNPF   117:BNPNNPPNPF   118:BNNNNNPPPF   119:BPPPNNNPPF
BPNNPNPNNF   121:BNNPPPPPPF   122:BNPNNPPPNF   123:BNNNNPNNPF
 124:BNPNNPPPNF   125:BNNNNPPPNF   126:BPNPPPPPPF BPNNNPPNNF
 128:  128:  128:
F
```

Figure 9.5 Programming Tape Listing.