FACTOR

FOR

SENTRY 600/610/II

A PROGRAMMED INSTRUCTION MANUAL

Training Center

FAIRCHILD SYSTEMS TECHNOLOGY
1725 TECHNOLOGY DRIVE
SAN JOSE, CALIFORNIA 95110

FAIRCHILD
SYSTEMS TECHNOLOGY
A DIVISION OF FAIRCHILD CAMERA AND INSTRUMENT CORPORATION

TABLE OF CONTENTS

HOW TO USE THIS MANUAL

The text of this manual is self-instructional permitting the reader, as a student, to learn at his own rate. To accomplish this, the text is divided into information frames, reinforcement frames, and answer frames. The information frames, as might be expected, provide the reader with data relating to a particular topic. To reinforce the data presented and to verify your understanding of it, subsequent frame(s) will rephrase a statement of fact and require you to fill in a missing word or group of words, or to respond with an answer to a question. To validate your response, turn to the next page and check the corresponding numbered answer frame. Afterwards, return to the next sequential information frame and continue your study.

1. This is an "information" frame.	
2. A "reinforcement" frame is used to reinforce data or information presented in previous "information" frame.	
3. This is a Rein Forcement frame.	
4. After validating your answer for frame 3, you should continue reading here.	

5. This can be information or reinforcement frame.	
6. So can this one.	
7. This one too.	3. Reinforcement This is the "answer" frame for frame 3 on the previous page.
8. If you understand the difference between "information", "reinforcement", and "answer" frames then begin the lesson on Page 1, otherwise go back to frame 1 of this exercise.	

LESSON 1

TOPIC:

AN INTRODUCTION TO FACTOR

GIVEN:

1. The Booklet
   AN INTRODUCTION TO FACTOR

2. Pencil and eraser.

PERFORMANCE:

Student proceeds through the numbered frames, sequentially.

Student writes short responses in the response frames, compares his responses with the data provided in the answer frames and corrects his responses to agree with the furnished answers.

STANDARD:

The student provides 100% correct responses within 20 minutes.

AN INTRODUCTION

TO

FACTOR*

A COMPILER LANGUAGE

FOR THE

SENTRY 600

SENTRY 610

SENTRY II

---

*Herein referred to as "S600 FACTOR"

1. FACTOR is a human language. People understand it, machines don't.	
2. After you have learned to write a FACTOR program, it will have to be translated (compiled) before the Sentry System can react to the information in the program.	
3. When the FACTOR program has been compiled (translated) the end result could be called machine language. Some people learn to read this (people are smarter than machines)	
4. SO ARE YOU!!!! (But you're probably not nearly as ornery, stubborn, mule-headed, cussed....etc.,etc., etc.,....)! You will learn how to have your FACTOR program translated (compiled) in another lesson. This lesson introduces you to the FACTOR language.	

5. EVERY Factor statement ends with a semicolon. ;	
6. EVERY program written with S600 Factor starts with a SET PAGE number; statement.	
7. For example SET PAGE 1Ø; is a valid opening statement for an S600 FACTOR program. Until you learn more FACTOR, don't use a number larger than 1Ø24;	
8. Try writing a statement that can start an S600 FACTOR program. SET PAGE 200 ;	

9. Take a look at your answer to frame 8. Did you make any spelling errors? Did you leave at least one space after SET and after PAGE? Did you remember the semi-colon? Did you use solid capitals?	
10. If you wrote the statement correctly, the compiler would be able to translate (compile) it correctly.	
11. FACTOR is a compiler language using English language-type statements.	
12. FACTOR is unique to SENTRY test systems.	8. SET PAGE 1024; (Any number between 1 and 1024 is satisfactory during this lesson.)

12. Circle all true statements. a. FACTOR is a machine language. b. FACTOR is a compiler language. c. FACTOR is a programming language for Sentry test systems. d. FACTOR is a universal compiler language used by the semiconductor industry.	
14. ALL FACTOR statements end with a ; .	
15. FACTOR provides two types of statement: 1. Arithmetic and logical control statements. 2. Test control statements, which set up and execute tests on electronic devices.	
16. You can learn much about the arithmetic and control statements without even seeing a machine Let's start right now.	

17. A = 6 is algebra A = 6; is a FACTOR statement.	13. b and c should be circled.
18. The FACTOR statement A = 6; is not a simple statement of equality.	14. ; If you wrote the word "semicolon", erase it and replace it with ";".
19. A = 6; means store a value of 6 in a location named A.	
20. A = 6; is a variable assignment statement.	

21. You don't have to know where the location called A is. That is the Sentry's problem.	
22. Once you have told (programmed) the system that a location, that you name, contains a value, the Value is stored in the named location.	
23. A = 6; is a FACTOR variable ASSIGNMENT statement.	
24. A = 6; means store 6 in a location named A.	

25. A + 1 is an algebraic expression. It can be used in FACTOR as an algebraic expression.	
26. A = A + 1; is a legal FACTOR statement, but it is not an algebraic equation.	22. value location
27. A = A+ 1; is legal in FACTOR because it is a variable assignment statement, not an equation.	23. assignment
28. A = A + 1; means Fetch the present contents of A Add 1 to the number fetched Assign the sum to location A	24. (a value of) 6 location A

29. Interpret the following:

```
SET PAGE 1Ø24;
 A = 6;          A = 6
 A = A + 1;      A = 6 + 1 = 7
 END;
```

30. In framé 29, the statement

A = A + 1; means

fetch the value 6 from A (LOCATION)
Add 1
Store the sum in "A" LOCATION

31. In frame 29, the final value of A (LOCATION) (before END;) is 7.

32. Only the system knows where A is.

33. Is A = A + 1; an algebraic equation? *No!*	
34. A + 1 = A; is a MISTAKE (and the compiler will call it a syntax error).	30. 6 (location) A 1 (location) A
35. Location of a FACTOR variable must be defined by a single variable identifier.	31. I have no idea, but I know that location A contains 7.
36. Circle the legal FACTOR statements. C = 25; A = A + C; 14 = D; C + 2 = 27;	

37. Perhaps you wondered about 14 = D; This is incorrect because the variable identifier must always be on the left of the equal sign.	33. NO
38. A variable assignment statement can generally be described by `variable = expression;`	
39. In frame 34, because the words "variable" and "expression" are written in lower case letters, the programmer is free to use his own variable identifier and his own expression.	
40. `variable = expression;` is the SYNTACTICAL format of the variable assignment statement. SYNTAX is the set of rules for a language.	36. `C = 25;` `A = A + C;`

41. Every S600 FACTOR program must start with SET PAGE number; Every FACTOR program ends with END;	
42. The words supplied for the SET PAGE number; statement imply, syntactically, that you must write SET PAGE exactly as shown (because of the capitals) but that you may supply your own number.	
43. Syntax is the set of rules for a language.	
44. FACTOR is a (human/machine) human language.	

45. FACTOR is a (compiler/machine) Compiler language.	41. SET PAGE
46. FACTOR is a procedural test language used universally by the semiconductor industry. (True/false) False.	
47. Every FACTOR statement ends with a ; . Every FACTOR program ends with END.	43. SYNTAX
48. X = 27; is a FACTOR Variable Assignment statement.	44. human

49. variable = expression; where variable must be a (single variable identifier/any algebraic expression) ________. singly Variable identifier	45. compiler
50. You may make up your own names (for variable identifiers) with very few exceptions.	46. false
51. You won't have to memorize the rules for naming variables because: 1. They are simple. 2. The compiler will warn you if you violate a rule. 3. As you learn more FACTOR statements the reasons for the rules tend to become obvious.	47. ; END
52. The following are acceptable variable identifiers: A CHISAUARE ALARGEIDENTIFIER A1B2C3D4 PHOENIX	48. variable assignment

53. The FACTOR compiler accepts only the first eight characters of a variable identifier, ignoring any additional characters.	49. single variable identifier
54, Thus ALARGEIDENTIFIER ALARGEID ALARGEIDEA ALARGEIDEAINDEED are all the same variable name that the compiler recognizes as ALARGEID .	
55. The compiler will not warn you if you use more than eight characters to identify a variable.	
56. It is good practice, since the usage of identifiers is totally determined by the programmer, to use names (identifiers) that represent the meaning or use of a variable. TEMP, for instance, could be the name given to a working variable. COUNTER might be the name given a variable that is used as a general purpose counter, and so on. (This does not imply, however, that FACTOR attaches any significance to these names. They are purely artificial devices that aid the user's memory and make a program more intelligible.)	

57. If you use a variable without first assigning a value to it, the initial value becomes zero. An example is in frame 58.	
58. B = A +1; If this statement is the very first reference to A in the program, zero (0) is stored in A. The statement B = A + 1; results in 1 being stored in location B.	54. ALARGEID
59. SET PAGE 1Ø24; B = 9; 9 B = A + 4; A+4 => 0+4 B (location) contains 4 END;	
60. Refer to frame 59. The last value stored in the location called B is 4.	

61. The following are not acceptable variable identifiers:

123	(Identifiers may not start with a digit)
AB C	(Special characters, including blanks, are not allowed)
END	(Reserved words are illegal)

62. You are not finished with this lesson until you look at the rules on the next page.

60.

4

(If you wrote down 9 or 13, repeat frames 57-60)

RULES FOR NAMING VARIABLES

1. A variable name can be of any length up to eight characters. Any additional legal characters are ignored.

2. The first character must be a

   Letter (English alphabet only)
   or
   # (often called a "pound sign").
   or
   $ (The dollar sign)

3. The rest of the characters can be any combination of

   Any character allowed for the first character (letter, #, $).
   or
   Numbers (digits)
   or
   . (Period).

4. The following words are reserved for the system and are, therefore, not acceptable variable identifiers.

AND	DCL	GOTO	PAUSE	WR
ASSIGN	DISABLE	GT	PGEN	WRITE
AT	DO	IF	PGM	XCON
BEGIN	ELSE	INSERT	RD	XCONF
BLOCK	ENABLE	LEQ	READ	XPMU
BRANCH	END	LT	REM	
BY	EOR	MEASURE	RESET	
CALL	EQ	NEG	SELECT	
CGEN	EXEC	NEQ	SET	
CONF	FOR	NOISE	SOCKET	
CONN	FORCE	NOT	SUBR	
CLEAR	FUNCT	ON	THEN	
CPMU	GE	OR	THRU	

LESSON TWO

TOPIC:

FACTOR RELATIONAL LOGIC AND BRANCHING

GIVEN:

1. The Booklet

   FACTOR RELATIONAL LOGIC AND BRANCHING

2. Pencil and eraser

PREREQUISITE: COMPLETION OF LESSON ONE

PERFORMANCE:

Student proceeds through the number frames responding to branching instructions.

Student writes short responses in the response frames, compares his responses with the data provided in answer frames and corrects his responses to agree with the furnished answers.

Standard: The student provides 100% correct responses within one hour.

FACTOR

RELATIONAL LOGIC

AND

BRANCHING

Special Notice to YOU
the
student

If a friend of yours offers to help you in a task, don't offend him by reminding him that the nails go into the board "pointy-end first."

On the other hand, when you are dealing with a computer, you often have to be ridiculously specific.

Some of the frames in this lesson might tend to aggrevate you.

e.g.

Getting angry with a machine provides it with correct data *False* (true/false)

If this frame made you angry then programming might not be your bag.

1. The Sentry System, in response to FACTOR programs, can react as if making logical decisions.	
2. A= expression is a variable *Assignment* statement.	
3. The English statement "16 is greater than 4" is *True* (true/false). *16 > 4*	
4. The FACTOR variable assignment statement A=16 GT 4; stores the value of the expression in the location called A.	

5. GT is a relational operator meaning greater than

---

6. A=16 GT 4; requires that a value, indicating that the relation 16 GT 4 is true, be in location A.

2. assignment

---

7. The value stored when variable = relational expression is 1 for true or Ø for false.

Ø

3. true

(Watch it!! STUPID was not one of the options)

---

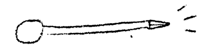
8. The relational operators are:

Symbol	Operation
EQ	equal
GE	greater than or equal
GT	greater than
LT	less than
LEQ	less than or equal
NEQ	not equal

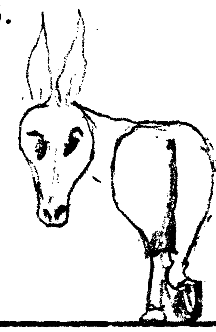
9. Practice by completing frames 10 through 12	5. (is) greater than
10. GIVEN: SET PAGE 1; A=6; B=1; C=A LT B; END; Location C contains ____Ø____.	6. stored in (location) A
11. SET PAGE 1: A=6; B=1; C=A LT B; 6 ≮ 1 False C=Ø D=A GT B; 6 > 1 True D=1 E=D GE C; 1 ≥ 0 True 1 END; Location E contains ____1____.	
12. Circle the true relationships (5 NEQ 6) 9 EQ 2 14 LEQ 13 (-4 LT 3) -4 < 3 (12 GE 6) 9 GT 9	

13. If you had trouble with frame 11 GOTO frame 15. *No Trouble*	
14. GOTO frame 16	10. Ø
15. In frame 11, C is Ø because 6 LT 1 is false, D is 1 because 6 GT 1 is true. Therefore E = D GE C; is E= 1 GE Ø; True, therefore E=1.	11. 1
16. Fill in the blanks *GT* means greater than NEQ means *Not Equel* *GE* means greater than or equal *EQ* means equal LT means *Less Than* *LEQ* means less than or equal	12. 5 NEQ 6 12 GE 6 -4 LT 3

17. = means Assign to variable a value	
18. IF 7 NEQ 28 then write 'true'. true	
19. You can cause the Sentry System to respond to the statement in frame 18 if you program as shown in frame 20.	
20. IF 7 NEQ 28 THEN WRITE 'TRUE'; Execution of this statement causes the word TRUE to be written out on an output device.	16. GT not equal GE EQ less than LEQ

21. IF relation THEN statement;	17. assign a value to or store a value in 
22. According to FACTOR syntax you, the programmer, must fill in (refer to frame 21) the *relation* and a *statement*.	18. true
23. IF 7 EQ 28 THEN WRITE 'TRUE'; Is the relation (ship) true? *No*	
24. Should a computer executing the statement in frame 23 write out anything at all? *No*	

25. If 7 is less than 6 then write true in the blank __________.	
26. You didn't have any choice, but to ignore the blank in frame 25, and proceed to the next frame.	22. relation (ship) and a (FACTOR) statement.
27. If 2 EQ 2 then write true in the blank True.	23. NO
28. After you filled in the blank you still had no choice but to proceed to the next frame.	24. NO

29. If 2 EQ 2 then GOTO frame 33.	25. 
30. You must have a problem of some sort or you wouldn't be reading this frame. Go back to frame 29 and do what it says to do.	
31. The branch you just executed is (conditional/ unconditional) UNCONDITIONAL	27. true
32. GOTO frame 38.	

33. You just performed a conditional branch. You branched from frame 29 to frame 33 because the condition in frame 29 is true.	
34. The IF - THEN statement is a form of conditional branch.	
35. Perhaps you feel that a conditional branch requires a GOTO.	31. unconditional
36. IF relation THEN A=6; A=A+1; Now, if the relation is true, A=6 is executed. If the relation is false, the program branches around A=6 and does not execute it. The branch is conditional.	

37. GOTO frame 31.	
38. The simple GOTO is an unconditional. branching statement.	34. conditional
39. The simpliest form of the GOTO statement is GOTO label;	
40. A label is a symbolic statement identifier.	

41. You can make up label names using the same rules you learned for naming variables.	
42. Once you have named a variable, that name cannot be used as a label and vice versa. Fortunately, the compiler knows this and issues warnings as shown in the next two frames.	38. unconditional
43.	
44.	

43.

```
000001                      SET PAGE 1;
000002                      TURKEY=1;
000003                      GOTO TURKEY;
000004     TURKEY: END;

TURKEY USE ERROR - DEFINED LABEL
0001B COMPILATION ERRS
```

44.

```
000001                      SET PAGE 1;
000002    DUCK:             A=2
000003                      DUCK=4;

DUCK USE ERROR - DEFINED LABEL

000004                      END;

0001B    COMPILATION ERRS
```

45. A label is a symbolic statement identifier.

46. LOOP: WRITE X;

Loop is a symbolic statement identifier which identifies the statement immediately following it (WRITE X;).

47. Note (frame 46) that a colon was used to separate (delimit) LOOP from WRITE.

48. But the colon is not part of the label and is not used when the label is referenced.

E.g.
```
LOOP: WRITE X;
      IF X LEQ 6 GOTO LOOP;
```

49. A label is a symbolic statement identifier.

50. A label is delimited by a ":" but it does not contain a : when referenced

46. symbolic

51. Label names follow the same rules as Variable names.

52. The same name can be used, in the same main program for both a label and a variable. (true/false) False

53. Now, if you are new to programming, you are probably anxious to see a practical use for all of this.	49. symbolic identifier
54. Let us suppose that you wish to prepare a form with numbers printed at the left side of the page.	50. : :
55. If you only wanted five numbered lines you would surely use a typewriter. 1 2 3 4 5	51. variable
56. If you wanted to number 9999 lines, consider frame 57.	52. false

57.
```
SET PAGE 1;
X = 1;
LOOP: WRITE X;
X = X + 1;
IF X LEQ 9999 THEN
GOTO LOOP;
END;
```

58. Now that is easier than using a typewriter! (Naturally, economics might be a factor, too!)

59. You now have some programming power. With power you have responsibility also.

60. Careless programming can put a computer into a loop from which it can't escape. Such a loop is often called an infinite loop. An example is in the next frame.

61. SET PAGE 1;

```
LOOP: X=1;
      WRITE X;
      X=X + 1;
      IF X LEQ 9999 THEN
      GOTO LOOP;
      END;
```

62. If you see why frame 61 is an infinite loop GOTO frame 66.

63. On the first pass thru the program, X starts as 1 then increments to 2. The relation is true. The program branches to loop which forces X to 1, again.

X is never assigned a value greater than 2.

64. If you don't see why 61 is an infinite loop THEN

BEGIN

```
SHAFT=1;
IF SHAFT LT 2 THEN
GOTO frame 60;
```

65. I hope you lived through the last page.

66. The simplest form of the IF-THEN statement is

IF relation THEN statement 1;

Statement 2;

(note: assume statement 1 is not a GOTO label statement).

67. Refer to frame 66

If the relation is true,

Statement 1 is executed (true/false) True

Statement 2 is executed (true/false) True

68. Refer to frame 66

If the relation is false,

Statement 1 is executed (true/false) FAlse

Statement 2 is executed (true/false) True

69. Often, it is desirable to execute more than one statement if the relation is true.

70. This can be accomplished by nesting the statements between

BEGIN

and

END;

71. Look closely at frame 70.

Is there a semicolon after BEGIN?

No.

Is there a semicolon after END? yes

67.

true

true

72. Past experience shows that many programmers forget the information in frame 71

```
IF N LT 2 THEN
BEGIN
N=N + 1;
GOTO frame 71;
END;
```

68.

false

true

73. If the relation in frame 74 is true, statements 1 thru n are executed before statement X. If the relation is false statements 1 thru n are not executed (i.e. program branches directly to statement X.)	

74. IF relation THEN
BEGIN
Statement 1;
Statement 2;
:
:
Statement n;
END;
Statement X;	

75. If statement n in frame 74 is a GOTO label statement, Statement X (must, cannot, might) _might_ be executed.	71. no yes
76. The correct response for frame 75 requires you to realize that the location of the label could cause a branch that bypasses statement X;	

77. The "END", statement that closes a nest of statements started by "BEGIN" does not end the program.	
78. A FACTOR program may contain more than one "END;" statement. (true/false) True	
79. If you fail to close the group of statements started with "BEGIN", all statements following "BEGIN" are seen by the compiler as inside the nested group. The next "END;" card would close the group.	75. might
80. Fortunately, the compiler counts the number of "END;" statements in your program. If it reads too few, it writes END OF FILE INPUT and points an arrow at the last statement it read.	

81.
```
000001      SET PAGE 1;
000002      A=1;   B=9;
000004      IF A LT B THEN
000005      BEGIN
000005              BUCKET =B;
000006              B=A;
000007              A=BUCKET;
000010                    END;
                            ^
END OF FILE INPUT
```

82. In frame 83 is an example of a properly ended program.

Note the statement

"0000B COMPILATION ERRS"

(the "B" reminds you that the number of errors is printed in Octal)

78.

true

83.
```
000001       SET PAGE 1;
000002       A=1;   B=9;
000004       IF A LT B THEN
000005       BEGIN
000005               BUCKET =B,
000006-              B=A;
000007               A=BUCKET;
000010           END;
000010       END;
0000B    COMPILATION ERRS
```

84. In the following frames are examples of coding. Each example is followed by a compiler listing. Try to spot the errors in each example before "peeking" at the compiler listing.

85.

```
SET PAGE 1;
A=1; B=9;
IF A LT B THEN
     BEGIN;
          BUCKET = B;
          B=A;
          A= BUCKET;
     END;
END;
```

BEGIN does not HAVE A Semicolone Ending IT

86.

```
000001      SET PAGE 1;
000002      A=1; B=9;
000004      IF A LT B THEN
000005           BEGIN;
INVALID TERMINATOR ^
000005                BUCKET = B;
000006                B=A;
000007                A= BUCKET;
000010           END;
000010      END;
0001B  COMPILATION ERRS
```

87.

```
SET PAGE 1;
A=1 B=9;
IF A LT B THEN
     BEGIN
          BUCKET = B;
          B=A;
          A= BUCKET;
     END;
END;
```

A=1 needs a semicolone To End the statement

88.

```
000001      SET PAGE 1;
000002      A=1 B=9;
EXPRESSION SYNTAX^
000003      IF A LT B THEN
000004           BEGIN
000004                BUCKET = B;
000005                B=A;
000006                A= BUCKET;
000007           END;
000007      END;
0001B  COMPILATION ERRS
```

Note:

The compiler's interpretation of the error may differ from yours. i.e. 1 B =9 is an improper expression.

89.

```
SET PAGE 1;
A=1;  B=9;
IF A LT B
     BEGIN
          BUCKET = B;
          B=A;
          A= BUCKET;
     END;
END;
```

→ No THEN

90.

```
000001     SET PAGE 1;
000002     A=1;  B=9;
000004     IF A LT B
000005          BEGIN
STATEMENT SYNTAX^
000005               BUCKET = B;
000006               B=A;
000007               A= BUCKET;
000010          END;
000010     END;
0001B  COMPILATION ERRS
```

Note:

The compiler does not detect that "THEN" is missing until it reads "B" of BEGIN.

91.

```
SET PAGE 1;
A=1;  B=9;
IF A LT B THEN
     BEGIN
          BUCKET = B;
          B=A:
          A= BUCKET;
     END;
END;
```

→ is not semicolon

92.

```
000001     SET PAGE 1;
000002     A=1;  B=9;
000004     IF A LT B THEN
000005          BEGIN
000005               BUCKET = B;
000006               B=A:
EXPRESSION SYNTAX      ^
000007               A= BUCKET;
000007          END;
000007     END;
0001B  COMPILATION ERRS
```

You expected maybe 'improper semicolon' ?

93.

```
SET PAGE 1;
A=1;   B=9;
IF A LT B THEN
      BEGIN
            BUKCET = B;
            B=A;
            A= BUCKET;
      END;
END;
```

None

94.

```
000001      SET PAGE 1;
000002      A=1;   B=9;
000004      IF A LT B THEN
000005            BEGIN
000005                  BUKCET = B;
000006                  B=A;
000007                  A= BUCKET;
000010            END;
000010      END;
 0000B   COMPILATION ERRS
```

Quote:

"Compiler quite happy pounding on pointy end of nail".

— Fucius Kahn

95. The program in frame 94 executes ending up with zero stored in A. You see B is stored in BUKCET.

96.

END;

LESSON THREE

TOPIC: NUMBERS, COMPILED AND PRINTED

GIVEN:

1. The Booklet

   NUMBERS, COMPILED AND PRINTED

2. Pencil and eraser

PREREQUISITE: COMPLETION OF LESSONS ONE AND TWO

PERFORMANCE:

Student proceeds through the number frames responding to branching instructions.

Student writes short responses in the response frames, compares his responses with the data provided in answer frames and corrects his responses to agree with the furnished answers.

Standard: The student provides 100% correct responses within 30 minutes.

NUMBERS,

COMPILED

AND

PRINTED

1. Integers may be entered by variable assignment statements

For instance

DECINT=8;

2. The FACTOR compiler recognizes a number as Octal if it is immediately followed by B.

For instance

OCTINT=1ØB;

3. No imbedded characters are allowed in an integer or between an octal integer and B.

4. INTEGERS

ACCEPTABLE	UNACCEPTABLE
8	8_Ø
1ØB	2,ØØØ
+9	27_B
+1ØB	9_B
+71	
-77B	

5. There is a limit to the magnitude of integer inputs.

It is

37777777B or, in

decimal form 8388607.

6.

ACCEPTABLE	UNACCEPTABLE
+8388607	+8388608
8388607	8388608
-8388607	-8388608
+37777777B	+40000000B
37777777B	40000000B
-37777777B	-40000000B

7. There is no compiler warning if you enter an integer that is too large.

See frame 8.

8. This entry

A=8388608;

results in

A=-1;

G O T C H A ! ! !

9. Decimal numbers may be programmed with a fractional part preceded by a period. Octal numbers cannot. The string cannot exceed 8388607.

10.

Acceptable	Unacceptable
20.2	37.1B
838860.7	838860.8
999.999	99999.99
.00002	

11. No number can end with a period.

Acceptable	Unacceptable
88.0	88.

12. Circle the unacceptable input values.

100.1	8388607.1
37.77B	9.0
267.	40000000
34218B	1,024

13. Decimal numbers may be followed by a "power of ten" multiplier specified as E followed by an integer. An example is in frame 14.	
14. 17E6 means 17 times 10^6	
15. Numbers expressed as shown in frame 14 are often called "exponentials" or are said to be in "engineering format".	
16. The exponent (following the E) must be a decimal integer. It can be signed or unsigned. FACTOR accepts an unsigned integer as positive.	12. UNACCEPTABLE 37.77B 8388607.1 267. 40000000 34218B 1,024

17.

ACCEPTABLE	UNACCEPTABLE
2E1Ø	2E1ØB
+2.4E-1Ø	+2.4E-1Ø.1
-6EØ3	-6.EØ3
-14E+2	-14BE+2
8388607E2	8388608E2

18.

NOTE

83886Ø7E2 is acceptable

8388607ØØ is not !!!

19. Exponential values can exceed the limit for integers.

(true/false) True

20. There is a limit, even for exponentials.

The maximum range of FACTOR numbers is (approximately)

2.7105E-20 ≤/N/≤ 9.2223E+18 where /N/ means the magnitude of N.

21. FACTOR stores all numbers as floating point numbers. Thus A= 100; B= 144B; C= 1E2; cause identical data to be stored in A, B and C.	
22. The mantissa of the floating point format is 16 (binary) bits. Therefore, the resolution of numbers stored in your FACTOR program is 1 part in 65535.	
23. The resolution of the floating point is much more than you will need for testing semiconductor devices. But if you insist on trying to add 65535 and 0.1, you can succeed in wasting the computer's (and your) time.	19. true
24. The numerical format resulting from WRITE statements is different, but simpler.	

25. ONLY FACTOR WRITE STATEMENTS TO THE LINE PRINTER OR THE VIDEO SCREEN ARE WITHIN THE SCOPE OF THIS LESSON.	
26. The general form for writing a numeric output is WRITE (LP) expression; to write to the line printer or WRITE (TTP) expression; to write to the video screen	
27. TTP is actually a mnemonic for TeleType Printer. Most systems use a VKT (Video Key-board Terminal) which is a compatible alternative to the teletype. Hence TTP can mean the video screen of the VKT.	
28. WRITE (LP) expression; or WRITE (TTP) expression; always writes a decimal number on the appropriate device	

29. consider:

```
SET PAGE 1;
A= 10B;
WRITE (LP) A;
END;
```

30. The program in frame 29 prints

8

on the line printer

31. consider:

```
SET PAGE 1;
B= -8;
WRITE (LP) B;
END;
```

32. The program in frame 31 writes

-ØØ8

on the line printer

33. The plus sign is suppressed for positive integer printouts. The minus sign is not. Integer printouts are limited to four characters.

34. The maximum positive number that the LP or TTP can print out is 9999.
The maximum negative is -999.

35. Consider:

```
SET PAGE 1;
X= 16384;
WRITE (TTP) X;
END;
```

36. The program in frame 35 writes

+1.638 E+Ø4

on the TTP.

37. Is 16384 an integer? yes

Does it print as an integer? No

38. How many characters were printed out in frame 36?

10

34.

9999
-999

39. Non-integer printouts always appear as a signed four-digit decimal number between 1.000 and 9.999 with a signed two digit exponent. Plus signs are not suppressed.

40. Consider:

```
SET PAGE 1;
WRITE (LP) 409.6;
END;
```

This program prints 4.096 E+02 on the line printer.

41. In frame 42, write the given number as it would appear on the line printer or video screen.

37.

yes

no

42.

Given	Output
-27B	-023
+21E1	210
13ØØØ	1.300 E+04
-6.Ø7	-6.070 E-00

38.

10

43. Numbers printed out are "rounded off", not truncated.

44.

END

(The answer to frame 42 is on the next sheet).

40.

+4.096E+02

42.

Given	Output
-27B	-Ø23
+21E1	21Ø
13ØØØ	+1.300E+Ø4
-6.07	-6.070E-ØØ

LESSON FOUR

TOPIC:

FACTOR ARITHMETIC EXPRESSIONS

GIVEN:

1. The Booklet
   FACTOR ARITHMETIC EXPRESSIONS
2. Pencil and eraser

PREREQUISITE: COMPLETION OF LESSONS ONE, TWO AND THREE

PERFORMANCE:

Student proceeds through the number frames responding to branching instructions.

Student writes short responses in the response frames, compares his responses with the data provided in answer frames and corrects his responses to agree with the furnished answers.

Standard: The student provides 100% correct responses within 40 minutes.

FACTOR

ARITHMETIC

EXPRESSIONS

1. Somewhere, at some time, you probably learned that XY is the product of a variable named X and a variable named Y. In other words, XY means X times Y.	
2. The FACTOR compiler recognizes XY as the name of a Variable location or of a symbolic statement identifier called a Label.	
3. The FACTOR multiplication operator is the asterisk (*). Thus X*Y means X times Y to the FACTOR compiler.	
4. Write the FACTOR statement that stores the product of R and T in location D. D=R*T;	

5. Write the FACTOR statement equivalent to the algebraic equation C = $2\pi r$ A = 3.14; C = 2*A*R;	
6. Frame 5 illustrates a very practical point. There is no point in forcing a machine to look up (access) a number unless you have a good reason for it. Accessing takes a finite amount of time.	2. variable label

7. Which is faster?
example a. C= 6.28*R;
example b. PI =3.14;
C=2*PI*R;
example A. C C = 6.28*R;	

8. Circle the incorrect FACTOR statements A = B*C; Z = MN; Q = 9T;	4. D = R*T;

9. HOLD IT!!! If you circled Z=MN in frame 8, you jumped to the conclusion that M and N were different variables. MN is a legitimate variable name. Look at the compiler's opinion.	5. C = 2*3.14*R; or C = 6.28*R; or PI = 3.14; C =2*PI*R; etc.
10. 000001 SET PAGE 1; 000002 A =B*C; 000003 Z = MN; 000004 Q =9T; EXPRESSION SYNTA^ 000005 END; 0001B COMPILATION ERRS	
11. FACTOR uses the traditional + for an addition operator and - for subtraction.	7. a
12. The algebraic equation A = B+C+D+E is easily programmed as A =B+C+D+E; Similarly F = G-H+I-K is programmed as F = G-H+I-K;	8. (Q =9T;)

13. The compiler scans an expression from left to right just as you (usually) do.

14. The compiler

ALWAYS

scans an expression from left to right.

15. Consider:

A = P+PRT, where P, R and T are separate variables.

You probably would "group" this equation like this

A = P (1 + RT)

16. But you wouldn't try to program that as

A = P*1 + R*T;

No! No! No! No! No!

17. The FACTOR compiler recognizes parentheses as grouping symbols.	
18. Write a FACTOR statement equivalent to A = P (1 + RT). A = P*(1+R*T);	
19. The FACTOR operator for division is the slash (/). That is, Percent = N÷100 can be programmed as PERCENT = N/100;	
20. Write the FACTOR statement equivalent to $X = \frac{A+B}{C+D}$ X=(A+B)/(C+D);	

21. The compiler Always scans an expression from Left to Right

22. The FACTOR operator for division is

"/" A slash

18.

A = P*(1+R*T);

23. Since the FACTOR operator for division cannot group terms you must use parentheses to program a fraction where the numerator or denominator is a polynomial.

24. $\frac{A + B}{C + D} \neq A + B/C + D$

You know it. I know it.

BUT

The FACTOR compiler will never know what that "thing" on the left is.

20.

X = (A + B) / (C + D);

25. The FACTOR operator for exponentiation is the up arrow (↑). That is Given: $X = \sqrt{Y}$ Program: X = Y ↑ (½) ;	21. always left right
26. Since NEGATION takes precedence over EXPONENTIATION never try to raise a negative number to a power. IF Y in frame 25 were negative, X would be "imaginary". FACTOR handles only real numbers.	22. /
27. Given: N = 3*4↑2; Result: 48 (not 144) is stored in N. Because exponentiation has precedence over multiplication	
28. The FACTOR compiler ALWAYS scans an expression from left to right but performs operations in order of precedence. Read the rules on the next page.	

RULES FOR EVALUATION OF FACTOR ARITHMETIC EXPRESSIONS

Arithmetic expressions are evaluated left-to-right according to the following rules:

1.) Parenthesized expressions are evaluated first. If parenthesized expressions are nested, the innermost expression is evaluated, then the next innermost until the entire expression has been evaluated.

2.) Within parenthesis and/or whenever parenthesis do not govern the order or evaluation, the hierarchy of operations in order of precedence is

   a.) Negation (NEG)

   b.) Exponentiation (↑)

   c.) Multiplication or division (*, /),

   d.) Addition or subtraction (+, -).

Example:

The expression

A* (Z-(B↑C)/T)) + VAL

is evaluated in the following sequence:

$B\uparrow C \rightarrow e_1$ NOTE: That is, B↑C is temporarily held as partial solution e_1.

$e_1 / T \rightarrow e_2$

$Z - e_2 \rightarrow e_3$

$e_3 * A \rightarrow e_4$

$e_4 + VAL \rightarrow e_5$

29. Program the equation

$A = P(1 + RT)^N$

where R is rate and T is time

A = P*((1+R*T)↑N);

30. CAUTION

The Sentry's computer does not handle "imaginary" numbers. Therefore B↑C is not allowed when B is negative, since the result could be "imaginary".

31. Circle the incorrect FACTOR statements

```
W = 5↑2;
X = -5↑2;
Y = -(5↑2);
Z = 5↑(-2);
```

32. Negation takes highest precedence! The minus sign in front of the 5 in the value assigned to X (frame 31) is a negation not a subtraction.

33. Compare frame 34 with frame 32.	29. A = P* (1+R*T)↑N;
34. Q = 15 - 3↑2; is just fine. Plus 9 is subtracted from 15. This time the minus sign is a subtraction operator.	
35. Negation can be programmed with the symbol "NEG" or with the minus sign in a context other than subtraction. Note examples in frame 36.	31. X = -5↑2
36. A = -6; A = NEG 6; C = D*NEG E; C = D* - E; C = D* (-E); All these examples are permissable but most programmers use only the first and last forms, avoiding the others as unnecessarily confusing.	

LESSON FIVE

TOPIC:

PIN DEFINITIONS

GIVEN:

1. The booklet: Pin Definitions
2. Pencil and eraser

PREREQUISITE:

Completion of Lesson Four

PERFORMANCE:

Student proceeds through the number frames responding to branching instructions.

Student writes short responses in the response frames, compares his responses with the data provided in answer frames and corrects his responses to agree with the furnished answers.

Standard: The student provides 100% correct responses within one hour.

PIN DEFINITIONS

A PROGRAMMED STUDY AID

SENTRY 600 PROGRAMMING

1725 Technology Drive
San Jose, California 95110

PIN DEFINITIONS

1. In this lesson, a pin refers to a land on a pin electronics card that mates with a connector on the performance board.	
2. A pin is a Land on an electronics card.	
3. There are two sets of identical pin electronics per card; hence, two electronically identical pins per card.	
4. Therefore, the pin electronics of: A "30-PIN" Sentry 600 requires 15 cards. A "60-PIN" Sentry 600 requires 30 cards.	

5. Since all pins are electronically identical, any pin can be programmed as either an input or an output pin.	
6. The terms input and output are relative to the DUT; an input pin connects an input to the DUT from the Sentry 600.	2. land
7. An input pin connects an input To the DUT From the Sentry 600.	
8. Input pins may be used to apply power, clocks or data to the DUT.	4. 15 30

9. Input pins may be used to apply power, clock or data to the DUT.	
10. Power pins provide a steady DC voltage to the DUT.	
11. Data pins normally apply NRZ (non-return-to-zero) data to the DUT.	7. to from
12. Data pins normally apply NRZ (non - return - to - zero) data to the DUT.	

13. The source of the data is local memory in the high speed test station controller.	9. power clocks data
14. The *source* of the data is *Local* *Memory* in the high speed test station controller.	
15. NRZ data stays at the programmed level for a period equal to that programmed by the SET PERIOD statement.	
16. NRZ data, stays at the programmed level for a period *Equal* to that programmed by the SET PERIOD statement.	12. non-return-to-zero

17. NRZ data stays at the programed Level for a period EQUAL to that programmed by the SET PERIOD statement.

18. Compare the statements in frame 19 with the waveforms in frame 20.

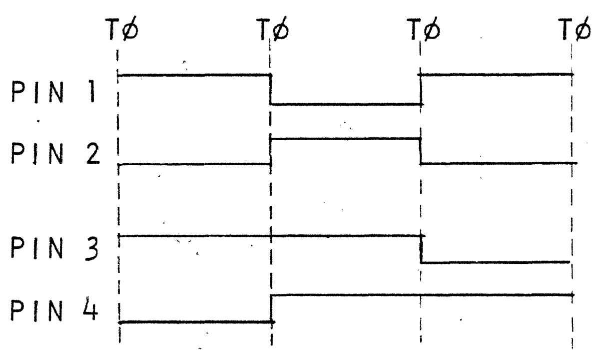
14.

source

local memory

19.
```
SET DA    1111 . .;
SET F     1Ø1Ø . .; REM 1st PERIOD DATA;
SET F     Ø111 . .; REM 2nd PERIOD DATA;
SET F     1ØØ1 . .; REM 3rd PERIOD DATA;
SET PERIOD 500E-9, RNGØ;
ENABLE TEST;
```

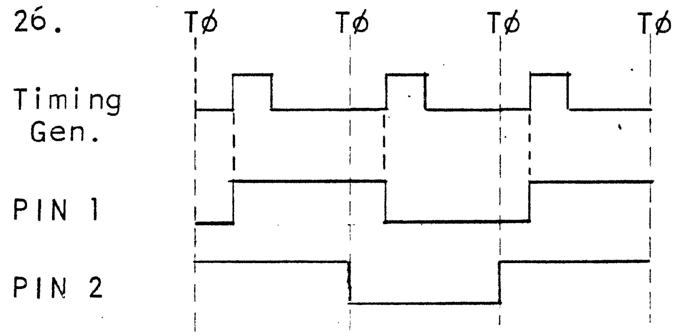
20. Assume positive logic



16.

equal

21. The simplest programming for a data pin, syncs the data with the programmed period. Review frames 19 and 20, noting the data transition points, then go on to frame 22.	17. programmed level equal SET PERIOD
22. The simplest programing case for a data pin applies (NRZ/RZ) NRZ data to the pin. This data is in synchronization with the Programed period.	
23. NRZ data can be synchronized with a timing generator by connecting a timing generator to the pin.	
24. NRZ data can be synchronized with a Timing Generator by connecting it to the pin.	

25. Frame 26 shows the same data on pins 1 and 2 but pin 1 is connected to a timing generator while pin 2 is not. Both pins are NRZ.	
26. 	22. NRZ programmed
27. Note that the data on pins 1 and 2 have the same period but different transition points.	
28. Transitions on pin 1 are synchronized by the leading edge of the *Timing Generator* pulse. Transitions on pin 2 occur at *TØ*.	24. timing generator

29. If no timing generator is connected to a data pin, data transitions occur at ___TØ___.	
30. If a timing generator is connected to an NRZ data pin, data transitions occur at ___positive edge of timing pulse___.	
31. NRZ data stays at the programmed level for a period equal to that programmed by the ___SET___ ___PERIOD___ statement.	
32. Machines with the "1-nanosecond option" provide a special case that can violate the statement in frame 31. This case is covered in the OPTIONS lesson.	28. timing generator TØ

33. Clock pins normally provide RZ (return-to-zero) waveforms to the DUT.	29. TØ
34. Clock pins normally provide RZ (return-To-zero) waveforms to the DUT.	30. the leading edge of the timing generator pulse.
35. Data can drive an RZ waveform to a "1" level only during the timing generator pulse. When the pulse ends, the waveform returns to zero.	31. SET PERIOD
36. Data can drive an RZ waveform to a "1" One level only during the Timing Generator pulse. When the pulse ends, the waveform returns to zero.	

37. Since a clock pin normally applies RZ data to the DUT and RZ data requires a timing generator, a <u>Timing Generator</u> must be connected to a clock pin.	
38. An input pin becomes a clock pin when its number appears in a CONN CLK pin list; statement.	34. <u>return-to-zero</u>
39. Input pins can provide <u>Power</u>, <u>DATA</u>, or <u>Clock</u> to the DUT.	
40. An input pin becomes a <u>Clock</u> pin when its number appears in a CONN CLK pin list; statement.	36. <u>"1"</u> <u>timing generator</u> <u>zero</u>

41. An input pin becomes a power pin when its number appears in a CONN [DPS1/DPS2/DPS3/TCOM] pin list; statement.	37. timing generator
42. A "1" must appear in the n^{th} position of either a SET DA or a SET DB pin-pattern; statement before pin n can be an input pin.	
43. SET DA 1001001; SET DB 0000111; Pins 1, 4, 5, 6 and 7 can be input pins.	39. power clocks data
44. Study frame 45 then respond to frame 46.	40. clock

45. SET DA 10011011; CONN CLK 1, 2, 4, 5; CONN DPS1 7; CONN TCOM 8; (Assume no SET DB statement)	
46. Pins 1, 4, and 5 are clock pins. Pins 7 and 8 are power pins. Pin 2 is incorrectly programmed.	
47. Data pins require more definition. Pin n can be a data pin only if a "1" appears in the n^{th} position of a SET [DA/DB] pin pattern; statement and the DA/DB register is enabled.	43. 1, 4, 5, 6 and 7
48. Pin n can be a data pin only if a 1 appears in the n^{th} position of a SET [DA/DB] pin pattern; statement and the DA/DB register is enabled.	

49. In order for an input pin to be a data pin, the DA/DB register defining the input pin must be Enabled.	
50. The DA register is enabled by the ENABLE DA; statement. The DB register is enabled by the ENABLE DB; statement. The enable statement applies to the memory loads that follow it.	46. 1, 4, 5 / 7, 8 / 2
51. If no ENABLE [DA/DB] ; statement appears in a FACTOR program, DA is enabled by default. If an ENABLE [DA/DB] statement appears, it prevails until the next one appears.	
52. No ENABLE DA; statement is required to enable the DA register unless an Enable DB; statement is currently programmed.	48. "1" / DA/DB

53. A power or clock pin must have a "1" in the pin pattern of either the DA or the DB register but a data pin must have a "1" in the pin pattern of the *Enabled* D register.	49. enabled
54. A power or clock pin must have a "1" in the pin pattern of *Either* the DA *OR* the DB register, but a *DATA* pin must have a "1" in the pin pattern of the *ENAbled.* D register.	
55. Power, clock and data pins are mutually exclusive. The last definition in the FACTOR program prevails.	
56. Therefore, if a pin number appears in the pin list of a CONN [DPS1/DPS2/DPS3/TCOM/CLK] pinlist; statement it cannot be a data pin.	52. ENABLE DB

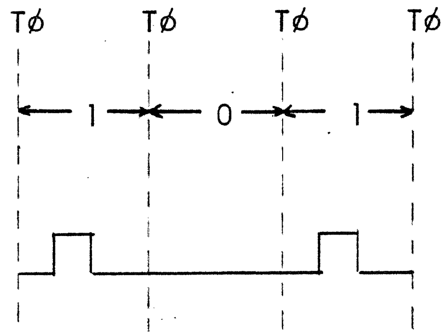
57. The data shown on pin 4 in frame 58 is (RZ/NRZ) RZ data.

53. enabled

58.

DATA IN Local Memory

PIN 4



54. either or data enabled

59. Pin 4 could be either a DATA or a Clock pin depending upon whether or not 4 appears in a CONN CLK pinlist; statement.

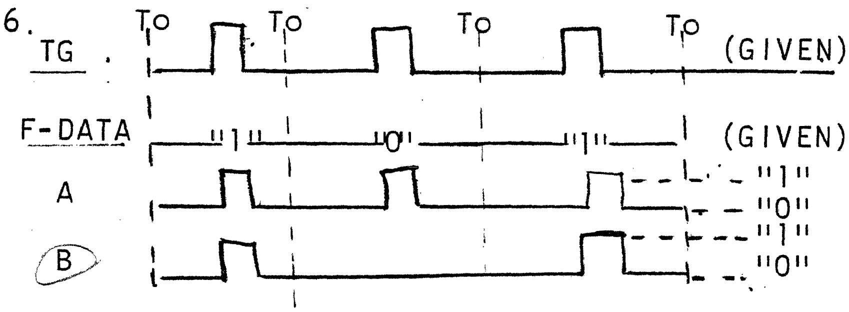
60. An RZ pulse can appear during a data period only if the data for that period is a logical 1.

61. Since data pins are normally provided with NRZ waveforms, a SET RZ pin pattern; statement is necessary to apply RZ waveforms to a data pin.	57. RZ
62. A logical "1" in the pin pattern of a SET RZ statement defines a pin as RZ, a "Ø" defines the pin as NRZ.	
63. SET RZ 0011 . . . .; The above statement defines pins __1__ and __2__ as NRZ and pins __3__ and __4__ as __RZ__.	59. clock data
64. An RZ pin receives a pulse only during a period where F=data is a (one/zero) _1_.	60. 1

65. Choose the waveform in frame 66 that is correct for an RZ pin (i.e. choose A or B).

B

66.



67. An RZ pin receives a pulse only during a period where F-data is a 1.

63.

1 and 2
3 and 4 as RZ

68. Circle A, B, or C.

The width of an RZ pulse is determined by:
A. The programmed period.
B. The width of the timing generator pulse.
C. The F-data

64.

one

69. A data pin IS NRZ unless a "1" is programmed for the pin in SET RZ pinlist.	
70. A clock pin is normally (RZ/NRZ) RZ.	66. B (An RZ pulse can appear during a data period only if the data for that period is a logical one.)
71. The CONN CLK pinlist; statement automatically loads a "1" into the RZ register for every pin specified in the pinlist.	67. one
72. Data pins are normally (RZ/NRZ) NRZ but clock pins are normally (RZ/NRZ) RZ.	68. B. The width of the timing generator pulse.

73. ~~Clock pins~~ can be redefined as NRZ by programming a SET RZ pinlist; statement after the CONN CLK pinlist; statement.	69. RZ
74. Clock pins can be redefined as NRZ by programming a (one/zero) zero into the SET RZ pinlist; statement (before/after) After the appropriate CONN CLK pinlist; statement.	70. RZ
75. Data transitions for NRZ pins, with no timing generator connected to them, are at T∅ .	
76. Data transitions for NRZ pins, with a timing generator connected to them, are at the Leading Edge of the Timig generator pulse	72. NRZ RZ

77. Pulses are applied to RZ pins only during the programmed width of a timing generator pulse and only if the F-data for the pin is a logical 1.	
78. Each pin can be programmed as either an input or an output pin	74. zero after
79. Each pin is connected to a functional test comparator during functional testing.	75. TØ
80. The functional test comparator can be programmed to perform a pass/fail test on each pin during each period of a functional test.	76. leading edge timing generator pulse

81. Usually test data is desired only for output pins.	77. width one
82. Any pin that is not an input pin can be considered an output pin.	78. output
83. A pin that is neither a power pin nor a clock pin and that does not have a "1" assigned in the pin pattern of an enabled D register is an *output* pin.	
84. Functional test data for a pin is ignored unless a "1" appears for that pin in an enabled MA or MB register.	

85. If no ENABLE [MA/MB]; statement is programmed, the MA register is enabled by default. Once programmed, an ENABLE [MA/MB] ; statement prevails for subsequent memory loads until replaced by another ENABLE [MA/MB]	
86. Definition [DA/DB] and Mask [MA/MB]; registers can be enabled together in one statement or separately. See frame 87.	
87. Possible valid enable statements ENABLE MA; ENABLE DA ENABLE MA, DA; ENABLE DA, MA; ENABLE MA, DB; ENABLE DA, MB; ENABLE MB; ENABLE DB; ENABLE MB, DA; ENABLE DB, MA; ENABLE MB, DB; ENABLE DB, MB;	83. output
88. Only the mask register that is *Enabled* specifies the pins to be tested.	

	88. <u>enabled</u>

LESSON SIX

TOPIC:

LOCAL MEMORY MANAGEMENT

GIVEN:

1. The booklet: Local Memory Management
2. Pencil and eraser

PREREQUISITE:

Completion of Lesson Five

PERFORMANCE:

Student proceeds through the number frames responding to branching instructions.

Student writes short responses in the response frames, compares his responses with the data provided in answer frames and corrects his responses to agree with the furnished answers.

Standard: The student provides 100% correct responses within one hour.

LOCAL MEMORY MANAGEMENT

A PROGRAMMED STUDY AID

SENTRY 600 PROGRAMMING

1725 Technology Drive
San Jose, California 95110

LOCAL MEMORY MANAGEMENT

1. Local memory locations are loaded sequentially by a series of SET F statements. (REF: FACTOR MANUAL paragraph 11.4.5)	
2. A series of SET F statements load local memory Locations (how?) sequentially	
3. Each memory load starts at address Ø, unless modified by an AT statement.	
4. Unless modified by an AT statement, each memory load starts at address Ø.	

5. A memory load consisting of 16 SET F statements loads local memory locations 0 through 15.	
6. The first SET F (or SET FI) following an ENABLE TEST statement starts a new memory load.	2. sequentially
7. If sixteen memory locations are programmed followed by ENABLE TEST, the next SET F statement loads address Ø.	
8. Unless modified by an AT statement, each memory load starts at address Ø.	4. Ø

9. An AT statement can specify an address to be loaded. For example AT 11; SET F pin pattern; loads memory location eleven (decimal). (REF: FACTOR MANUAL paragraph 11·4·23)	5. Ø 15
10. The AT statement can be used to start a new memory load at an address other than Ø. Subsequent loads are sequential. Note the programming and remarks in frame 11.	
11. ENABLE TEST; REM THE PREVIOUS TEST LOADED 16 LOCATIONS; AT 11; SET F pin pattern; REM THIS LOADS LOCATION 11; SET F pin pattern; REM THIS LOADS LOCATION 12; SET F pin pattern; REM THIS LOADS LOCATION 13; ENABLE TEST;	7. Ø
12. The programming in frame 11 modified the contents of locations 11, 12 and 13 but did not change locations Ø through 1Ø, 14, and 15.	Ø

13. The first SET F statement of a memory load or following an AT statement programs all pins to zero that are not specified as logical ones. Examples are given in frame 14.

14.
```
SET PAGE 100; REM 30-PIN SYSTEM;
.... set-up instructions ....
ENABLE DA, MA;
SET F 0010111; REM 3,5, 6 and 7 PRO-
               TO ONES, THE OTHER 26
               PINS ARE PROGRAMMED TO
               ZEROES;
      .
      .
      .
ENABLE TEST;
AT 10; SET F 1; REM F-DATA IS ONE
FOLLOWED BY 29 ZEROES;
```

15. The first SET F statement of a memory load or after an AT statement programs all pins to zero that are not specified as logical 1's.

16. Subsequent SET F statements within one memory load need specify only the change from one location to the next. Note the example in frame 17.

17. Sequential SET F's

	LOC	F-DATA (30 PINS)					
SET F ØØØØ1;	Ø	ØØØØ1	ØØØØØ	ØØØØØ	ØØØØØ	ØØØØØ	ØØØØØ
SET F ØØØ1;	1	ØØØ11	ØØØØØ	ØØØØØ	ØØØØØ	ØØØØØ	ØØØØØ
SET F ØØ1;	2	ØØ111	ØØØØØ	ØØØØØ	ØØØØØ	ØØØØØ	ØØØØØ
SET F Ø1;	3	Ø1111	ØØØØØ	ØØØØØ	ØØØØØ	ØØØØØ	ØØØØØ
SET F 1;	4	11111	ØØØØØ	ØØØØØ	ØØØØØ	ØØØØØ	ØØØØØ

18. During compilation of a FACTOR program, each statement that controls the tester is converted into a code that can be executed by the Sentry 600.

19. During execution, this executable code causes the pin pattern of each SET F statement to be loaded into local memory as F-Data.

15.

AT

ones

20. Two enable bits are loaded into the local memory along with the F-Data. One enables a D register. The other enables an M register.

21. Two enable bits are loaded into local memory along with the DATA. One enables a D register. The other enables an M register.

22. The state of these two enable bits is determined, at compile time, ENABLE [MA/MB/DA/DB] (,MA/MB/DA/DB); statement.

23. The state of these two enable bits is determined at compile time.

24. Thus, the ENABLE MA etc. statement does not appear as executable code in the compiled program. It has already caused the compiler to modify the code that loads local memory.

25. Since ENABLE MA etc. produces no executable code it does not exist in the executable DATA file.	21. F-Data D M
26. Since it is impossible to branch to a statement that does not exist at execution time, an ENABLE MA etc. statement should not be labelled.	
27. An ENABLE MA etc statement should *NOT* be labelled.	23. Compile
28. Any executable statement written within a string of SET F statements starts a new memory load.	

29. Any executable statement written within a string of SET F statements starts a new local memory load.	
30. Note the remarks Delete SET F ......; REM LOAD LOC Ø; SET F ......; REM LOAD LOC 1; SET F ......; REM LOAD LOC 2; If A EQ 2 THEN SET F ...... ELSE SET F ......; ENABLE TEST;	
31. In frame 30, note that the first location of local memory is changed if A equals 2 but the second location (LOC 1) is changed if A does not equal 2. Delete	27. not
32. BUT ENABLE [MA/MB/DA/DB] (,MA/MB/DA/DB/) generates no executable code Therefore it CAN be nested within a string of SET F statements without starting a new load.	

33. A D or an M enable CAN (can/cannot) be loaded within a string of SET F statements without starting a new memory load.	29. new
34. This characteristic of the D or M enable allows changing pin definition (data/output) and mask definition (care/ don't care) during execution of a functional test.	
35. When the Test Operating System executes the ENABLE TEST; statement the F-Data are executed at the high-speed test rate.	
36. F-Data are executed at the high-speed test rate.	

37. The high-speed test rate is programmed by the SET PERIOD statement. This statement must appear before the first ENABLE TEST; in a program. (REF: FACTOR MANUAL paragraph 11.4.17)	33. Can
38. The functional test rate is programmed by the SET PERIOD statement.	
39. The shortest period is determined by the system capability. Thus, minimum period for a 10 MHZ system is 100 nanoseconds; minimum period for a 5 MHZ system is 200 nanoseconds.	
40. Rate (frequency) and period are reciprocals, i.e. $\text{frequency} = \frac{1}{\text{period}}$; $\text{period} = \frac{1}{\text{frequency}}$ $\text{HERTZ} = \frac{1}{\text{seconds}}$; $\text{seconds} = \frac{1}{\text{Hertz}}$	36. high-speed

41. A test rate of 2 MHZ requires a programmed period of 500 ns. $\frac{1}{2\times10^6} = .5\times10^{-6} = 500\ ns$	
42. When the functional test takes place, the first local memory word executed is that in location Ø, unless a SET START statement appeared since the last ENABLE TEST; statement.	38. SET PERIOD
43. During functional test execution the first F-data is from memory location Ø unless a SET START statement appeared since the last ENABLE TEST; statement. (REF: FACTOR MANUAL paragraph 11·4·22)	
44. A SET START statement does not "carry-over" from one functional test to the next. That is, if the first functional test has a SET START statement but the second does not, the second test starts at location zero.	

45. A SET START statement does NOT (does/does not) carry over from one functional test to the next..	41. 500 nanoseconds
46. Note the difference between the AT and the SET START statements. An AT statement may control the first location loaded. A SET START statement controls the first location executed.	
47. An AT statement may control the first location loaded. A SET START statement controls the first statement executed.	43. Ø
48. It is possible to modify part of local memory using the AT statement but execute all of local memory by not using SET START. Consider the remarks in frame 49.	

49. ENABLE TEST; REM THIS TEST USED LOCATIONS Ø THRU 15; AT 12; SET F ....; SET F ....; SET F ....; SET F ....; ENABLE TEST; REM TEST STARTS AT LOCATION Ø;	45. does not
50. If frame 49 were modified by a SET START 12; statement placed just before the AT or just before the last ENABLE TEST; the test would execute locations 12 through 15 without using or changing locations Ø thru 11.	
51. A functional test starts at location Ø unless modified by a SET START statement.	47. loaded
52. Unless a SET MAJOR statement is programmed, the last local memory location executed for a specific load is the highest location loaded during that load. (REF: FACTOR MANUAL paragraph 6.5.2.2)	

53. Unless a SET MAJOR statement is pro=grammed, the last memory location executed for a specific load is the highest location loaded during that load.	
54. Unless a SET MAJOR statement is programmed, the last memory location executed for a specific load is the highest location loaded during THAT load.	
	51. SET START

	53. highest
	54. SET MAJOR highest that