APPENDIX F

THE FORTRAN/8 COMPILER

```
DECLARE NSY LITERALLY '105', NT LITERALLY '47';
DECLARE V(NSY) CHARACTER INITIAL ( '<ERROR: TOKEN = 0>', ';', '(', ')', ',',
    '+', '-', '/', '*', '=', 'IF', 'DO', 'GO', 'TO', '_|_', 'END', 'ABS',
    'SQR', 'TAB', 'STOP', 'SQRT', '.OR.', '.LT.', '.LE.', '.EQ.', '.NE.',
    '.GT.', '.GE.', 'GOTO', 'REAL', 'DATA', 'CALL', 'READ', 'FLOAT', '.NOT.',
    '.AND.', 'WRITE', 'RETURN', 'COMMON', 'INTEGER', 'WRITEON', '<NUMBER>',
    'CONTINUE', 'FUNCTION', '<STRING>', 'DIMENSION', 'SUBROUTINE',
    '<IDENTIFIER>', '<IF>', '<TERM>', '<GOTO>', '<DATA>', '<CALL>', '<PAREN>',
    '<COMMA>', '<LABEL1>', '<LABEL2>', '<END GO>', '<PROGRAM>', '<PRIMARY>',
    '<DO HEAD>', '<PRIMARY*>', '<VARIABLE>', '<RELATION>', '<DO LABEL>',
    '<STATEMENT>', '<SECONDARY>', '<DATA HEAD>', '<READ HEAD>',
    '<LOGICAL IF>', '<EXPRESSION>', '<RIGHT PART>', '<WRITE HEAD>',
    '<DO VARIABLE>', '<GO TRANSFER>', '<DECLARATION>', '<IF STATEMENT>',
    '<DO STATEMENT>', '<GO STATEMENT>', '<DOUBLE LABEL>', '<BOOLEAN TERM>',
    '<ARITHMETIC IF>', '<VARIABLE LIST>', '<MASTER PROGRAM>',
    '<STATEMENT LIST>', '<READ STATEMENT>', '<SUBSCRIPT HEAD>',
    '<PROCEDURE HEAD>', '<PROCEDURE TYPE>', '<TAB EXPRESSION>',
    '<STATEMENT BLOCK>', '<PROCEDURE BLOCK>', '<BASIC STATEMENT>',
    '<SUBROUTINE CALL>', '<WRITE STATEMENT>', '<BOOLEAN PRIMARY>',
    '<DECLARATION LIST>', '<DECLARATION TYPE>', '<DATA DECLARATION>',
    '<LABELED STATEMENT>', '<PROCEDURE HEADING>', '<BOOLEAN EXPRESSION>',
    '<LOGICAL EXPRESSION>', '<PARAMLESS PROCEDURE>', '<ASSIGNMENT STATEMENT>',
    '<PROCEDURE & PARAMETERS>');
DECLARE V_INDEX(12) BIT(8) INITIAL ( 1, 10, 14, 19, 33, 37, 39, 41, 45, 46,
    47, 47, 48);
DECLARE C1(NSY) BIT(96) INITIAL (
    "(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 000",
    "(2) 00000 00000 22202 20002 00000 00022 22200 02222 22020 222",
    "(2) 00300 33000 00000 03320 30000 00000 00033 00000 03002 003",
    "(2) 03023 22222 00200 00002 02222 22220 02200 22200 21000 002",
    "(2) 00300 33000 00000 03320 30000 00000 00030 00000 03002 003",
    "(2) 00100 00000 00000 01100 10000 00000 00010 00000 01000 001",
    "(2) 00100 00000 00000 01100 10000 00000 00010 00000 01000 001",
    "(2) 02100 00000 00000 01100 10000 00000 00010 00000 03000 001",
    "(2) 00300 00020 00000 03300 30000 00000 00030 00000 03000 003",
    "(2) 00100 11000 00000 01100 10000 00000 00010 00000 01000 001",
    "(2) 00100 00000 00000 00000 00000 00000 00000 00000 00000 000",
    "(2) 00000 00000 00000 00000 00000 00000 00000 00000 01000 000",
    "(2) 00000 00000 00010 00000 00000 00000 00000 00000 00000 000",
    "(2) 00200 00000 00000 00000 00000 00000 00000 00000 02000 000",
    "(2) 00000 00000 11100 00001 00000 00011 11100 01111 11010 111",
    "(2) 01000 00000 00000 00000 00000 00000 00000 00000 00000 000",
    "(2) 00100 00000 00000 00000 00000 00000 00000 00000 00000 000",
```

```
"(2) 00100 00000 00000 00000 00000 00000 00000 00000 00000 000",
"(2) 00100 11000 00000 01100 10000 00000 00010 00000 01000 001",
"(2) 02000 00000 00000 00000 00000 00000 00000 00000 00000 000",
"(2) 00100 00000 00000 00000 00000 00000 00000 00000 00000 000",
"(2) 00100 11000 00000 01100 10000 00000 00011 00000 01000 001",
"(2) 00200 22000 00000 02200 20000 00000 00020 00000 02000 002",
"(2) 00200 22000 00000 02200 20000 00000 00020 00000 02000 002",
"(2) 00200 22000 00000 02200 20000 00000 00020 00000 02000 002",
"(2) 00200 22000 00000 02200 20000 00000 00020 00000 02000 002",
"(2) 00200 22000 00000 02200 20000 00000 00020 00000 02000 002",
"(2) 00200 22000 00000 02200 20000 00000 00020 00000 02000 002",
"(2) 00200 00000 00000 00000 00000 00000 00000 00000 02000 000",
"(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 002",
"(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 002",
"(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 002",
"(2) 00100 00000 00000 00000 00000 00000 00000 00000 00000 000",
"(2) 00100 00000 00000 00000 00000 00000 00000 00000 00000 000",
"(2) 00100 11000 00000 01100 10000 00000 00010 00000 01000 001",
"(2) 00100 11000 00000 01100 10000 00000 00010 00000 01000 001",
"(2) 00100 00000 00000 00000 00000 00000 00000 00000 00000 000",
"(2) 02000 00000 00000 00000 00000 00000 00000 00000 00000 000",
"(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 002",
"(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 002",
"(2) 00100 00000 00000 00000 00000 00000 00000 00000 00000 000",
"(2) 02023 22320 22200 00002 02222 22220 02200 22200 22200 002",
"(2) 02000 00000 00000 00000 00000 00000 00000 00000 00000 000",
"(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 001",
"(2) 00011 00000 00000 00000 00000 00000 00000 00000 00000 000",
"(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 002",
"(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 001",
"(2) 03133 22222 00000 00000 02222 22200 00000 20000 00000 000",
"(2) 00100 11000 00000 01100 10000 00000 00011 00000 01000 001",
"(2) 02022 22100 00000 00000 02222 22200 00000 20000 00000 000",
"(2) 00100 00000 00000 00000 00000 00000 00000 00000 01000 000",
"(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 001",
"(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 001",
"(2) 00000 00000 00000 00000 00000 00000 00000 00000 01000 000",
"(2) 00000 00000 00000 00000 00000 00000 00000 00000 01000 000",
"(2) 00000 00000 00000 00000 00000 00000 00000 00000 03000 000",
"(2) 00000 00000 11100 00001 00000 00010 01100 01100 11100 001",
"(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 001",
"(2) 00000 00000 11102 00001 00000 00011 11100 01111 11010 111",
"(2) 02022 22200 00000 00000 02222 22200 00000 20000 00000 000",
"(2) 02001 00000 00000 00000 00000 00000 00000 00000 00000 000",
"(2) 00100 00010 00000 01100 10000 00000 00010 00000 01000 001",
"(2) 02033 22221 00000 00000 02222 22200 00000 20000 00000 000",
"(2) 00100 11000 00000 01100 10000 00000 00010 00000 01000 001",
"(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 001",
```

```
    "(2) 03000 00000 00000 00000 00000 00000 00000 00000 00000 000",
    "(2) 02022 22210 00000 00000 02222 22200 00000 20000 00000 000",
    "(2) 00000 00100 00000 00000 00000 00000 00000 00000 00000 000",
    "(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 001",
    "(2) 00000 00000 00100 00001 00000 00010 01100 01100 10000 001",
    "(2) 02033 11000 00000 00000 02111 11100 00000 20000 00000 000",
    "(2) 02000 00000 00000 00000 00000 00000 00000 00000 00000 000",
    "(2) 00100 11000 00000 01110 10000 00000 00010 00000 01001 001",
    "(2) 00001 00000 00000 00000 00000 00000 00000 00000 00000 000",
    "(2) 00011 00000 00000 00000 00000 00000 00000 00000 00000 000",
    "(2) 01000 00000 00000 00000 00000 00000 00000 00000 00000 000",
    "(2) 02000 00000 00000 00000 00000 00000 00000 00000 00000 000",
    "(2) 02000 00000 00000 00000 00000 00000 00000 00000 00000 000",
    "(2) 02000 00000 00000 00000 00000 00000 00000 00000 00000 000",
    "(2) 00000 00000 00000 00000 00000 00000 00000 00000 01000 000",
    "(2) 00020 00000 00000 00000 02000 00000 00000 10000 00000 000",
    "(2) 00010 00000 00000 00000 00000 00000 00000 00000 00000 000",
    "(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 001",
    "(2) 00000 00000 00002 00000 00000 00000 00000 00000 00000 000",
    "(2) 00000 00000 11100 20001 00000 00010 01100 01100 11000 001",
    "(2) 02000 00000 00000 00000 00000 00000 00000 00000 00000 000",
    "(2) 00100 11000 00000 01100 10000 00000 00010 00000 01000 001",
    "(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 001",
    "(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 002",
    "(2) 00011 00000 00000 00000 00000 00000 00000 00000 00000 000",
    "(2) 00000 00000 00000 10000 00000 00000 00000 00000 00000 000",
    "(2) 00000 00000 11100 00001 00000 00010 01100 01100 11000 001",
    "(2) 02000 00000 00000 00000 00000 00000 00000 00000 00000 000",
    "(2) 02000 00000 00000 00000 00000 00000 00000 00000 00000 000",
    "(2) 02000 00000 00000 00000 00000 00000 00000 00000 00000 000",
    "(2) 00020 00000 00000 00000 02000 00000 00000 20000 00000 000",
    "(2) 00000 00000 33300 00003 00000 00031 13300 03311 33000 103",
    "(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 001",
    "(2) 00000 00000 00000 00000 00000 00000 00000 00000 01000 000",
    "(2) 02000 00000 00000 00000 00000 00000 00000 00000 00000 000",
    "(2) 00000 00000 22200 00002 00000 00021 12200 02211 22000 102",
    "(2) 00010 00000 00000 00000 01000 00000 00000 00000 00000 000",
    "(2) 00020 00000 00000 00000 02000 00000 00000 20000 00000 000",
    "(2) 00000 00000 22200 00002 00000 00022 22200 02222 22000 202",
    "(2) 02000 00000 00000 00000 00000 00000 00000 00000 00000 000",
    "(2) 00000 00000 22200 00002 00000 00022 22200 02222 22000 202");
DECLARE NC1TRIPLES LITERALLY '259';
DECLARE C1TRIPLES(NC1TRIPLES) FIXED INITIAL ( 131586, 131589, 131590, 131600,
    131601, 131604, 131617, 131618, 131625, 131631, 148995, 207108, 210729,
    262658, 262661, 262662, 262672, 262673, 262676, 262689, 262697, 262703,
    328194, 328197, 328198, 328208, 328209, 328212, 328225, 328233, 328239,
    393730, 393733, 393734, 393744, 393745, 393748, 393761, 393769, 393775,
    459266, 459269, 459270, 459280, 459281, 459284, 459297, 459305, 459311,
```

```
    524802, 524805, 524806, 524816, 524817, 524820, 524833, 524841, 524847,
    590338, 590341, 590342, 590352, 590353, 590356, 590369, 590377, 590383,
    934145, 942090, 942091, 942092, 942099, 942108, 942111, 942112, 942116,
    942117, 942120, 942121, 942127, 1049090, 1049093, 1049094, 1049104,
    1049105, 1049108, 1049121, 1049129, 1049135, 1114626, 1114629, 1114630,
    1114640, 1114641, 1114644, 1114657, 1114665, 1114671, 1180162, 1180165,
    1180166, 1180176, 1180177, 1180180, 1180193, 1180201, 1180207, 1311234,
    1311237, 1311238, 1311248, 1311249, 1311252, 1311265, 1311273, 1311279,
    1376770, 1376773, 1376774, 1376784, 1376785, 1376788, 1376801, 1376802,
    1376809, 1376815, 2163202, 2163205, 2163206, 2163216, 2163217, 2163220,
    2163233, 2163241, 2163247, 2228738, 2228741, 2228742, 2228752, 2228753,
    2228756, 2228769, 2228770, 2228777, 2228783, 2294274, 2294277, 2294278,
    2294288, 2294289, 2294292, 2294305, 2294306, 2294313, 2294319, 3026689,
    3080961, 3146242, 3146245, 3146246, 3146256, 3146257, 3146260, 3146273,
    3146274, 3146281, 3146287, 3213097, 3358212, 3614980, 3817729, 3825674,
    3825675, 3825676, 3825683, 3825692, 3825695, 3825696, 3825700, 3825701,
    3825704, 3825705, 3825711, 3933186, 3933189, 3933190, 3933200, 3933201,
    3933204, 3933217, 3933225, 3933231, 3998210, 3998213, 3998214, 3998224,
    3998225, 3998228, 3998241, 3998249, 3998255, 3999746, 3999760, 3999761,
    3999764, 3999777, 3999785, 3999791, 4129282, 4129285, 4129286, 4129296,
    4129297, 4129300, 4129313, 4129321, 4129327, 4472323, 4472324, 4719106,
    4719109, 4719110, 4719120, 4719121, 4719124, 4719137, 4719145, 4719151,
    4736515, 4736516, 4785154, 4785157, 4785158, 4785168, 4785169, 4785172,
    4785185, 4785193, 4785199, 4850436, 5389828, 5521665, 5636610, 5636613,
    5636614, 5636624, 5636625, 5636628, 5636641, 5636649, 5636655, 5654019,
    5654020, 5713667, 5713668, 5980417, 6308097, 6372868, 6433028, 6433031);
DECLARE PRTB(122) FIXED INITIAL (0, 3824143, 5713667, 23055, 11823, 24651,
    21569, 75, 65, 11055, 11823, 32, 10, 36, 40, 47, 0, 1049158, 1114694,
    1311302, 2163270, 17470, 18502, 18476, 18521, 613, 12389, 582, 22086,
    21054, 22319, 25129, 17470, 18502, 18476, 18521, 22086, 3, 41, 62, 0,
    25129, 67, 66, 12, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 20815, 12853,
    18998, 11, 50, 0, 0, 56, 0, 0, 17925, 17926, 5, 6, 0, 823, 0, 12551, 61,
    0, 0, 24914, 13138, 19001, 97, 51, 52, 0, 56, 15624, 0, 4210185, 15364,
    18692, 17983, 9, 18, 48, 2366, 62, 0, 0, 0, 25877, 0, 96, 91, 0, 0, 0, 69,
    0, 0, 0, 20515, 34, 0, 100, 0, 0, 0, 0, 0, 0);
DECLARE PRDTB(122) BIT(8) INITIAL (0, 3, 104, 2, 103, 84, 8, 83, 7, 107, 108,
    112, 23, 117, 118, 67, 80, 40, 41, 42, 43, 111, 114, 115, 116, 51, 44, 39,
    66, 93, 106, 95, 113, 119, 120, 121, 68, 82, 25, 92, 81, 87, 94, 36, 76,
    19, 53, 54, 55, 56, 57, 58, 77, 90, 98, 110, 18, 91, 89, 20, 78, 79, 73,
    74, 61, 38, 60, 88, 65, 27, 28, 29, 30, 26, 24, 1, 33, 32, 31, 69, 86, 97,
    75, 85, 96, 109, 37, 59, 35, 34, 72, 70, 71, 52, 63, 122, 22, 64, 62, 9,
    11, 14, 46, 45, 5, 6, 4, 16, 105, 21, 10, 15, 17, 49, 48, 47, 100, 12, 99,
    50, 101, 13, 102);
DECLARE HDTB(122) BIT(8) INITIAL (0, 58, 105, 58, 103, 96, 84, 96, 84, 88,
    88, 68, 48, 72, 72, 86, 53, 66, 66, 66, 85, 94, 94, 94, 95, 69, 66,
    62, 82, 87, 98, 68, 72, 72, 72, 86, 57, 55, 82, 54, 75, 98, 61, 50, 92,
    63, 63, 63, 63, 63, 63, 50, 97, 51, 52, 92, 97, 97, 76, 74, 74, 64, 78,
    56, 66, 99, 97, 62, 70, 70, 70, 70, 70, 79, 83, 49, 49, 49, 77, 75, 67,
```

```
   78, 75, 67, 93, 66, 99, 59, 59, 73, 77, 60, 102, 71, 89, 81, 71, 104, 65,
   65, 92, 101, 101, 90, 90, 90, 92, 87, 76, 65, 92, 92, 80, 80, 80, 91, 65,
   91, 95, 100, 92, 100);
DECLARE PRLENGTH(122) BIT(8) INITIAL (0, 4, 4, 3, 3, 3, 3, 2, 2, 3, 3, 2, 2,
   2, 2, 2, 1, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2,
   2, 2, 1, 3, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 2,
   2, 1, 1, 2, 1, 1, 3, 3, 2, 2, 1, 3, 1, 3, 2, 1, 1, 3, 3, 3, 2, 2, 2, 1, 2,
   3, 1, 4, 3, 3, 3, 2, 2, 2, 3, 2, 1, 1, 1, 3, 1, 2, 2, 1, 1, 1, 2, 1, 1, 1,
   3, 2, 1, 2, 1, 1, 1, 1, 1, 1);
DECLARE CONTEXT_CASE(122) BIT(8) INITIAL (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
DECLARE LEFT_CONTEXT(0) BIT(8) INITIAL ( 0);
DECLARE LEFT_INDEX(58) BIT(8) INITIAL ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1);
DECLARE CONTEXT_TRIPLE(0) FIXED INITIAL ( 0);
DECLARE TRIPLE_INDEX(58) BIT(8) INITIAL ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1);
DECLARE PR_INDEX(105) BIT(8) INITIAL ( 1, 9, 17, 29, 41, 41, 41, 43, 44, 44,
   44, 44, 44, 45, 45, 45, 45, 45, 45, 46, 46, 46, 47, 48, 49, 50, 51, 52,
   53, 54, 55, 56, 56, 56, 56, 56, 56, 57, 58, 59, 59, 66, 67, 67, 67, 68,
   68, 69, 69, 74, 74, 74, 74, 74, 74, 75, 75, 75, 76, 79, 80, 80, 87, 87,
   87, 88, 90, 90, 90, 90, 97, 99, 99, 99, 99, 99, 100, 101, 102, 102, 104,
   104, 104, 104, 107, 108, 108, 108, 109, 109, 109, 109, 111, 112, 113, 116,
   117, 117, 117, 118, 119, 119, 120, 121, 122, 123);

/*  END OF CARDS PUNCHED BY SYNTAX                                          */

/*  DECLARATIONS FOR THE SCANNER                                            */

/* TOKEN IS THE INDEX INTO THE VOCABULARY V() OF THE LAST SYMBOL SCANNED,
   CP IS THE POINTER TO THE LAST CHARACTER SCANNED IN THE CARDIMAGE,
   BCD IS THE LAST SYMBOL SCANNED (LITERAL CHARACTER STRING). */
DECLARE (TOKEN, CP) FIXED, BCD CHARACTER;

/* SET UP SOME CONVENIENT ABBREVIATIONS FOR PRINTER CONTROL */
DECLARE EJECT_PAGE LITERALLY 'OUTPUT(1) = PAGE',
   PAGE CHARACTER INITIAL ('1'), DOUBLE CHARACTER INITIAL ('0'),
   DOUBLE_SPACE LITERALLY 'OUTPUT(1) = DOUBLE',
   X70 CHARACTER INITIAL ('
                          ');

/* LENGTH OF LONGEST SYMBOL IN V */
```

```
DECLARE (RESERVED_LIMIT, MARGIN_CHOP) FIXED;

/* CHARTYPE() IS USED TO DISTINGUISH CLASSES OF SYMBOLS IN THE SCANNER.
   TX() IS A TABLE USED FOR TRANSLATING FROM ONE CHARACTER SET TO ANOTHER.
   CONTROL() HOLDS THE VALUE OF THE COMPILER CONTROL TOGGLES SET IN $ CARDS.
   NOT_LETTER_OR_DIGIT() IS SIMILIAR TO CHARTYPE() BUT USED IN SCANNING
   IDENTIFIERS ONLY.

   ALL ARE USED BY THE SCANNER AND CONTROL() IS SET THERE.
*/
DECLARE (CHARTYPE, TX) (255) BIT(8),
        (CONTROL, NOT_LETTER_OR_DIGIT)(255) BIT(1);

/* ALPHABET CONSISTS OF THE SYMBOLS CONSIDERED ALPHABETIC IN BUILDING
   IDENTIFIERS     */
DECLARE ALPHABET CHARACTER INITIAL ('ABCDEFGHIJKLMNOPQRSTUVWXYZ_$@#');

/* BUFFER HOLDS THE LATEST CARDIMAGE,
   TEXT HOLDS THE PRESENT STATE OF THE INPUT TEXT
   (NOT INCLUDING THE PORTIONS DELETED BY THE SCANNER),
   TEXT_LIMIT IS A CONVENIENT PLACE TO STORE THE POINTER TO THE END OF TEXT,
   CARD_COUNT IS INCREMENTED BY ONE FOR EVERY SOURCE CARD READ,
   ERROR_COUNT TABULATES THE ERRORS AS THEY ARE DETECTED,
   SEVERE_ERRORS TABULATES THOSE ERRORS OF FATAL SIGNIFICANCE. */
DECLARE (BUFFER, TEXT) CHARACTER,
   (TEXT_LIMIT, CARD_COUNT, ERROR_COUNT, SEVERE_ERRORS, PREVIOUS_ERROR) FIXED
   ;

/* NUMBER_VALUE CONTAINS THE NUMERIC VALUE OF THE LAST CONSTANT SCANNED,
*/
DECLARE NUMBER_VALUE FIXED;

/* EACH OF THE FOLLOWING CONTAINS THE INDEX INTO V() OF THE CORRESPONDING
   SYMBOL.    WE ASK:    IF TOKEN = IDENT    ETC.    */
DECLARE (IDENT, NUMBER, DIVIDE, EOFILE) FIXED;

/* STOPIT() IS A TABLE OF SYMBOLS WHICH ARE ALLOWED TO TERMINATE THE ERROR
   FLUSH PROCESS.  IN GENERAL THEY ARE SYMBOLS OF SUFFICIENT SYNTACTIC
   HIERARCHY THAT WE EXPECT TO AVOID ATTEMPTING TO START CHECKING AGAIN
   RIGHT INTO ANOTHER ERROR PRODUCING SITUATION.  THE TOKEN STACK IS ALSO
   FLUSHED DOWN TO SOMETHING ACCEPTABLE TO A STOPIT() SYMBOL.
   FAILSOFT IS A BIT WHICH ALLOWS THE COMPILER ONE ATTEMPT AT A GENTLE
   RECOVERY.   THEN IT TAKES A STRONG HAND.   WHEN THERE IS REAL TROUBLE
   COMPILING IS SET TO FALSE, THEREBY TERMINATING THE COMPILATION.
*/
DECLARE STOPIT(100) BIT(1), (FAILSOFT, COMPILING) BIT(1);

DECLARE S CHARACTER;  /* A TEMPORARY USED VARIOUS PLACES */
```

```
   /* THE ENTRIES IN PRMASK() ARE USED TO SELECT OUT PORTIONS OF CODED
      PRODUCTIONS AND THE STACK TOP FOR COMPARISON IN THE ANALYSIS ALGORITHM */
   DECLARE PRMASK(5) FIXED INITIAL (0, 0, "FF", "FFFF", "FFFFFF", "FFFFFFFF");

   /*THE PROPER SUBSTRING OF POINTER IS USED TO PLACE AN  |  UNDER THE POINT
      OF DETECTION OF AN ERROR DURING CHECKING.  IT MARKS THE LAST CHARACTER
      SCANNED.  */
   DECLARE POINTER CHARACTER INITIAL ('
                                      |');
   DECLARE CALLCOUNT(20) FIXED   /* COUNT THE CALLS OF IMPORTANT PROCEDURES */
      INITIAL(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);

   /* RECORD THE TIMES OF IMPORTANT POINTS DURING CHECKING */
   DECLARE CLOCK(5) FIXED;

   /* COMMONLY USED STRINGS */
   DECLARE X1 CHARACTER INITIAL(' '), X4 CHARACTER INITIAL('    ');
   DECLARE PERIOD CHARACTER INITIAL ('.');

   /* TEMPORARIES USED THROUGHOUT THE COMPILER */
   DECLARE (I, J, K, L) FIXED;

   DECLARE TRUE LITERALLY '1', FALSE LITERALLY '0', FOREVER LITERALLY 'WHILE 1';

   /*  THE STACKS DECLARED BELOW ARE USED TO DRIVE THE SYNTACTIC
      ANALYSIS ALGORITHM AND STORE INFORMATION RELEVANT TO THE INTERPRETATION
      OF THE TEXT.  THE STACKS ARE ALL POINTED TO BY THE STACK POINTER SP.  */

   DECLARE STACKSIZE LITERALLY '75';  /* SIZE OF STACK  */
   DECLARE PARSE_STACK (STACKSIZE) BIT(8); /* TOKENS OF THE PARTIALLY PARSED
                                              TEXT */
   DECLARE VAR (STACKSIZE) CHARACTER;/* EBCDIC NAME OF ITEM */
   DECLARE FIXV (STACKSIZE) FIXED;   /* FIXED (NUMERIC) VALUE */

   /* SP POINTS TO THE RIGHT END OF THE REDUCIBLE STRING IN THE PARSE STACK,
      MP POINTS TO THE LEFT END, AND
      MPP1 = MP+1.
   */
   DECLARE (SP, MP, MPP1) FIXED;


DECLARE
      ADVANCE LITERALLY 'CALL ADVANCE_PAGE',
      AFLAGCK LITERALLY 'CALL AFLAG_CHECK',
      EMIT LITERALLY 'CALL STORE_CODE',
```

```
      EMITA LITERALLY 'CALL EMIT_ADDRESS',
      EMITCAR LITERALLY 'CALL STORE_ASCII_CODE',
      EMITCK LITERALLY 'CALL EMIT_CHECK',
      CONSIZE LITERALLY '1023';  /* SIZE OF CONSTANT ARRAYS */

DECLARE
  AFLAG BIT(8) INITIAL(0),  /* INDICATES PROCESSING REAL EXPRESSIONS        */
  AT BIT(8) INITIAL(0),     /* POINTER TO TOP OF VCELL_ADDRESS TABLE        */
  BEGIN BIT(8) INITIAL(0),  /* SETS UP JUMP TO MAIN IF SUBPROGRAMS PRECEED */
  BUFFER1 CHARACTER,        /* CONTAINS NEXT CARD IMAGE TO BE SCANNED       */
  BUFFER2 CHARACTER INITIAL(' '), /* TEMPORARILY HOLDS FORTRAN STATEMENT    */
  CA BIT(16) INITIAL(2845), /* POINTS TO NEXT VARIABLE ARRAY SPACE          */
  CB BIT(16) INITIAL(128),  /* NEXT INSTRUCTION CELL TO BE ALLOCATED        */
  CFLAG BIT(8) INITIAL(0),  /* INDICATES CALL TO SUBROUTINE                 */
  CODE(4095) FIXED,         /* CONTAINS CODE FOR PDP-8                      */
  CONSTANT1(CONSIZE) FIXED, /* LOWER 12 BITS=EXPONENT,NEXT 12 OCTAL LOCATION */
  CONSTANT2(CONSIZE) FIXED, /* MANTISSA OR NUMBER < 409                     */
  CT BIT(16) INITIAL(255),  /* TOP LOCATION OF PRESENT PAGE                 */
  DFLAG BIT(8) INITIAL(3),   /* DETERMINES ACTION FOR DECLARATIONS          */
  DIM(70) FIXED,            /* TEMPORARY ARRAY FOR SUBSCRIPTING             */
  DT FIXED,                 /* POINTER WITHIN DIM ARRAY                     */
  EP FIXED INITIAL(128),    /* USED FOR CODE OUTPUT WHILE COMPILING         */
  ENTRY FIXED,              /* CONTAINS PROCEDURE ENTRY & FUNCTION RETURN   */
  FIXM(STACKSIZE) FIXED,    /* HOLDS MANTISSA OF REAL NUMBER                */
  (HOLD1,HOLD2) FIXED,      /* PASSES REAL NUMBERS TO COMPILATION_LOOP      */
  LAB(253) FIXED,           /* CONTAINS LABEL ENTRIES                       */
  LOC(STACKSIZE) FIXED,     /*CODE LOCATIONS OF VARIABLE OR CONSTANT        */
  MAXTCELL BIT(16) INITIAL(62), /* MAX NO. TEMP CELLS IN USE DURING COMPILE */
  NEXT BIT(8) INITIAL(0),   /* NEXT PARAMETER TO BE ASSIGNED THIS ADDRESS   */
  NFLAG BIT(8) INITIAL(0),  /* INDICATES NEXT NUMBER IN SCAN IS A REAL      */
  PAGE_BASE BIT(16) INITIAL(128), /* LOCATION ZERO OF PRESENT PAGE          */
  PAGE_BLOCK BIT(16) INITIAL(128), /* UPDATES LABELS WITHIN THIS BLOCK      */
  PARM BIT(8),              /* IF 1 THEN VARIABLE IS A PARAMETER            */
  PARMCELL BIT(8) INITIAL(127), /* POINTER TO PASS PARAMETER                */
  PC FIXED INITIAL(1),      /*POINTER INTO PTABLE & PSYMBOL                 */
  PCELL BIT(8) INITIAL(16), /* NEXT CELL FOR PROCEDURE INDIRECT ADDRESSING  */
  PRT(383) FIXED,           /* HOLDS ADDRESS,TYPE,ETC. OF IDENTIFIER        */
  PSYMBOL(13) CHARACTER,    /* HOLDS NAMES OF PROCEDURES                    */
  PT FIXED INITIAL(147),    /* ADDRESSES TOP OF PRT                         */
  PTABLE(13) FIXED,         /* HOLDS STATISTICS ON PROCEDURES               */
  RFLAG BIT(8) INITIAL(0),  /* CHECKS FOR RETURN STATEMENT IN SUBPROGRAMS   */
  (SAVE_LABEL_ADDRESS,SAVE_FIRST,STEP,DO_UNTIL) FIXED, /* DO STATE. VARIABLE */
  SC BIT(8) INITIAL(0),     /* NEXT COMMON CELL IN SYMBOL TABLE             */
  SFLAG BIT(8) INITIAL(0),  /* INDICATES SCANNING A SUBPROGRAM              */
  SIZE(255) BIT(16),        /* CONTAINS NUMBER OF CELLS ASSOC. WITH VARIABLE */
  ST FIXED INITIAL(19),     /* ADDRESSES TOP OF SYMBOL TABLE                */
  STOP BIT(8) INITIAL(1),   /* CHECKS FOR STOP STATEMENT IN MAIN PROGRAM    */
```

```
  STRING FIXED,                /* TOKEN INDEX FOR V SET IN INITIALIZE          */
  SYMBOL(255) CHARACTER,       /* HOLDS VARIABLE NAMES                         */
  TCELL BIT(16) INITIAL(62),   /* POINTS TO NEXT TEMPORARY CELL                */
  TYPE FIXED,                  /* VARIABLE TYPE, SET IN LOOKUP & ENTER         */
  VCELL_ADDRESS(127) FIXED;    /* ADDRESSES OF VARIABLES ON OTHER PAGES        */

    /* FLOAT SUBPROGRAM CONVERTS AN INTEGER TO REAL */
DECLARE FLOAT(30) BIT(16) INITIAL(0,3648,3904,2723,3664,242,1574,3776,
      1572,3716,4008,2731,754,3616,3857,550,3880,2738,1060,3592,2733,
      550,3912,2743,3588,2739,3592,3905,1573,1574,2973);

    /* WRITE_STRING ARRAY CONTAINS TWO SUBPROGRAMS
         SUBPROGRAM              NUMBER OF INSTRUCTIONS      OCTAL LOCATIONS
       TAB                              15                      5474-5513
       WRITE_STRING                     12                      5514-5527 */
DECLARE WRITE_STRING(27) BIT(16) INITIAL(0,956,1739,1212,2333,141,0,715,
      4032,3004,2333,160,0,1227,2755,0,0,3776,3110,972,1228,3880,3020,
      3105,2771,3110,3776,2767);

    /* INTEGER_IO ARRAY CONTAINS TWO SUBPROGRAMS
         SUBPROGRAM              NUMBER OF INSTRUCTIONS      OCTAL LOCATIONS
       INTEGER OUTPUT                   27                      5530-5562
       INTEGER INPUT                    13                      5563-5577 */
DECLARE INTEGER_IO(39) BIT(16) INITIAL(0,1573,549,4032,2786,549,3616,
      516,1573,754,3857,1574,1572,549,3592,1060,3872,2790,549,3588,3904,
      2795,3592,3905,1573,3032,2048,0,548,767,4032,2813,549,3656,1573,
      1060,2804,549,3059,4085);


    /* SUBPROGS ARRAY CONTAINS FOUR SUBPROGRAMS WHICH ARE PRELOADED
       ALONG WITH FLOATING POINT PACKAGE(DIGITAL 8-5A-S). PROGRAM
       EXECUTION WILL OVERLAY RIM LOADER LOCATION 7756-7775 OCTAL
         SUBPROGRAM              NUMBER OF INSTRUCTIONS      OCTAL LOCATIONS
        ARRAY SUBSCRIPTOR               42                      7600-7651
        MULTIPLY                        18                      7652-7673
        DIVIDE                          28                      7674-7727
        EXPONENTIATE                    22                      7730-7755 */
DECLARE SUBPROGS(109) BIT(16) INITIAL(0,896,1775,1152,1007,1776,1263,
      563,3904,2703,3616,1774,750,750,750,1774,1264,2707,2721,1264,1007,
      1777,1263,563,3616,516,1778,754,1266,2715,750,1774,2704,647,1687,
      1007,3616,515,751,750,2688,   0,0,3776,938,1774,1006,1774,1194,938,
      1775,1007,3616,1775,1194,1263,2746,2986,750,2743,0,3776,956,1774,
      1006,1774,1212,956,1212,1775,1007,3616,516,3872,2764,3842,1775,
      1776,750,751,1264,3936,2767,3904,2774,514,752,3004,0,3776,984,
      1774,1006,3616,516,1774,1240,984,1775,1240,2311,3055,3311,0,2311,
      1775,0,1262,2792,3032);

    /* ARRAYS A OF 56 THROUGH 74 CONTAIN FLOATING POINT PACKAGE */
```

```
DECLARE
     A56(127) BIT(16) INITIAL(0,3776,1571,1575,896,1707,683,179,4008,
      2700,689,128,1710,690,171,686,1710,692,171,4008,2711,942,
      1710,1152,942,1568,686,1711,1199,943,1569,1199,943,1570,683,3654,
      3590,176,693,1708,940,1708,2988,0,0,0,0,0,15,3968,127,128,256,
      2998,3042,3022,3039,3057,3269,3006,3013,3067,544,1572,545,1573,
      546,1574,2689,548,1966,1198,549,1966,1198,550,1966,2689,2553,
      2689,2554,3648,547,551,1575,3588,546,550,1574,3588,545,549,1573,
      2552,2689,2529,2766,3328,683,176,3880,2944,752,1708,940,1708,
      0640,1709,2476,685,1664,2689,3428,3713,544,548,1572,2551,2689,
      3217,3456,3088,3444,2552,2689,0,0,0),
     A60(127) BIT(16) INITIAL(0,3776,551,3617,1575,550,3616,3864,3649,
      1574,549,3616,3864,3649,1573,2944,0,549,4000,2711,550,4008,2783,
      545,4000,2717,546,4008,2960,544,3617,548,3880,2764,3904,3617,1762,
      738,739,4032,2734,2254,3864,744,743,2773,2254,3856,744,743,1764,
      738,3617,996,2020,1252,740,1765,1253,741,1766,1254,3648,996,
      3912,3600,3592,2020,997,3592,2021,998,3592,2022,1250,2750,1168,
      2960,0,544,3617,548,3588,3712,3022,1764,996,1572,1252,996,1573,
      1252,996,1574,2960,544,1572,2764,0,24,0,0,0,36,4092,2945,754,
      1572,754,1573,3616,1574,1073,3049,2047,0,2311,3114,1578,0,3059,
      0,3065,0,0,0,0,0,0),
     A62(127) BIT(16) INITIAL(0,3776,549,3912,3664,3592,1573,550,3592,
      1574,551,3592,1575,3648,1060,3584,2944,0,3776,755,1767,2270,546,
      2030,550,2541,3712,1007,1575,549,2030,546,2541,551,1575,3588,
      1007,1783,3588,1784,545,2030,550,2541,551,1575,3588,1007,759,
      1783,3588,760,1784,549,2030,545,2541,759,1574,3588,1007,760,1573,
      2544,1575,1269,2961,2555,2961,545,4000,2763,546,4008,3068,544,
      3617,548,3585,1572,754,1767,2270,2545,1007,545,3992,3535,1570,
      1569,1269,2555,3037,3024,0,758,1781,549,4032,2790,2555,1269,
      545,4032,3038,2553,1269,3584,3038,3359,3385,3381,3456,3386,4040,
      4032,2945,0,4094,0,0,3328,3344,3072,3178,0,0,0),
     A64(127) BIT(16) INITIAL(0,3776,547,3617,1571,546,3616,3864,3649,
      1570,545,3616,3864,3649,1569,2944,0,3776,545,3912,3664,3592,1569,
      546,3592,1570,547,3592,1571,3648,2960,0,1718,1717,696,1719,3648,
      694,3592,1718,693,3856,2733,3648,697,3592,1717,1207,2725,694,3592,
      3648,2975,0,0,0,4084,0,0,3776,1577,740,1719,2758,550,3588,1574,
      549,3588,1573,546,550,1721,3588,549,545,3856,2769,1573,697,1574,
      3712,553,3588,1577,551,3588,1575,1207,2752,553,1574,549,3654,1717,
      551,1573,1575,693,3002,4073,3187,3502,3193,3193,3193,3193,3193,
      3193,3193,3193,3193,3193,3193,3193,3193,0,2192,2554,1056,3584,
      3060,3200,0,0,0,0,0),
     A66(127) BIT(16) INITIAL(0,3776,1709,1708,549,3912,1196,4000,
      2703,550,4000,2703,551,4008,2729,684,4000,2475,549,3652,4040,
      2721,551,3652,1575,550,3588,1574,549,3588,1573,1197,2706,685,
      3617,548,1572,684,4000,2475,2944,1572,2944,3072,0,0,0,1778,2311,
      3114,0,549,4040,2793,548,3648,3912,3600,3592,1774,3864,1262,3584,
      753,1775,1776,555,4000,2760,556,4008,2796,2311,2602,2286,750,0,
      3744,548,1572,548,3617,750,4000,2789,549,3617,751,4000,2789,550,
```

```
      3617,752,3904,3617,3585,4040,2789,754,1585,2990,2311,3310,0,2760,
      2475,1266,2736,1572,2990,0,0,0,1549,0,0,0,0,0,0,558,4008,3063,
      767,2558,3063,3812,138),
    A70(127) BIT(16) INITIAL(0,3712,1573,1574,1717,1718,2274,734,3880,
      2703,733,3872,2704,3744,1717,2274,3712,559,735,3904,2944,736,
      3912,2944,1715,549,225,3872,2703,1072,1206,2209,2703,0,550,1713,
      549,1714,1716,2231,2231,2243,2231,691,1713,1714,2243,692,2977,
      0,0,0,0,0,0,0,3776,550,3588,1574,549,3588,1573,692,3588,1716,2999,
      0,3776,550,689,1574,3588,549,690,1573,3588,692,1716,3011,0,3776,
      1205,3024,550,3617,1574,549,2616,3864,3585,1573,3024,4094,3925,3910,
      10,3968,0,3712,3097,2788,3102,1583,559,2550,559,3880,2787,760,
      3880,3063,761,4008,2549,559,3042,3575,3812,3841,3841,114,0,0,0,0,0,0),
    A72(127) BIT(16) INITIAL(0,2191,724,1572,739,2276,2527,557,4008,
      2944,737,2276,738,2276,2944,0,3776,549,4040,3728,727,3864,728,
      2276,2282,729,2276,3776,549,4032,2722,3616,2011,2522,3744,548,1572,
      1748,548,3904,2739,726,4032,2744,2311,2016,0,3744,724,1748,2726,
      2311,1786,0,1236,2726,2012,2526,2525,3848,2287,1060,2748,3880,
      2761,2282,725,1572,2525,2282,1060,2756,2959,3744,724,1748,549,
      4000,2770,550,4008,1748,3744,2754,0,4090,4,171,2,174,3664,3637,3635,
      3617,3639,3922,3908,141,138,197,0,3105,2789,3110,3712,3044,0,
      750,2276,3050,176,0,3656,1764,549,3592,1573,550,3592,1574,740,
      3055,4093,1638,1639,0,0,0),
    A74(127) BIT(16) INITIAL(0,3744,1739,1584,2510,3712,559,714,4000,
      2704,715,4008,2706,2001,1739,3023,715,4008,977,3617,1740,2512,
      713,1572,2311,3584,3114,0,559,712,4000,2730,2510,2512,549,3912,
      3585,4000,2751,550,716,1740,2311,2602,0,716,3880,2944,4032,2744,2311,
      1991,0,1228,2733,2944,2311,1732,0,3744,716,1740,2733,717,1572,
      717,1573,2944,4,1280,0,3834,3899,23,3922,0,0,2047,3584,3599,
      3664,3638,0,3776,548,3912,3633,1572,758,3864,759,2556,1573,548,1061,
      760,3904,2782,761,1572,3616,549,3872,2556,1573,548,1061,762,3904,
      2794,763,1574,3744,549,2556,550,2556,3026,4091,2,3996,100,4086,
      10,3818,0,0,0);


  /*                P R O C E D U R E S                              */


I_FORMAT:
   PROCEDURE (NUMBER, WIDTH) CHARACTER;
      DECLARE (NUMBER, WIDTH, L) FIXED, STRING CHARACTER;
      STRING = NUMBER;
      L = LENGTH(STRING);
      IF L >= WIDTH THEN RETURN STRING;
      ELSE RETURN SUBSTR(X70, 0, WIDTH-L) || STRING;
   END I_FORMAT;
```

```
READ_OCTAL:
   PROCEDURE(N) FIXED;
      /* OCTAL BINARY N CHANGED TO OCTAL NUMBER REPRESENTATION */
      DECLARE (M,N) FIXED;
      M=(SHR(N,9) & "7") * 1000;  M=M+((SHR(N,6) & "7") * 100);
      M=M+((SHR(N,3) & "7") * 10);  M=M+(N & "7");  RETURN M;
   END READ_OCTAL;

LIST_CARD:
   PROCEDURE(S);
      /*  PRINTS THE FORTRAN STATEMENT  */
      DECLARE (C,S,REST) CHARACTER, I FIXED;
      IF BUFFER2=' ' THEN DO; BUFFER2=S;
                              RETURN;  END;
      C=BUFFER2;  BUFFER2=S;
      CARD_COUNT=CARD_COUNT+1;
      IF MARGIN_CHOP > 0 THEN
        DO;  /* THE MARGIN CONTROL FROM DOLLAR | */
           I=LENGTH(C) - MARGIN_CHOP;  REST=SUBSTR(C,I);
           C=SUBSTR(C,0,I);  END;
        ELSE REST=' ';
      IF CONTROL(BYTE('M')) THEN OUTPUT=C;
         ELSE IF CONTROL(BYTE('L')) THEN
            OUTPUT=I_FORMAT(CARD_COUNT,4) || '|' || C || '|' || REST;
      IF CONTROL(BYTE('C')) THEN DO I=EP TO CB-1;
         EP=CODE(I);  C='                '||READ_OCTAL(I)||'    ';
         IF EP<4096 THEN OUTPUT=C||READ_OCTAL(EP);
          ELSE IF EP=4096 THEN OUTPUT=C||'----FEXT';
          ELSE IF EP=4097 THEN OUTPUT=C||'----WEXT';
          ELSE OUTPUT=C||'----LABEL= '||LAB(SHR(EP,16));  END;
      CP=0;  EP=CB;
   END LIST_CARD;

ERROR:
   PROCEDURE(MSG, SEVERITY);
      /* PRINTS AND ACCOUNTS FOR ALL ERROR MESSAGES */
      /* IF SEVERITY IS NOT SUPPLIED, 0 IS ASSUMED */
      DECLARE MSG CHARACTER, SEVERITY FIXED;
      CALL LIST_CARD(' '); /* FORCE THE LISTING OF THE CARD IN BUFFER2 */
      ERROR_COUNT = ERROR_COUNT + 1;
      /* IF LISTING IS SUPPRESSED, FORCE PRINTING OF THIS LINE */
      IF ¬ CONTROL(BYTE('L')) THEN
         OUTPUT = I_FORMAT (CARD_COUNT, 4) || ' |' || BUFFER || '|';
      OUTPUT = SUBSTR(POINTER, TEXT_LIMIT-CP+MARGIN_CHOP);
      OUTPUT = '*** ERROR, ' || MSG ||
            '.  LAST PREVIOUS ERROR WAS DETECTED ON LINE ' ||
            PREVIOUS_ERROR || '.  ***';
```

```
      PREVIOUS_ERROR = CARD_COUNT;
      IF SEVERITY > 0 THEN
         IF SEVERE_ERRORS > 25 THEN
            DO;
               OUTPUT = '*** TOO MANY SEVERE ERRORS, CHECKING ABORTED ***';
               COMPILING = FALSE;  END;
         ELSE SEVERE_ERRORS = SEVERE_ERRORS + 1;
   END ERROR;



  /*                    CARD IMAGE HANDLING PROCEDURE                    */

GET_CARD:
   PROCEDURE;
      DECLARE (FLAG,I) FIXED, C CHARACTER;
        /* FLAG HOLDS COUNT OF CONTINUATION CARDS - - I DETERMINES
           WHERE TO INSERT ; FOR END OF STATEMENT  */
      I=72;   FLAG=0;   BUFFER=BUFFER1;
      DO FOREVER;
        C=SUBSTR(BUFFER,0,1);
        IF C=' ' THEN
           DO FOREVER;
              BUFFER1=INPUT;
              IF (SUBSTR(BUFFER1,0,1)='C') | (SUBSTR(BUFFER1,5,1)=' ')
                 THEN DO; IF FLAG=0 THEN CALL LIST_CARD(BUFFER);
                          BUFFER=SUBSTR(BUFFER,0,I) || ';'  ;
                          TEXT=BUFFER;   TEXT_LIMIT=LENGTH(TEXT) -1;
                          RETURN;
                      END;   /* ELSE A CONTINUATION  */
              IF FLAG=0 THEN CALL LIST_CARD(BUFFER);
                        ELSE IF FLAG > 2 THEN
                             DO;CALL ERROR('ONLY 2 CONTINUATIONS ALLOWED',2);
                                RETURN;  END;
              CALL LIST_CARD(BUFFER1);
              BUFFER=SUBSTR(BUFFER,0,I) || SUBSTR(BUFFER1,6,66);
              FLAG=FLAG+1;   I=I+66;   END;
        IF C='C' THEN CALL LIST_CARD(BUFFER);
          ELSE IF SUBSTR(BUFFER,0,3)='$GO' THEN
              DO; CALL LIST_CARD(BUFFER);  TEXT='EOF;  ';
                  TEXT_LIMIT=LENGTH(TEXT)-1;  BUFFER1='$EOF;  ';  RETURN;
                END;
          ELSE IF SUBSTR(BUFFER,0,4)='$EOF' THEN
              DO; TEXT='EOF;  ';  TEXT_LIMIT=LENGTH(TEXT)-1;  RETURN;  END;
          ELSE DO; I=BYTE(BUFFER,1);  CONTROL(I)= ¬CONTROL(I);
```

```
                    CALL LIST_CARD(BUFFER);  I=72;  END;
         BUFFER,BUFFER1=INPUT;  END;
   END GET_CARD;

   /*                  THE SCANNER PROCEDURES                  */

COMPUTE:
   PROCEDURE (M,N,P);
      /* CONVERTS DECIMAL FRACTIONS TO OCTAL EXPONENTIAL (REAL)
         RETURNS EXPONENT IN HOLD1 AND MANTISSA IN HOLD2 */
      DECLARE (I,J,K,L,M,N,P) FIXED, C CHARACTER;
      J=0;
      DO I=1 TO 8;
        C=N * 8;
        IF LENGTH(C)>M THEN DO; K=BYTE(C) - "F0";  J=SHL(J,3) | K;
                                DO L=2 TO LENGTH(C);  K=K*10;  END;
                                N=N * 8 - K;  END;
                       ELSE DO; N=N * 8;  J=SHL(J,3);  END;  END;
      N=SHL(J,8);  J=1;
      DO I=0 TO 30;
        IF P<J THEN GO TO FINISH;  J=SHL(J,1);  END;
      FINISH: HOLD1=I;  HOLD2=SHR( (SHR(N,I) | SHL(P,32-I) ),9);
      IF NUMBER_VALUE > 0 THEN RETURN;
      /* HAVE TO RESET FOR NUMBER < 0.5 */
      IF SHR(HOLD2,22)=1 THEN RETURN;
      HOLD1=4095;  HOLD2=SHL(HOLD2,1);
      DO I=1 TO 24;
         IF SHR(HOLD2,22)=1 THEN RETURN;
         HOLD1=HOLD1-1;  HOLD2=SHL(HOLD2,1);  END;
      CALL ERROR('UNABLE TO CONVERT NUMBER',2);
         OUTPUT='NUMBER= '||NUMBER_VALUE||'.'||SUBSTR(TEXT,CP-M,M);
   END COMPUTE;

SCAN:
   PROCEDURE;
      DECLARE(S1,S2,S3) FIXED;
      CALLCOUNT(3) = CALLCOUNT(3) + 1;
      FAILSOFT = TRUE;
      BCD = '';  NUMBER_VALUE = 0;
   SCAN1:
      DO FOREVER;
         IF CP > TEXT_LIMIT THEN CALL GET_CARD;
         ELSE
            DO; /* DISCARD LAST SCANNED VALUE */
               TEXT_LIMIT = TEXT_LIMIT - CP;
               TEXT = SUBSTR(TEXT, CP);
```

```
      CP = 0;  END;
/*  BRANCH ON NEXT CHARACTER IN TEXT                          */
DO CASE CHARTYPE(BYTE(TEXT));

   /*  CASE 0  */
   /* ILLEGAL CHARACTERS FALL HERE  */
   CALL ERROR ('ILLEGAL CHARACTER: ' || SUBSTR(TEXT, 0, 1));

   /*  CASE 1  */
   /*  BLANK  */
   DO;
      CP = 1;
      DO WHILE BYTE(TEXT, CP) = BYTE(' ') & CP <= TEXT_LIMIT;
         CP = CP + 1;  END;
      CP = CP - 1;  END;

   /*  CASE 2  */
;  /*  NOT USED IN SKELETON (BUT USED IN XCOM)  */

   /*  CASE 3  */
   DO; /* LOCATE STRING AND PLACE IN BCD */
       TOKEN=STRING;
       DO CP=CP+1 TO TEXT_LIMIT;
         S1=BYTE(TEXT,CP);
         IF S1=125 THEN DO;  CP=CP+1;  RETURN;  END;
         BCD=BCD || SUBSTR(TEXT,CP,1);  END;  END;

   /*  CASE 4  */
   DO FOREVER;  /* A LETTER:  IDENTIFIERS AND RESERVED WORDS */
      DO CP = CP + 1 TO TEXT_LIMIT;
         IF NOT_LETTER_OR_DIGIT(BYTE(TEXT, CP)) THEN
            DO;  /* END OF IDENTIFIER  */
               IF CP > 0 THEN BCD = BCD || SUBSTR(TEXT, 0, CP);
RES_WORD:      S1=LENGTH(BCD);
               IF S1 > 1 THEN IF S1 <= RESERVED_LIMIT THEN
                  /* CHECK FOR RESERVED WORDS */
                  DO I = V_INDEX(S1-1) TO V_INDEX(S1) - 1;
                     IF BCD = V(I) THEN
                        DO;
                           TOKEN = I;
                           RETURN;  END;  END;
               /*  RESERVED WORDS EXIT HIGHER: THEREFORE <IDENTIFIER>*/
               TOKEN = IDENT;
               RETURN;  END;  END;
      /*  END OF CARD  */
      BCD = BCD || TEXT;
      CALL GET_CARD;
      CP = -1;  END;
```

```
      /*  CASE 5  */
      DO;       /*  DIGIT:  A NUMBER  */
        TOKEN=NUMBER;  NFLAG=0;
        DO FOREVER;
          DO CP=CP TO TEXT_LIMIT;
            S1=BYTE(TEXT,CP);
            IF S1=BYTE('.') THEN
             DO; CP=CP+1;  S2,S3=0;  NFLAG=1;
               DO FOREVER;
                 DO CP=CP TO TEXT_LIMIT;
                   S1=BYTE(TEXT,CP);
                   IF S1 < "F0" THEN
                     DO; IF NUMBER_VALUE + S3 = 0
                           THEN HOLD1,HOLD2=0; /* VALUE IS 0.0 */
                           ELSE CALL COMPUTE(S2,S3,NUMBER_VALUE);
                         RETURN;  END;
                   S2=S2+1;  S3=10 * S3 + S1 - "F0";  END;
                 CALL GET_CARD;  END;  END;
            IF S1 < "F0" THEN RETURN;
            NUMBER_VALUE=10*NUMBER_VALUE + S1 - "F0";  END;
          CALL GET_CARD;  END;  END;

      /*  CASE 6  */
    ; /*  CASE 6 NOT UTILIZED  */

      /*  CASE 7  */
      DO;       /*  SPECIAL CHARACTERS  */
        TOKEN = TX(BYTE(TEXT));
        CP = 1;
        RETURN;  END;

      /*  CASE 8  */
    ; /*  NOT USED IN SKELETON (BUT USED IN XCOM)  */

      /*   CASE 9   */
      /*  DETERMINATION OF PERIOD  */
      DO; BCD='.';  CP=CP+1;
        DO FOREVER;
          IF BYTE(TEXT,CP) = BYTE('.') THEN
            DO;  BCD= BCD || '.';  CP=CP+1;
              GO TO RES_WORD;  END;
          ELSE IF BYTE(TEXT,CP) ¬= BYTE(' ') THEN
            BCD=BCD ||SUBSTR(TEXT,CP,1);
          CP=CP+1;  END;  END;
    END;     /* OF CASE ON CHARTYPE  */
    CP = CP + 1;  /* ADVANCE SCANNER AND RESUME SEARCH FOR TOKEN  */
END;
```

```
   END SCAN;


  /*                          TIME AND DATE                                */

PRINT_TIME:
   PROCEDURE (MESSAGE, T);
      DECLARE MESSAGE CHARACTER, T FIXED;
      MESSAGE = MESSAGE || T/360000 || ':' || T MOD 360000 / 6000 || ':'
         || T MOD 6000 / 100 || '.';
      T = T MOD 100;  /* DECIMAL FRACTION  */
      IF T < 10 THEN MESSAGE = MESSAGE || '0';
      OUTPUT = MESSAGE || T || '.';
   END PRINT_TIME;

PRINT_DATE_AND_TIME:
   PROCEDURE (MESSAGE, D, T);
      DECLARE MESSAGE CHARACTER, (D, T, YEAR, DAY, M) FIXED;
      DECLARE MONTH(11) CHARACTER INITIAL ('JANUARY', 'FEBRUARY', 'MARCH',
         'APRIL', 'MAY', 'JUNE', 'JULY', 'AUGUST', 'SEPTEMBER', 'OCTOBER',
         'NOVEMBER', 'DECEMBER'),
      DAYS(12) FIXED INITIAL (0, 31, 60, 91, 121, 152, 182, 213, 244, 274,
         305, 335, 366);
      YEAR = D/1000 + 1900;
      DAY = D MOD 1000;
      IF (YEAR & "3") ¬= 0 THEN IF DAY > 59 THEN DAY = DAY + 1; /* ¬ LEAP YEAR*/
      M = 1;
      DO WHILE DAY > DAYS(M);  M = M + 1;  END;
      CALL PRINT_TIME(MESSAGE || MONTH(M-1) || X1 || DAY-DAYS(M-1) ||  ', '
         || YEAR || '.  CLOCK TIME = ', T);
   END PRINT_DATE_AND_TIME;

  /*                          INITIALIZATION                               */


INITIALIZATION:
   PROCEDURE;
      EJECT_PAGE;
   CALL PRINT_DATE_AND_TIME ('   SYNTAX CHECK -- STANFORD UNIVERSITY -- SKELETON
 III VERSION OF ', DATE_OF_GENERATION, TIME_OF_GENERATION);
      DOUBLE_SPACE;
      CALL PRINT_DATE_AND_TIME ('TODAY IS ', DATE, TIME);
      DOUBLE_SPACE;
      DO I = 1 TO NT;
```

```
      S = V(I);
      IF S = '<NUMBER>' THEN NUMBER = I;  ELSE
      IF S= '<STRING>' THEN STRING=I;  ELSE
      IF S = '<IDENTIFIER>' THEN IDENT = I;  ELSE
      IF S = '/' THEN DIVIDE = I;  ELSE
      IF S = '_|_' THEN EOFILE = I;  ELSE
      IF S = ';' THEN STOPIT(I) = TRUE;  ELSE
      ;
   END;
   IF IDENT = NT THEN RESERVED_LIMIT = LENGTH(V(NT-1));
   ELSE RESERVED_LIMIT = LENGTH(V(NT));
   V(EOFILE) = 'EOF';
   STOPIT(EOFILE) = TRUE;
   CHARTYPE(BYTE(' ')) = 1;
   DO I = 0 TO 255;
      NOT_LETTER_OR_DIGIT(I) = TRUE;
   END;
   DO I = 0 TO LENGTH(ALPHABET) - 1;
      J = BYTE(ALPHABET, I);
      TX(J) = I;
      NOT_LETTER_OR_DIGIT(J) = FALSE;
      CHARTYPE(J) = 4;
   END;
   DO I = 0 TO 9;
      J = BYTE('0123456789', I);
      NOT_LETTER_OR_DIGIT(J) = FALSE;
      CHARTYPE(J) = 5;
   END;
   DO I = V_INDEX(0) TO V_INDEX(1) - 1;
      J = BYTE(V(I));
      TX(J) = I;
      CHARTYPE(J) = 7;
   END;
   CHARTYPE(BYTE('/')) = 6;
   /* FIRST SET UP GLOBAL VARIABLES CONTROLLING SCAN, THEN CALL IT */
   CP = 0;  TEXT_LIMIT = -1;
   TEXT = '';
   CONTROL(BYTE('C'))=FALSE; /* OUTPUT CODE WHILE COMPILING */
   CONTROL(BYTE('L')) = TRUE;
   CONTROL(BYTE('P'))=FALSE;  /* OUTPUT PRODUCTIONS */
   CONTROL(BYTE('R'))=FALSE; /* OUTPUT COMPILED CODE */
   CONTROL(BYTE('S'))=FALSE; /* OUTPUT FOR SYMBOL TABLE          */
   CONTROL(BYTE('T'))=FALSE;  /* OUTPUT PROCEDURE TABLE */
   CHARTYPE(BYTE('.'))=9;
   CHARTYPE(BYTE('/'))=7;
   CHARTYPE(125)=3;  /* SINGLE QUOTE */
   BUFFER1=INPUT;
   DO I=0 TO 126;  PRT(I),LAB(I)=0;  LAB(I+127),VCELL_ADDRESS(I)=0;  END;
```

```
      DO I=0 TO CONSIZE; CONSTANT1(I),CONSTANT2(I)=0; END;
      DO I= 0 TO 2876;  CODE(I)=0;  END;
      CODE(2)=4095;  CODE(3)=2;  CODE(4)=1;
      CONSTANT1(1)=SHL(4,12);  CONSTANT2(1)=1;
      CONSTANT1(2)=SHL(3,12);  CONSTANT2(2)=2;
      I=4095 MOD CONSIZE;  CONSTANT1(I)=SHL(2,12); CONSTANT2(I)=4095;
      CODE(5)=3712;   /* 7200 OCTAL */
      CODE(6)=3840;   /* 7400 OCTAL */
      CODE(7)=2944;   /* 5600 OCTAL */
      CODE(27)=2845;
      CODE(28)=2876; /* TAB ROUTINE */
      CODE(29)=2892; /* WRITE STRING ROUTINE */
      CODE(30)=2904; /* INTEGER WRITE ROUTINE */
      CODE(31)=2931; /* INTEGER READ ROUTINE */
      CODE(45),CODE(46)=4095; /* FLOATING POINT I/O INITIALIZATION */
      CODE(50)=3968; /* 7600 OCTAL ARRAY-SUBSCRIPTOR SUBPROGRAM */
      CODE(54)=4010; /* 7652 OCTAL-MULTIPLIER */
      CODE(55)=4028; /* 7674 OCTAL-DIVIDER */
      CODE(56)=1; CODE(57)=3072; /* NEG 1 IN FLOATING POINT */
      CODE(61)=4056; /* 7730 OCTAL-EXPONENTIATOR */
      DO I=0 TO 30;  CODE(I+2845)=FLOAT(I);  END;
      DO I=0 TO 27;  CODE(I+2876)=WRITE_STRING(I);  END;
      DO I=0 TO 39;  CODE(I+2904)=INTEGER_IO(I);  END;
      DO I=0 TO 127;   /* LOAD FLOATING POINT PACKAGE */
        CODE(I+2944)=A56(I);  CODE(I+3072)=A60(I);
        CODE(I+3200)=A62(I);  CODE(I+3328)=A64(I);
        CODE(I+3456)=A66(I);  CODE(I+3584)=A70(I);
        CODE(I+3712)=A72(I);  CODE(I+3840)=A74(I);  END;
      DO I=0 TO 109;  CODE(I+3968)=SUBPROGS(I);  END;
        /* SET INITIAL JUMP TO MAIN PROGRAM */
      CODE(128)=2945; /* JMP I 201 */  CODE(129)=130;  CB=CB+2;
    . CALL SCAN;

      /* INITIALIZE THE PARSE STACK */
      SP = 1;  PARSE_STACK(SP) = EOFILE;

  END INITIALIZATION;


DUMPIT:
  PROCEDURE;     /* DUMP OUT THE STATISTICS COLLECTED DURING THIS RUN  */
    DOUBLE_SPACE;
    /*  PUT OUT THE ENTRY COUNT FOR IMPORTANT PROCEDURES */
    OUTPUT = 'STACKING DECISIONS= ' || CALLCOUNT(1);
    OUTPUT = 'SCAN               = ' || CALLCOUNT(3);
    OUTPUT = 'FREE STRING AREA   = ' || FREELIMIT - FREEBASE;
  END DUMPIT;
```

```
STACK_DUMP:
   PROCEDURE;
      DECLARE LINE CHARACTER;
      LINE = 'PARTIAL PARSE TO THIS POINT IS: ';
      DO I = 2 TO SP;
         IF LENGTH(LINE) > 105 THEN
            DO; OUTPUT=LINE;
                LINE=X4;  END;
         LINE = LINE || X1 || V(PARSE_STACK(I));  END;
      OUTPUT = LINE;
   END STACK_DUMP;

  /*                          PROCEDURES FOR SYNTHESIZE                          */

ADVANCE_PAGE:
   PROCEDURE(N);
        /* SHIFT CODE GENERATION TO NEXT PAGE IF INSUFFICIENT ROOM
           AVAILABLE ON PRESENT ONE */
      DECLARE(N) FIXED;
      IF (CT-CB-N-AFLAG) < 1 THEN DO;
         IF AFLAG THEN DO; CODE(CB)= 0; /* FEXT */
                           CB=CB+1;  END;
         CODE(CB)=2944+CB+1-PAGE_BASE; /* 2944=56XX OCTAL, A JUMP */
         PAGE_BASE=PAGE_BASE+128;  CODE(CB+1)=PAGE_BASE;  CB=PAGE_BASE;
         CT=PAGE_BASE+127;
         DO I=0 TO AT;  VCELL_ADDRESS(I)=0;  END;  AT=0;
         IF AFLAG THEN DO; CODE(CB)=2311; /* JMS I 7 */ CB=CB+1;  END;  END;
   END ADVANCE_PAGE;

GET_ACELL:
   PROCEDURE(N) FIXED;
        /* RETURNS NEXT ARRAY CELL BLOCK, STARTING TOP PAGE 27 */
      DECLARE N FIXED;
      CA=CA-N;    RETURN CA;
   END GET_ACELL;

GET_PCELL:
   PROCEDURE FIXED;
        /* SETS CORE LOCATION FOR PROCEDURE M FROM 20 TO 36 OCTAL */
      IF PCELL=27 THEN DO; CALL ERROR('TOO MANY SUBPROGRAMS',2);
                           RETURN;      END;
      PCELL=PCELL+1;  RETURN PCELL-1;
   END GET_PCELL;

GET_TCELL:
   PROCEDURE (N,T) FIXED;
        /* RETURNS TEMPORARY STORAGE FROM PAGE ZERO DEPENDING ON
```

```
            WHETHER THAT LOCATION CONTAINS AN ADDRESS OR HOLDS A
            REAL OR INTEGER VALUE */
         DECLARE (N,T) FIXED;
         LOC(N)=T|SHL(TCELL,4);
         IF TCELL > PARMCELL THEN CALL ERROR
            ('PARAMETER STORAGE OVERLAYED,SEPARATE INTO ADDITIONAL STATEMENTS',2);
         IF T=1 THEN DO; TCELL=TCELL+3;
                         IF TCELL>MAXTCELL THEN MAXTCELL=TCELL;
                         RETURN TCELL-3;   END;
         IF TCELL+1 > MAXTCELL THEN MAXTCELL=TCELL+1;
         TCELL=TCELL+1;  RETURN TCELL-1;
      END GET_TCELL;

   GET_VCELL:
      PROCEDURE(N) FIXED;
           /* RETURNS NEXT N CELLS FROM TOP OF PRESENT PAGE IF POSSIBLE */
         DECLARE N FIXED;
         ADVANCE(N+1);
         CT=CT-N;  RETURN CT+1;
      END GET_VCELL;


   STORE_CODE:
      PROCEDURE (N) FIXED;
           /* STORE N IN NEXT AVAILABLE INSTRUCTION LOCATION OF CODE ARRAY
              RETURNS CODE LOCATION */
         DECLARE N FIXED;
         IF AFLAG THEN ADVANCE(2);
                  ELSE ADVANCE(1);
         CODE(CB)=N;  CB=CB+1;  RETURN CB-1;
      END STORE_CODE;

   AFLAG_CHECK:
      PROCEDURE;
           /* SETS AFLAG AND EMITS JMS TO INTREPTER IF AFLAG=0 */
         IF AFLAG=1 THEN RETURN;  ADVANCE(2);  AFLAG=1;
         IF CODE(CB-1)=4096 THEN CB=CB-1; /* REMAIN IN INTREPTER */
                            ELSE EMIT(2311); /* JMS I 7 */
      END AFLAG_CHECK;

   EMIT_CHECK:
      PROCEDURE(OP,L);
           /* CHECK IF INDIRECT ADDRESSING REQUIRED: OP=OPERATION,LOC(L) */
         DECLARE(OP,L,P,M) FIXED;
         P=SHR(LOC(L),4);
         IF P<128 THEN DO;
           IF (LOC(L)&"4")>0 THEN EMIT(P|SHL(OP,9)|SHL(1,8));/* INDIRECT ADDRESS */
                             ELSE EMIT(P|SHL(OP,9)); /* DIRECT ADDRESS */
```

```
            RETURN;    END;
      IF(P >= PAGE_BASE) & (P < PAGE_BASE+128)
         THEN DO; M=(P-PAGE_BASE)|SHL(1,7); /* VARIABLE ON PRESENT PAGE */
                  EMIT(M|SHL(OP,9));  RETURN;  END;
      /* CHECK IF ADDRESS OF VARIABLE ON CURRENT PAGE */
      DO I=0 TO AT;
         IF (VCELL_ADDRESS(I)&"FFFF")=P THEN DO; P=SHR(VCELL_ADDRESS(I),16);
                                                 M=(P-PAGE_BASE)|SHL(3,7);
                                                 EMIT(M|SHL(OP,9));  RETURN;  END;  END;
      /* ENTER REFERENCE TO VARIABLE ON CURRENT PAGE AND INDIRECT ADDRESS */
      M=GET_VCELL(I);  CODE(M)=P;  VCELL_ADDRESS(AT)=SHL(M,16)|P;
      AT=AT+1;  M=M|SHL(3,7);  EMIT(M|SHL(OP,9));
   END EMIT_CHECK;

STORE_ASCII_CODE:
   PROCEDURE(N);
      /* CHECKS IF WRITE_STRING SUBPROGRAM STILL ENGAGED AND EMITS
         ASCII CHARACTER FOR TELETYPE OUTPUT */
      DECLARE N FIXED;
      IF CODE(CB-1)=4097 THEN DO; CB=CB-1;
                              ADVANCE(2);  END;
                 ELSE DO; ADVANCE(3);
                              EMIT(2333); /* JMS I 35 */  END;
      EMIT(N);
      EMIT(4097); /* WEXT */
   END STORE_ASCII_CODE;

EMIT_ADDRESS:
   PROCEDURE(L,BACK);
      /* IF L CONTAINS AN ADDRESS LOAD THE ADDRESSED SPACE
         IN THE NEXT POSITION ELSE LOAD THE ADDRESS */
    . DECLARE(L,BACK,M) FIXED;
      M=SHR(LOC(L),4);
      IF (LOC(L)&"4")=0 THEN DO;EMIT(M);
                             RETURN; END;
      DO I=0 TO BACK-1;  CODE(CB-I+1)=CODE(CB-I-1);  END;
      CODE(CB-BACK)=512+M;  CODE(CB-BACK+1)=1664+CB-PAGE_BASE+2;
      IF BACK=2 THEN  IF CODE(CB-3)=(1664+CB-PAGE_BASE-1) THEN
           CODE(CB-3)=CODE(CB-3)+2;
      CB=CB+3;
   END EMIT_ADDRESS;

EMIT_STRING:
   PROCEDURE(C);
      /* OUTPUT STRING C */
      DECLARE C CHARACTER, (I,L,N) FIXED;
      DO I=0 TO LENGTH(C)-1;   N=BYTE(C);
         IF N=64 THEN EMITCAR(160); /* SPACE */
```

```
            ELSE IF N>192 & N<202 THEN EMITCAR(N); /* LETTER A - I */
            ELSE IF N>208 & N<218 THEN EMITCAR(N-7); /* LETTER J - R */
            ELSE IF N>225 & N<234 THEN EMITCAR(N-15); /* LETTER S - Z */
            ELSE IF N>239 & N<250 THEN EMITCAR(N-64); /* DECIMAL DIGIT */
            ELSE DO;
                    /* FORTRAN STRING CHARACTER | ::= "   */
                 DO J=0 TO 14; /* CHECK FOR    " # $ % & ' ( ) * + , - . /   */
                   IF SUBSTR(C,0,1)=SUBSTR('¬|#$%&"()*+,-./' ,J,1)
                      THEN DO; EMITCAR(161+J);  GO TO FOUND;  END;  END;
                 DO J=0 TO 6; /* CHECK FOR : ; < = > ? @   */
                   IF SUBSTR(C,0,1)=SUBSTR(':;<=>?@' ,J,1)
                      THEN DO; EMITCAR(186+J);  GO TO FOUND;  END;  END;
            CALL ERROR('ILLEGAL STRING SYMBOL '||SUBSTR(C,0,1),2);
            FOUND:  END;  C=SUBSTR(C,1);  END;
   END EMIT_STRING;

STORE_CONSTANT:
   PROCEDURE(N1,N2,REQ) FIXED;
        /* STORE CONSTANT IN REQ NUMBER OF CELLS-RETURNS CODE LOCATION  */
      DECLARE(L,N1,N2,M,REQ) FIXED;
      IF (N1+N2)=0 THEN RETURN 58; /* ZERO CONSTANT CORE LOCATION */
      L=(N1+N2) MOD CONSIZE;
      DO FOREVER;
         IF CONSTANT2(L)=N2
            THEN IF (CONSTANT1(L) & "FFF")=N1
               THEN RETURN SHR(CONSTANT1(L),12);
         IF CONSTANT2(L) + CONSTANT1(L)=0 THEN GO TO ENTER;
         IF L=CONSIZE THEN L=0;
                      ELSE L=L+1;  END;
    ENTER:  /* CONSTANT NOT LOCATED - INSERT CONSTANT */
      M=GET_VCELL(REQ);  CONSTANT2(L)=N2;
      CONSTANT1(L )=N1 | SHL(M,12);
      IF REQ>1 THEN DO; CODE(M)=N1 & "FFF";  CODE(M+1)=SHR(N2,12);
                        CODE(M+2)=N2 & "FFF";  END;
               ELSE CODE(M)=N2 & "FFF";
      RETURN M;
   END STORE_CONSTANT;


SET:
   PROCEDURE(C) FIXED;
        /* RETURN BASED ON I THRU N BEING INTEGERS
           INTEGER=0   , REAL=1     */
      DECLARE C CHARACTER, K FIXED;
      K=BYTE(C);
      IF (K > 200) & (K < 214) THEN RETURN 0; /* AN INTEGER */
                               ELSE RETURN 1; /* A REAL */
   END SET;
```

```
HASH:
   PROCEDURE(C,L) FIXED;
        /* L= NUMBER OF CHARACTERS IN C
                RETURNS ENTRY INTO HASH SECTION OF PRT  */
      DECLARE C CHARACTER, (I,K,L) FIXED;
      K=L;   I=0;
      DO WHILE (I <= 2) & (I < L);  K=SHL(K,8)|BYTE(C,I);  I=I+1;  END;
      RETURN K MOD 127;
   END HASH;

LOOKUP:
   PROCEDURE (C) FIXED;
        /* FIND IDENTIFIER C AND RETURN PRT LOCATION */
      DECLARE C CHARACTER, (N,L) FIXED;
      L=HASH(C,LENGTH(C));
      IF PRT(L)=0 THEN RETURN 0;  L=PRT(L);
      DO FOREVER;
         IF SYMBOL(PRT(L) & "FF")=C THEN DO; TYPE=SHR(PRT(L),30);
                                             PARM=SHR(SHL(PRT(L),2),31);
                                             RETURN L;  END;
         L=SHR(SHL(PRT(L),3),23);
         IF PRT(L)=0 THEN RETURN 0;  END;
   END LOOKUP;

ENTER:
   PROCEDURE (C,T) FIXED;
        /* C IS IDENTIFIER OF TYPE T TO ENTER IN PRT, RETURNS PRT LOCATION */
      DECLARE C CHARACTER, (L,M,N,T) FIXED;
      N=LENGTH(C);
      IF N > 6 THEN
         DO; OUTPUT='* * * WARNING * * *';
             CALL ERROR('IDENTIFIER LENGTH EXCEEDS 6 CHARACTERS',0);
             C=SUBSTR(C,0,6);  END;
      IF ST>255 THEN CALL ERROR('SYMBOL TABLE LIMIT EXCEEDED',2);
      L=HASH(C,N);
      IF PRT(L)=0 THEN DO; ST=ST+1;  PRT(L)=PT;  PRT(PT)=ST | SHL(T,30);
                           SYMBOL(ST)=C;  PT=PT+1;  TYPE=T;
                           IF TYPE=0 THEN SIZE(ST)=1;
                                     ELSE SIZE(ST)=3;
                           RETURN PT-1;  END;
      L=PRT(L); /* A COLLISION OCCURRED */
      DO FOREVER;
         M=SHR(SHL(PRT(L),3),23);
         IF M=0 THEN DO; PRT(L)=PRT(L) | SHL(PT,20);  ST=ST+1;
                         PRT(PT)=ST | SHL(T,30);  SYMBOL(ST)=C;
                         PT=PT+1;  TYPE=T;
                         IF TYPE=0 THEN SIZE(ST)=1;
```

```
                                        ELSE SIZE(ST)=3;
                                RETURN PT-1;  END;
          L=M;  END;
    END ENTER;

ENTER1:
    PROCEDURE(C,N) FIXED;
          /* ENTERS VARIABLE IN PRT AND GETS CODE STORAGE FOR THAT VARIABLE
            RETURNS CODE LOCATION */
       DECLARE C CHARACTER, (L,M,N) FIXED;
       M=ENTER(C,SET(C));
       IF SHR(PRT(M),30)=1 THEN DO; L=GET_VCELL(3); LOC(N)=SHL(L,4)|1; END;
                           ELSE DO; L=GET_VCELL(1); LOC(N)=SHL(L,4)|0; END;
       PRT(M)=PRT(M) | SHL(L,8);
       RETURN M;
    END ENTER1;

COMMON_CHECK:
    PROCEDURE(L);
         /* RELOCATES VARIABLES INTO COMMON SECTION OF PRT
            L REFERS TO STACKSIZE POINTER (MP TO SP) */
       DECLARE(L,M,P) FIXED;
       P=LOOKUP(VAR(L));
       IF P=0 THEN
          DO; CALL ERROR('COMMON VARIABLE: '||VAR(L)||' ,MUST BE KNOWN',2);
             OUTPUT='**PLACE VARIABLE IN A DECLARATION STATEMENT';  END;
       IF P<147 THEN RETURN;  /* ALREADY RELOCATED */
       IF SC>19 THEN CALL ERROR('EXCESSIVE COMMON VARIABLES',2);    M=-1;
       DO I=0 TO 126;  /* LOCATE HASH ENTRY */
          IF PRT(I)=P THEN M=I;  END;
       IF M=-1 THEN DO;
          CALL ERROR('COMMON VARIABLE: '||VAR(L)||' ,CANNOT BE RELOCATED',2);
          M=HASH(VAR(L),LENGTH(VAR(L)));
          OUTPUT='***RENAME VARIABLE: '||PSYMBOL(PRT(PRT(M))&"FF");  END;
       PRT(M)=SC+127;  SYMBOL(SC)=VAR(L);
       SYMBOL(PRT(P) & "FF")='ZZZZZZZ'; /* RETAIN FOR COLLISION */
       PRT(SC+127)=(PRT(P) & "E00FFF00") +SC;  SIZE(SC)=SIZE(P-127);  SC=SC+1;
       PRT(P)=PRT(P) &"1FF000FF";
    END COMMON_CHECK;


FIND_PROC:
    PROCEDURE(C) FIXED;
          /* LOCATE IDENTIFIER C IN PROCEDURE TABLE - RETURNS LOCATION */
       DECLARE C CHARACTER, I FIXED;
       DO I=1 TO PC;  IF PSYMBOL(I)=C THEN RETURN I;  END;
       RETURN 0;
    END FIND_PROC;
```

```
SET_PROC:
   PROCEDURE(C,T) FIXED;
        /* PLACE IDENTIFIER C OF TYPE T IN PROCEDURE TABLES-RETURNS LOCATION */
      DECLARE C CHARACTER, T FIXED;
      PSYMBOL(PC)=C;  PTABLE(PC)=SHL(T,29) | SHL(GET_PCELL,12);
      PTABLE(PC)=PTABLE(PC) | PARMCELL;  PC=PC+1;  RETURN PC-1;
   END SET_PROC;


FIND_LABEL:
   PROCEDURE (N) FIXED;
        /* ENSURES ENTRY FOR LABEL N IN LABEL ARRAY
           RETURNS LOCATION IN LABEL ARRAY */
      DECLARE(L,CNT,N) FIXED;
      IF N=0 THEN CALL ERROR('LABEL ZERO CANNOT BE USED',2);
      L=N MOD 127;   CNT=0;
      DO FOREVER;
         IF (L=0) | (L=127) THEN L=1;
         IF LAB(L)=0 THEN DO; LAB(L)=N;  RETURN L;  END;
         IF LAB(L)=N THEN RETURN L;
         L=L+1;   CNT=CNT+1;
         IF CNT=127 THEN CALL ERROR('EXCESSIVE LABELS',2);  END;
   END FIND_LABEL;

SETLAB:
   PROCEDURE(L,POSITION) FIXED;
        /* RETURNS RELATIVE POSITION OF CODE WITHIN PRESENT PAGE
           WHERE FUTURE LABEL INSERTIONS SHALL BE MADE */
      DECLARE(L,POSITION,M) FIXED;
      ADVANCE(2);  M=GET_VCELL(1);
      CODE(M)=SHL(L,16) | POSITION;
      RETURN M-PAGE_BASE;
   END SETLAB;


SUBSCRIPT:
   PROCEDURE(T,L) FIXED;
        /* COMPUTE SUBSCRIPT OF ARRAY IN CODE BLOCK WITH BASE L OF TYPE T
           SUBSCRIPTS PASSED IN DIM ARRAY */
      DECLARE(L,N,T,SUM) FIXED;
      DT=DT+1;
      N=4096 - CODE(L); /* NUMBER OF DIMENSIONS */
      IF T<3 THEN T=1; /* INTEGER ARRAY ELSE REAL ARRAY */
      SUM=N+1+DIM(DT+N-1)*T+L-CODE(L+N);
      DT=DT+1;
      IF N=3 THEN DO; SUM=SUM+(DIM(DT)*CODE(L+1)+DIM(DT-1)*CODE(L+2))*T;
                      DT=DT+2;  END;
```

```
            ELSE IF N=2 THEN DO; SUM=SUM+(DIM(DT-1)*T*CODE(L+1));
                                 DT=DT+1;  END;
      RETURN SUM;
   END SUBSCRIPT;

INSERT_CHECK:
   PROCEDURE(T);
      /* CHECKS DATA INPUT FOR ASSIGNMENT COMPATABILITY */
      DECLARE(T) FIXED;
      IF FIXM(MP+1) > 0 THEN DO; IF T=1 THEN RETURN;  END;
                        ELSE DO; IF T=0 THEN RETURN;  END;
      CALL ERROR('VARIABLE AND NUMBER TYPE MUST AGREE',2);
   END INSERT_CHECK;

INSERT_DATA:
   PROCEDURE;
      /* INSERT DATA INTO PREVIOUSLY SAVED VARIABLES OF DIM ARRAY */
      DECLARE L FIXED;
      DIM(0)=DIM(0)+"100"; /* NUMBER COUNTER */
      TYPE=DIM(DT) & "F";
      L=SHR(DIM(DT),8); /* CORE LOCATION */
      DO CASE TYPE;
         /* CASE 0 INTEGER VARIABLE */
         DO; CODE(L)=FIXV(MP+1);  DT=DT+1;  CALL INSERT_CHECK(TYPE);  END;
         /* CASE 1 REAL VARIABLE  */
         DO; CALL INSERT_CHECK(TYPE);  GO TO INSERT_REAL;  END;
         /* CASE 2 INTEGER ARRAY */
         DO; L=SUBSCRIPT(TYPE,L);  CALL INSERT_CHECK(TYPE-2);
             CODE(L)=FIXV(MP+1);   END;
         /* CASE 3 REAL ARRAY */
         DO; L=SUBSCRIPT(TYPE,L);  CALL INSERT_CHECK(TYPE-2);
        INSERT_REAL:  CODE(L+1)=SHR(FIXM(MP+1),12) & "FFF";
         CODE(L)=FIXV(MP+1);  CODE(L+2)=(FIXM(MP+1) & "FFF");  END;  END;
   END INSERT_DATA;


DUMPING:
   PROCEDURE(FROM,LAST);
      /* OUTPUT PRT CELLS FROM TO LAST */
      DECLARE B CHARACTER, (FROM,I,J,LAST,M,N,P) FIXED;  B='            ';
      DO I=FROM TO LAST;
         OUTPUT=' '; IF SYMBOL(I)¬='ZZZZZZZ'  THEN DO; P=LOOKUP(SYMBOL(I));
         M=READ_OCTAL(SHR(PRT(P) & "FFF00",8)); /* LOCATION */
         IF TYPE>1 & PARM=0 THEN N=4096-CODE(SHR(PRT(P) & "FFF00",8));
                            ELSE N=0;
         OUTPUT='     '|| SYMBOL(I) ||B|| I ||B|| M ||B|| PARM || B ||
           SHR(SHL(PRT(P),3),23)||B||TYPE||B||N||B||P||B||SIZE(P-127);  END;
   END;
```

```
   END DUMPING;

DUMP:
   PROCEDURE;
      /* LIST PRT AND SYMBOL TABLE ENTRIES */
      DOUBLE_SPACE;
      OUTPUT='* * * COMBINED PRT/SYMBOL TABLE DUMP * * *';  OUTPUT=' ';
      OUTPUT='IDENTIFIER      SYMBOL#     BASE      PARAMETER      '||
             'COLLISION      TYPE      #DIM      PRT LOCATION     CELLS';
      CALL DUMPING(0,SC-1);
      CALL DUMPING(20,ST);
   END DUMP;

  /*                    THE SYNTHESIS ALGORITHM FOR XPL                          */


SYNTHESIZE:
PROCEDURE(PRODUCTION_NUMBER);
   DECLARE PRODUCTION_NUMBER FIXED;
      DECLARE (H,I,M,N,P,Z) FIXED;
   /*  THIS PROCEDURE IS RESPONSIBLE FOR THE SEMANTICS (CODE SYNTHESIS), IF
      ANY, OF THE SKELETON COMPILER.  ITS ARGUMENT IS THE NUMBER OF THE
      PRODUCTION WHICH WILL BE APPLIED IN THE PENDING REDUCTION.  THE GLOBAL
      VARIABLES MP AND SP POINT TO THE BOUNDS IN THE STACKS OF THE RIGHT PART
      OF THIS PRODUCTION.
       NORMALLY, THIS PROCEDURE WILL TAKE THE FORM OF A GIANT CASE STATEMENT
      ON PRODUCTION_NUMBER.  HOWEVER, THE SYNTAX CHECKER HAS SEMANTICS (THE
      TERMINATION OF CHECKING) ONLY FOR PRODUCTION 1.                          */
  DO CASE PRODUCTION_NUMBER;
 /*  PRODUCTION ZERO  */
   ;
 /*  .<MASTER PROGRAM> ::= <PROGRAM>     */
   DO; IF MAXTCELL > PARMCELL THEN CALL ERROR
         ('PARAMETER STORAGE OVERLAYED BY TEMPORARY CELLS',2);
      IF CONTROL(BYTE('T')) THEN DO;   DOUBLE_SPACE;
      OUTPUT='  '; OUTPUT='* * * PROCEDURE TABLE * * *'; OUTPUT='  ';
      OUTPUT=' PROCEDURE NAME     TYPE     INCLUDED=1     #PARAMETERS'||
        '          PARAMETER BASE          CORE LOCATION';
      DO I=1 TO PC-1;  M=SHR(SHL(PTABLE(I),4),28);
         IF M=0 THEN N=0;  ELSE N=SHR(SHL(PTABLE(I),20),20);
         OUTPUT='    ';  OUTPUT='      '||PSYMBOL(I)||'          '||
         SHR(PTABLE(I),29)||'                '||SHR(SHL(PTABLE(I),3),31)||
         '                '||M||'                              '||N||'              '||
         READ_OCTAL(SHR(SHL(PTABLE(I),8),20));  END;  END;
      COMPILING=FALSE;  DOUBLE_SPACE;
      IF STOP THEN CALL ERROR('STOP STATEMENT MISSING IN MAIN PROGRAM',0);
      IF BEGIN=0 THEN CALL ERROR('MISSING MAIN PROGRAM',2);
      N=CONTROL(BYTE('C'));     EP=0;
```

```
      IF N THEN OUTPUT='LOCATION    CODE';
      DO I=0 TO PAGE_BASE+127;
         M=CODE(I);
         IF M=4096 | M=4097 THEN CODE(I)=0; /* RESET FEXT & WEXT INSTRUCTIONS */
         IF N THEN OUTPUT='    '||READ_OCTAL(I)||'        '||READ_OCTAL(CODE(I));
         IF CODE(I)>4095 THEN CALL ERROR('CODE EMISSION ERROR',2);  END;
      DO I=PAGE_BASE+128 TO 2845;
         IF CODE(I)>4095 THEN CALL ERROR('CODE EMISSION ERROR',2);
         IF N & CODE(I) ¬= 0 THEN OUTPUT='    '||READ_OCTAL(I)||'        '||
            READ_OCTAL(CODE(I));  END;  END;

/*  <PROGRAM> ::= <STATEMENT BLOCK> END ;     */
  DO;
   PROG:
     DO I=PAGE_BLOCK TO PAGE_BASE+127; /* BACKSTUFF LABELS */
        M=SHR(CODE(I),16);
        IF M¬=0 THEN DO;
          IF  LAB(M+127)<2 THEN CALL ERROR('MISSING LABEL '||LAB(M),1);
          IF (CODE(I)&"1")=0 THEN N=SHR(LAB(M+127),16); /* FRONT */
                             ELSE N=LAB(M+127)&"FFFF"; /* REAR */
          CODE(I)=N;  END;  END;
     PAGE_BLOCK=CB;
     IF CB > 2944 THEN CALL ERROR
        ('PROGRAM TOO LARGE. ARRAYS AND FLOATING POINT PACKAGE OVERLAYED',0);
     IF CONTROL(BYTE('S')) THEN CALL DUMP;
     DO I=0 TO 126;
       LAB(I),LAB(I+127)=0;  IF PRT(I)>146 THEN PRT(I)=0;  END;
     ST=19;  PT=147;  EJECT_PAGE;  END;

/*  <PROGRAM> ::= <PROGRAM> <STATEMENT BLOCK> END ;     */
     GO TO PROG;

/*  <STATEMENT BLOCK> ::= <STATEMENT LIST>     */
  BEGIN=1;

/*  <STATEMENT BLOCK> ::= <DECLARATION LIST> <STATEMENT LIST>     */
  BEGIN=1;

/*  <STATEMENT BLOCK> ::= <PROCEDURE BLOCK> <STATEMENT LIST>    */
  DO;IF BEGIN=0 THEN CODE(129)=CB;        SFLAG=0;
     IF RFLAG THEN CALL ERROR('MISSING RETURN',2);  END;

/*  <STATEMENT LIST> ::= <STATEMENT> ;     */
  TCELL=62;

/*  <STATEMENT LIST> ::= <STATEMENT LIST> <STATEMENT> ;     */
  TCELL=62;
```

```
/*  <STATEMENT> ::= <IF STATEMENT>    */
  ;

/*  <STATEMENT> ::= <BASIC STATEMENT>    */
  ;

/*  <STATEMENT> ::= <DO STATEMENT>    */
  ;

/*  <STATEMENT> ::= <LABELED STATEMENT>    */
  ;

/*  <BASIC STATEMENT> ::= <ASSIGNMENT STATEMENT>    */
  ;

/*  <BASIC STATEMENT> ::= <GO STATEMENT>    */
  ;

/*  <BASIC STATEMENT> ::= <SUBROUTINE CALL>    */
  ;

/*  <BASIC STATEMENT> ::= <READ STATEMENT>    */
  ;

/*  <BASIC STATEMENT> ::= <WRITE STATEMENT>    */
  ;

/*  <BASIC STATEMENT> ::= RETURN    */
  DO;M=ENTRY & "FFFF"; /* PROCEDURE ENTRY POINT */
     IF (M >= PAGE_BASE) & (M < PAGE_BASE+128) THEN N=1;
      /* IF N=1 THEN ENTRY ON PRESENT PAGE */  ELSE N=0;
     IF SHR(ENTRY,16)=0 THEN
             DO; IF N=1 THEN EMIT(2944+M-PAGE_BASE); /* JMP I */
                        ELSE DO; ADVANCE(4);
                                 EMIT(896+CB-PAGE_BASE+3); /* TAD I */
                                 EMIT(1664+CB-PAGE_BASE+2); /* DCA */
                                 EMIT(2944+CB-PAGE_BASE+1); /* JMP I */
                                 CODE(CB)=M;  CB=CB+1;  END;
                 RFLAG=0;  RETURN;  END;
        /* ELSE MUST RETURN FUNCTION VARIABLE ADDRESS DURING RETURN */
     IF N=1 THEN DO; ADVANCE(3);
                     EMIT(896+CB-PAGE_BASE+2); /* TAD */
                     EMIT(2944+M-PAGE_BASE); /* JMP I */
                     CODE(CB)=SHR(ENTRY,16); CB=CB+1;   END;
            ELSE DO; ADVANCE(6);
                     EMIT(896+CB-PAGE_BASE+4); /* TAD I ENTRY */
                     EMIT(1664+CB-PAGE_BASE+3); /* DCA */
                     EMIT(640+CB-PAGE_BASE+3); /* TAD FUNCTION ADDRESS */
```

```
                        EMIT(2944+CB-PAGE_BASE+1); /* JMP I */
                        CODE(CB)=M;  CB=CB+2;
                        CODE(CB-1)=SHR(ENTRY,16);  END;
     RFLAG=0;  END;

/*  <BASIC STATEMENT> ::= STOP    */
  DO;EMIT(3842); /* HLT */
     IF SFLAG=0 THEN STOP=0;  END;

/*  <IF STATEMENT> ::= <ARITHMETIC IF> <DOUBLE LABEL> <NUMBER>    */
  DO;IF (LOC(MP)&"1")=1 THEN DO; ADVANCE(4);
                                 EMIT(2311); /* JMS I 7 */
                                 EMITCK(5,MP); /* FGET */
                                 EMIT(0000); /* FEXT */
                                 EMIT(549); /* TAD 45 */
                                 P=GET_TCELL(MP,1);
                                 EMIT(1536+P); /* DCA */  END;
                           ELSE EMITCK(1,MP); /* TAD */
     ADVANCE(5);
     EMIT(4032); /* SMA CLA */
     EMIT(2688+CB-PAGE_BASE+3); /* JMP I */
     EMIT(2944+CB-PAGE_BASE+1); /* JMP I */
     CODE(CB)=LOC(MP+1)&"FFFF0000"; /* EXPRESSION < ZERO */
     CB=CB+1;
     EMITCK(1,MP); /* TAD */
     ADVANCE(5);
     EMIT(4000); /* SZA CLA */
     EMIT(2944+CB-PAGE_BASE+3); /* JMP I */
     EMIT(2944+CB-PAGE_BASE+1); /* JMP I */
     CODE(CB)=SHL(LOC(MP+1),16); /* EXPRESSION = ZERO */
     CODE(CB+1)=SHL(FIND_LABEL(FIXV(SP)),16); /* EXPRESSION > ZERO */
   CB=CB+2;  END;

/*  <IF STATEMENT> ::= <LOGICAL IF> <BASIC STATEMENT>    */
     CODE(LOC(MP))=CB;

/*  <ARITHMETIC IF> ::= <IF> <EXPRESSION>    */
     LOC(MP)=LOC(MP+1);

/*  <IF> ::= IF (    */
  ;

/*  <DOUBLE LABEL> ::= ) <LABEL1> <LABEL1>    */
     LOC(MP)=SHL(LOC(MP+1),16) | LOC(SP);

/*  <LABEL1> ::= <NUMBER> ,    */
  LOC(MP)=FIND_LABEL(FIXV(MP));
```

```
/*  <EXPRESSION> ::= <TERM>     */
     IF AFLAG THEN DO; EMIT(4096); /* FEXT */
                       AFLAG=0;  END;

/*  <EXPRESSION> ::= <EXPRESSION> + <TERM>     */
  DO;Z=(LOC(MP) & "1") + (LOC(SP) & "1");
     DO CASE Z;
        /* CASE 0 BOTH INTEGERS */
        DO; EMITCK(1,MP); /* TAD */
            EMITCK(1,SP); /* TAD */
            P=GET_TCELL(MP,0);
            EMIT(1536+P); /* DCA */   END;
        /* CASE 1 ILLEGAL */
        CALL ERROR('ARITHMETIC TYPE INCOMPATABLE',2);
        /* CASE 2 BOTH REAL */
        DO; AFLAGCK;
            EMITCK(5,MP); /* FGET */
            EMITCK(1,SP); /* FADD */
            P=GET_TCELL(MP,1);
            EMIT(3072+P); /* FPUT */
            EMIT(4096); /* FEXT */  AFLAG=0;  END;  END;  END;

/*  <EXPRESSION> ::= <EXPRESSION> - <TERM>     */
  DO;Z=(LOC(MP) & "1") + (LOC(SP) & "1");
     DO CASE Z;
        /* CASE 0 BOTH INTEGERS */
        DO; EMITCK(1,SP); /* TAD */
            EMIT(3616); /* CMA */
            EMIT(516); /* TAD 4 */
            EMITCK(1,MP); /* TAD */
            P=GET_TCELL(MP,0);
            EMIT(1536+P); /* DCA */   END;
        /* CASE 1 ILLEGAL */
        CALL ERROR('ARITHMETIC TYPE INCOMPATABLE',2);
        /* CASE 2 BOTH REAL */
        DO; AFLAGCK;
            EMITCK(5,MP); /* FGET */
            EMITCK(2,SP); /* FSUB */
            P=GET_TCELL(MP,1);
            EMIT(3072+P); /* FPUT */
            EMIT(4096); /* FEXT */  AFLAG=0;  END;  END;  END;

/*  <EXPRESSION> ::= + <TERM>     */
  DO;LOC(MP)=LOC(MP+1);  EMIT(4096); /* FEXT */  AFLAG=0;  END;

/*  <EXPRESSION> ::= - <TERM>     */
  DO;IF (LOC(SP) & "1")=0
        THEN DO; EMITCK(1,SP); /* TAD */
```

```
                    EMIT(3616); /* CMA */
                    EMIT(516); /* TAD 4 */
                    P=GET_TCELL(MP,0);
                    EMIT(1536+P); /* DCA */  END;
                DO; AFLAGCK;
                    EMIT(2618); /* FGET 72 */
                    EMITCK(2,SP); /* FSUB */
                    P=GET_TCELL(MP,1);
                    EMIT(3072+P); /* FPUT */
                    EMIT(4096); /* FEXT */  AFLAG=0;  END;  END;

/*  <TERM> ::= <PRIMARY>    */
  ;

/*  <TERM> ::= <PRIMARY*> <PRIMARY>    */
  DO;Z=(LOC(MP) & "1") + (LOC(SP) & "1");
     DO CASE Z;
        /* CASE 0 BOTH INTEGERS */
        DO; ADVANCE(7);
            EMIT(2358); /* JMS I 66 */
            EMITA(MP,1); /* ADDRESS */
            EMITA(SP,2); /* ADDRESS */
            P=GET_TCELL(MP,0);
            EMIT(1536+P); /* DCA */   END;
        /* CASE 1 ILLEGAL */
        CALL ERROR('ARITHMETIC TYPE INCOMPATABLE',2);
        /* CASE 2 BOTH REAL */
        DO; AFLAGCK;
            EMITCK(5,MP); /* FGET */
            EMITCK(3,SP); /* FMPY */
            P=GET_TCELL(MP,1);
            EMIT(3072+P); /* FPUT */  END;  END;  END;

/*  <TERM> ::= <TERM> / <PRIMARY>    */
  DO;Z=(LOC(MP) & "1") + (LOC(SP) & "1");
     DO CASE Z;
        /* CASE 0 BOTH INTEGERS */
        DO; ADVANCE(7);
            EMIT(2359); /* JMS I 67 */
            EMITA(MP,1); /* ADDRESS */
            EMITA(SP,2); /* ADDRESS */
            P=GET_TCELL(MP,0);
            EMIT(1536+P); /* DCA */   END;
        /* CASE 1 ILLEGAL */
        CALL ERROR('ARITHMETIC TYPE INCOMPATABLE',2);
        /* CASE 2 BOTH REAL */
        DO; AFLAGCK;
            EMITCK(5,MP); /* FGET */
```

```
            EMITCK(4,SP); /* FDIV */
            P=GET_TCELL(MP,1);
            EMIT(3072+P); /* FPUT */  END;  END;  END;

/*  <PRIMARY> ::= <SECONDARY>    */
  ;

/*  <PRIMARY> ::= <PRIMARY*> * <SECONDARY>    */
  DO;IF (LOC(SP) &"1")=1 THEN CALL ERROR
       ('EXPONENT MUST EVALUATE TO AN INTEGER',2);
     IF (LOC(MP) & "1")=0 THEN DO;
       IF (TCELL+2) > PARMCELL THEN CALL ERROR
          ('CODE BEING OVERLAYED. SEPARATE INTO ADDITIONAL STATEMENTS',2);
       EMITCK(1,SP); /* TAD */
       EMIT(3616); /* CMA */
       EMIT(516); /* TAD 4 */
       EMIT(1536+TCELL+1); /* DCA */
       EMITCK(1,MP); /* TAD */
       EMIT(1536+TCELL); /* DCA */
       ADVANCE(8);
       LOC(MP+1)=SHL(CB,4); /* SAVE NEXT ADDRESS */
       EMIT(2358); /* JMS I 66 */
       EMITA(MP,1); /* ADDRESS */
       EMIT(TCELL); /* ADDRESS */
       EMIT(1536+TCELL); /* DCA */
       EMIT(1024+TCELL+1); /* ISZ */
       EMIT(2688+CB-PAGE_BASE-5); /* JMP */
       CALL GET_TCELL(MP,0);   END;
      ELSE DO;
       IF AFLAG THEN DO; CODE(CB)=0; /* FEXT */
                         CB=CB+1;  AFLAG=0;  END;
       ADVANCE(7);
       EMIT(2365); /* JMS 75 */
       EMITA(SP,1); /* EXPONENT ADDRESS */
       EMITA(MP,2); /* NUMBER ADDRESS */
       P=GET_TCELL(MP,1);
       AFLAGCK;
       EMIT(3072+P); /* FPUT */  END;  END;

/*  <PRIMARY*> ::= <SECONDARY> *    */
  ;

/*  <SECONDARY> ::= <VARIABLE>    */
  IF LOOKUP(VAR(MP))=0 THEN
     DO; IF VAR(MP+1)=BYTE('=') THEN RETURN;
         IF FIND_PROC(VAR(MP)) > 0 THEN RETURN;
         CALL ERROR('UNKNOWN VARIABLE:  '||VAR(MP),2);  END;
```

```
/*  <SECONDARY> ::= <NUMBER>    */
  DO;IF DFLAG¬=3 THEN RETURN;
     IF FIXM(MP)=0 THEN DO; /* NUMBER LESS THAN 12 BITS */
                            P=STORE_CONSTANT(0,FIXV(MP),1);
                            LOC(MP)=SHL(P, 4) | 0;  END;
                   ELSE DO; P=STORE_CONSTANT(FIXV(MP),(FIXM(MP)&"FFFFFF"),3);
                            LOC(MP)=SHL(P, 4) | 1;  END;  END;

/*  <SECONDARY> ::= ( <EXPRESSION> )    */
  LOC(MP)=LOC(MP+1);

/*  <SECONDARY> ::= ABS ( <EXPRESSION> )    */
  IF (LOC(SP) & "1")=0 THEN
        DO; EMITCK(1,SP-1); /* TAD */
            ADVANCE(3);
            EMIT(3912); /* SPA */
            EMIT(3616); /* CMA */
            P=GET_TCELL(MP,0);
            EMIT(1536+P); /* DCA */  END;
        ELSE DO; AFLAGCK;
                 ADVANCE(3);
                 EMITCK(5,SP-1); /* FGET */
                 P=GET_TCELL(MP,1);
                 EMITCK(6,MP); /* FPUT */
                 EMIT(0000); /* FEXT */
                 EMIT(512+P+1); /* TAD */
                 EMIT(3904); /* SMA */
                 ADVANCE(5);
                 EMIT(2688+CB-PAGE_BASE+4); /* JMP */
                 EMIT(3588); /* RAL */
                 EMIT(3648); /* CLL */
                 EMIT(3592); /* RAR */
                 EMIT(1536+P+1); /* DCA */  END;

/*  <SECONDARY> ::= SQR ( <EXPRESSION> )    */
  DO;IF (LOC(SP) & "1")=0 THEN CALL ERROR
        ('SQR REQUIRES REAL EXPRESSION',2);
     AFLAGCK;
     EMITCK(5,SP-1); /* FGET */
     EMIT(0001); /* SQR */
     P=GET_TCELL(MP,1);
     EMITCK(6,MP); /* FPUT */  END;

/*  <SECONDARY> ::= SQRT ( <EXPRESSION> )    */
  DO;IF (LOC(SP) & "1")=0 THEN CALL ERROR
        ('SQRT REQUIRES REAL EXPRESSION',2);
     AFLAGCK;
     EMITCK(5,SP-1); /* FGET */
```

```
      EMIT(0002); /* SQRT */
      P=GET_TCELL(MP,1);
      EMITCK(6,MP); /* FPUT */  END;

/*  <SECONDARY> ::= FLOAT ( <EXPRESSION> )    */
  DO;IF (LOC(SP-1) & "1")=1 THEN
       DO; CALL ERROR('EXPRESSION ALREADY DECLARED REAL',0);
         RETURN;  END;
      EMITCK(1,SP-1); /* TAD */
      EMIT(2331); /* JMS I 33 */
      AFLAGCK;
      CALL GET_TCELL(MP,1);
      EMITCK(6,MP); /* FPUT */  END;

/*  <LOGICAL IF> ::= <IF> <BOOLEAN EXPRESSION> )    */
  DO;EMIT(512+LOC(MP+1)); /* TAD */
      EMIT(514); /* TAD 2 */
      P=GET_VCELL(1);
      LOC(MP)=P; /* SAVE ADDRESS FOR JMP */
      ADVANCE(3);
      EMIT(4000); /* SZA CLA */
      EMIT  (2944+P-PAGE_BASE); /* JMP */
      TCELL=62;  END;

/*  <BOOLEAN EXPRESSION> ::= <BOOLEAN TERM>    */
  ;

/*  <BOOLEAN EXPRESSION> ::= <BOOLEAN EXPRESSION> .OR. <BOOLEAN TERM>    */
  DO;EMIT(512+LOC(MP)); /* TAD */
      EMIT(3857); /* MQL */
      EMIT(512+LOC(SP)); /* TAD */
      EMIT(3905); /* MQA */
      EMIT(1536+LOC(MP));  END;

/*  <BOOLEAN TERM> ::= <BCOLEAN PRIMARY>    */
  ;

/*  <BOOLEAN TERM> ::= .NOT. <BOOLEAN PRIMARY>    */
  DO;LOC(MP)=LOC(MP+1);
      EMIT(512+LOC(MP)); /* TAD */
      EMIT(514); /* TAD */
      EMIT(4000); /* SZA CLA */
      EMIT(3713); /* CLA IAC */
      EMIT(1536+LOC(MP)); /* DCA */  END;

/*  <BOOLEAN TERM> ::= <BOOLEAN TERM> .AND. <BCOLEAN PRIMARY>    */
  DO;EMIT(512+LOC(MP)); /* TAD */
      EMIT(LOC(SP)); /* AND */
```

```
     EMIT(1536+LOC(MP)); /* DCA */  END;

/*  <BOOLEAN PRIMARY> ::= <LOGICAL EXPRESSION>     */
  ;

/*  <BOOLEAN PRIMARY> ::= ( <BOOLEAN EXPRESSION> )     */
  LOC(MP)=LOC(MP+1);

/*  <LOGICAL EXPRESSION> ::= <EXPRESSION> <RELATION> <EXPRESSION>     */
  DO;Z=(LOC(MP) & "1") + (LOC(SP) & "1");
     IF LOC(MP+1)>2 THEN DO; M=LOC(SP); LOC(SP)=LOC(MP); LOC(MP)=M; END;
     EMIT(3776); /* CLA CLL */
     DO CASE Z;
        /* CASE 0 BOTH INTEGERS */
        DO; EMITCK(1,SP); /* TAD */
            EMIT(3616); /* CMA */
            EMIT(516); /* TAD 4 */
            EMITCK(1,MP); /* TAD */   END;
        /* CASE 1 ILLEGAL */
        CALL ERROR('RELATION REQUIRES ARITHMETIC TYPE COMPATABILITY',2);
        /* CASE 2 BOTH REAL */
        DO; ADVANCE(6);
            EMIT(2311); /* JMS I 7 */
            EMITCK(5,MP); /* FGET */
            EMITCK(2,SP); /* FSUB */
            EMIT(0000); /* FEXT */
            EMIT(549); /* TAD 45 */   END;   END;
        ADVANCE(4);
     DO CASE (LOC(MP+1) & "F");
        /* CASE 0 .EQ. & .NE. */
            IF VAR(MP+1)='.EQ.' THEN EMIT(4000); /* SZA CLA */
                               ELSE EMIT(4008); /* SNA CLA */
        /* CASE 1 .LT. & .GT. */
            EMIT(4032); /* SMA CLA */
        /* CASE 2 .LE. & .GE. */
            EMIT(4040); /* SPA CLA */  END;
     M,LOC(MP)=GET_TCELL(MP,0);
     EMIT(2688+CB-PAGE_BASE+2); /* JMP */
     EMIT(3713); /* CLA IAC */
     EMIT(1536+M); /* DCA */  END;

/*  <RELATION> ::= .LT.     */  /* STORE A FLAG FOR LOGICAL EXPRESSION */
  LOC(MP)=1;
/*  <RELATION> ::= .LE.     SET REVERSE SUBTRACTION */
  LOC(MP)=2 | SHL(1,8);
/*  <RELATION> ::= .EQ.     */
  LOC(MP)=0;
/*  <RELATION> ::= .NE.     */
```

```
  LOC(MP)=0;
/*  <RELATION> ::= .GT.     SET REVERSE SUBTRACTION */
  LOC(MP)=1 | SHL(1,8);
/*  <RELATION> ::= .GE.     */
  LOC(MP)=2;

/*  <LABELED STATEMENT> ::= <LABEL2> <STATEMENT>    */
  DO;
    LABELED_STATEMENT:
     M=LOC(MP); /* SAVED LABEL */
     IF (LAB(M+127)&"FFFF")=0 THEN DO; LAB(M+127)=LAB(M+127)|CB; RETURN; END;
     ADVANCE(3);
     EMIT(2944+CB-PAGE_BASE+1); /* JMP I */
     CODE(CB)=LAB(M+127) & "FFFF"; /* BACKSTUFF DO STATEMENT */
     CB=CB+1;
     LAB(M+127)=SHL(SHR(LAB(M+127),16),16) | CB;  END;

/*  <LABELED STATEMENT> ::= <LABEL2> CONTINUE    */
     GO TO LABELED_STATEMENT;

/*  <LABEL2> ::= <NUMBER>    */
  DO;LOC(MP),M=FIND_LABEL(FIXV(MP));
     LAB(M+127)=LAB(M+127) | SHL(CB,16);  END;

/*  <ASSIGNMENT STATEMENT> ::= <VARIABLE> <RIGHT PART>    */
  DO;IF LOOKUP(VAR(MP))=0 THEN DO; M=ENTER1(VAR(MP),MP);
              LOC(MP)=SHR(PRT(M)&"FFF00",4)|TYPE;  END;
     IF (LOC(MP)&"1")+(LOC(SP)&"1")=1 THEN
        CALL ERROR('ASSIGNMENT INCOMPATABLE',2);
     IF (LOC(MP)&"1")=1 THEN DO; EMITCK(6,MP); /* FPUT */
                                 EMIT(4096); /* FEXT */
                                 AFLAG=0;    END;
                        ELSE EMITCK(3,MP); /* DCA */  END;

/*  <RIGHT PART> ::= = <EXPRESSION>    */
  DO;LOC(MP)=LOC(MP+1);
     IF (LOC(MP) & "1")=1 THEN
       DO; ADVANCE(6); /* ALLOWS MAX OF TRIPLE ASSIGNMENT */
           AFLAGCK;
           EMITCK(5,MP); /* FGET */  END;
       ELSE EMITCK(1,MP); /* TAD */  END;

/*  <RIGHT PART> ::= = <VARIABLE> <RIGHT PART>    */
  DO;IF LOOKUP(VAR(MP+1))=0 THEN DO; M=ENTER1(VAR(MP+1),MP+1);
              LOC(MP)=SHR(PRT(M)&"FFF00",4)|TYPE;  END;
            ELSE LOC(MP)=LOC(MP+1);
     LOC(MP+1)=LOC(SP);
     IF (LOC(MP)&"1")+(LOC(SP)&"1")=1 THEN
```

```
         CALL ERROR('ASSIGNMENT INCOMPATABLE',2);
      IF (LOC(MP)&"1") =1 THEN EMITCK(6,MP); /* FPUT */
                          ELSE DO; EMITCK(3,MP); /* DCA */
                                   EMITCK(1,MP); /* TAD */  END;  END;

/*  <VARIABLE> ::= <IDENTIFIER>     */
  DO;IF VAR(SP-1)='CALL' THEN DO; /* PARAMETERLESS SUBROUTINE CALL */
      M=FIND_PROC(VAR(MP));
      IF M=0 THEN M=SET_PROC(VAR(MP),2);
      EMIT(2304+SHR(PTABLE(M) & "FFF000",12)); /* JMS I */
      RETURN;  END;
   M=LOOKUP(VAR(MP));
   IF DFLAG=5 THEN RETURN; /* COMMON STATEMENT */
   IF DFLAG=4 THEN DO; /* DATA DECLARATION */
      IF M=0 THEN CALL ENTER1(VAR(MP),MP);
      DIM(DT)=TYPE|(PRT(M) & "FFF00"); /* SAVE TYPE & CORE LOCATION */
      DT=DT+1;
      DIM(0)=DIM(0)+1; /* DATA VARIABLE COUNTER */
      RETURN;      END;
   IF DFLAG=3 THEN DO;
      IF M¬=0 THEN LOC(MP)=(SHL(SHR(SHL(PRT(M),12),20),4)|TYPE)|SHL(PARM,2);
      RETURN;     END;
   IF M=0 THEN M=ENTER(VAR(MP),DFLAG);
          ELSE PRT(M)=SHR(SHL(PRT(M),2),2) | SHL(DFLAG,30);
   IF SFLAG=1 THEN RETURN;
   IF TYPE=1 THEN DO; N=GET_VCELL(3); LOC(MP)=SHL(N,4)|1; END;
             ELSE DO; N=GET_VCELL(1); LOC(MP)=SHL(N,4)|0; END;
   PRT(M)=SHL(N,8) | PRT(M);  END;

/*  <VARIABLE> ::= <SUBSCRIPT HEAD> <EXPRESSION> )     */
  DO;N=(FIXV(MP) & "F") +1; /* COUNTER */
   IF DFLAG=3 THEN
      DO; /* NOT A DECLARATION */
         IF LENGTH(VAR(SP-1))¬=0 THEN
           DO; M=LOOKUP(VAR(SP-1));
               IF M=0 THEN M=ENTER1(VAR(SP-1),SP-1);  END;
         P=SHR(FIXV(MP),8); /* LOCATION */
         IF SHR(FIXV(MP) & "F0",4) <2 THEN
            DO; /* PROCEDURE CALL NOTE CFLAG NOW 1 FOR SUBROUTINE      */
               ADVANCE(3);
               I=GET_VCELL(1);
               CODE(I)=SHR(LOC(MP+1),4);
               EMIT(640+I-PAGE_BASE); /* TAD */
               EMIT(1536+NEXT); /* DCA */
               IF NEXT-1<PARMCELL THEN PARMCELL=NEXT-1;  NEXT=0;
               M=SHR(SHL(PTABLE(P), 4),28); /* # PARAMETERS */
               IF M=0 THEN PTABLE(P)=PTABLE(P) | SHL(N,24);
                      ELSE IF M¬=N THEN
```

```
                    CALL ERROR('PARAMETER COUNT DOES NOT AGREE',2);
                N=SHR(SHL(PTABLE(P), 8),20); /* OCTAL REFERENCE */
                EMIT(2304+N); /* JMS I TO SUBPROGRAM */
                IF CFLAG=0 THEN DO; /* FUNCTION CALL,STORE RETURNED VALUE */
                  N=SET(VAR(MP));
                  IF N=0 THEN P=GET_TCELL(1);
                         ELSE P=GET_TCELL(3);
                  EMIT(1664+P); /* DCA */
                  LOC(MP)=SHL(P,4) | N+4;  END;  END;
              ELSE DO; /* COMPUTE SUBSCRIPTS */
                    IF (LOC(MP+1) & "1")=1 THEN CALL ERROR
                      ('SUBSCRIPTING REQUIRES INTEGER EXPRESSION',2);
                    EMITCK(1,MP+1); /* TAD */
                    IF SHR(SHL(FIXV(MP),24),28)=3 THEN EMIT(3616);/* CMA */
                    EMIT(1587); /* DCA 63 */
                    M=52;  DT=DT-1;
                    DO WHILE DT¬=-1;
                      LOC(MP+1)=DIM(DT);
                      EMITCK(1,MP+1); /* TAD */
                      EMIT(1536+M); /* DCA */
                      DT=DT-1;    M=M+1;    END;
                    ADVANCE(4);
                    M=SHR(FIXV(MP),8); /* PRT ADDRESS */
                    N=(SHR(SHL(PRT(M),12),20)); /* ARRAY BASE */
                    IF (PRT(M) & "20000000") > 0 THEN
                      DO; EMIT(512+N); /* TAD */
                          EMIT(1664+CB-PAGE_BASE+2); /* DCA */
                          EMIT(2354); /* JMS I 62 */
                          CB=CB+1;  END;
                      ELSE DO; EMIT(2354);
                               EMIT(N);  END;
                    P=GET_TCELL(MP,0);
                    EMIT(1536+P); /* DCA */
                    P=SHR(SHL(FIXV(MP),24),28);
                    LOC(MP)=LOC(MP)|4+P-2;  END;
          RETURN;  END;
IF DFLAG=4 THEN DO; /* DATA DECLARATION */
   IF FIXV(MP+1)=0 THEN CALL ERROR
     ('DATA VARIABLE SUBSCRIPT MUST BE INTEGER NUMBER',2);
   DIM(DT)=FIXV(MP+1);
   DT=DT+1;  RETURN;  END;
IF (FIXV(MP+1))=0 THEN CALL ERROR('SUBSCRIPTS MUST BE INTEGER NUMBERS',2);
DIM(DT)=FIXV(SP-1); /* INSERT LAST DIMENSION INTO DIM ARRAY */
P=LOOKUP(VAR(MP));
IF P=0 THEN DO; IF DFLAG=2 THEN P=ENTER(VAR(MP),SET(VAR(MP))+2);
                           ELSE P=ENTER(VAR(MP),DFLAG+2);
                TYPE=SHR(PRT(P),30);  END;
     ELSE DO; PRT(P)=PRT(P)+"80000000"; /* DESIG PARAMETER AN ARRAY */
```

```
                    M=1;
                    DO I=0 TO DT;  M=M * DIM(I);  END;
                    SIZE(P-127)=M+DT+2;
                     RETURN;      END;
        IF TYPE=3 THEN DO; DIM(DT+1)=3; LOC(MP)=1; END;
                  ELSE DO; DIM(DT+1)=1; LOC(MP)=0; END;
        DO I=1 TO N-1; /* COMPUTE D SUB I */
           DIM(DT+I+1)=DIM(DT-I+1) * DIM(DT+I);   END;
        M=DT+N+1;
        DIM(M)=DIM(DT+1);
        DO I=0 TO DT;  /* COMPUTE TOTAL CELLS REQUIRED FOR ARRAY */
           DIM(M)=DIM(M) * DIM(I);    END;
        Z=GET_ACELL(DIM(M)+1+N); /* Z IS BASE OF ARRAY BLOCK */
        SIZE(P-127)=DIM(M)+N+1;
        LOC(MP)=LOC(MP) | SHL(Z,4);
        PRT(P)=PRT(P) | SHL(Z,8);
        CODE(Z)=4095-N+1; /* NEG OCTAL NUMBER=NUMBER OF SUBSCRIPTS */
        DO I=1 TO N-1;
           CODE(Z+I)=DIM(DT+I+1);   END;       H=0;
        DO I=1 TO N; /* SUM D SUB I */
           H=H+DIM(DT+I);   END;
        CODE(Z+N)=H;  END;

/*  <SUBSCRIPT HEAD> ::= <IDENTIFIER> (    */
  DO;M=LOOKUP(VAR(MP));
        IF DFLAG=4 THEN DO; /* DATA DECLARATION */
           IF M=0 THEN CALL ERROR('DATA DECLARATIONS REQUIRE ARRAY BE KNOWN',2);
           DIM(DT)=TYPE|(PRT(M) & "FFF00"); /* SAVE TYPE & CORE LOCATION */
           DT=DT+1;
           DIM(0)=DIM(0)+1; /* DATA VARIABLE COUNTER */
           RETURN;   END;
        IF DFLAG=3 THEN DO;  /* NOT A DECLARATION */
           IF M=0 THEN
              DO; /* PROCEDURE CALL */
                  IF NEXT > 0 THEN CALL ERROR
                    ('FUNCTION CALLS WITHIN SUBPROGRAM CALLS NOT ALLOWED',2);
                  M=FIND_PROC(VAR(MP));
                  IF M=0 THEN M=SET_PROC(VAR(MP),CFLAG+1);
                  NEXT=PTABLE(M) & "FFF";
                  IF CFLAG=0 THEN FIXV(MP)=SHL(M,8); /* FUNCTION */
                             ELSE FIXV(MP)="10"|SHL(M,8); /* SUBROUTINE */  END;
            ELSE DO; /* MUST COMPUTE SUBSCRIPT OF ARRAY */
                     IF M=0 THEN CALL ERROR('UNKNOWN ARRAY',2);
                     FIXV(MP)=SHL(M, 8) | SHL(TYPE,4);
                     DT=0;  END;
            RETURN;  END;  /* MUST DIMENSION AN ARRAY */
        DT=0;  FIXV(MP)=0;  END;
```

```
/*  <SUBSCRIPT HEAD> ::= <SUBSCRIPT HEAD> <EXPRESSION> ,     */
  DO;IF DFLAG=3 THEN DO; /* NOT A DECLARATION */
        IF SHR(FIXV(MP) &"F0",4)<2 THEN
           DO; /* A PROCEDURE */
              IF LENGTH(VAR(SP-1))¬=0 THEN
                 DO; M=LOOKUP(VAR(SP-1));
                     IF M=0 THEN M=ENTER1(VAR(SP-1),SP-1);  END;
               FIXV(MP)=FIXV(MP)+1; /* PARAMETER COUNT */
               ADVANCE(3);
               I=GET_VCELL(1);
               CODE(I)=SHR(LOC(MP+1),4);
               EMIT(5640+I-PAGE_BASE); /* TAD */
               EMIT(1536+NEXT); /* DCA */
               NEXT=NEXT-1;  END;
            ELSE DO; /* MUST COMPUTE SUBSCRIPT */
                     FIXV(MP)=FIXV(MP) + 1;    /* DIMENSION COUNTER */
                     IF (LOC(MP+1) & "1")=1 THEN CALL ERROR
                       ('SUBSCRIPTING REQURES INTEGER EXPRESSION',2);
                     DIM(DT)=LOC(MP+1);
                     DT=DT+1;  END;
           RETURN;  END;
     IF DFLAG=4 THEN DO; /* DATA DECLARATION */
        IF LENGTH(VAR(MP+1))¬=0 THEN CALL ERROR
          ('DATA VARIABLE SUBSCRIPT MUST BE INTEGER NUMBER',2);
        DIM(DT)=FIXV(MP+1);
        DT=DT+1;
        RETURN;   END;
     IF (FIXV(MP+1))=0 THEN CALL ERROR('SUBSCRIPTS MUST BE INTEGER NUMBERS',2);
     DIM(DT)=FIXV(SP-1);
     DT=DT+1;
     FIXV(MP)=FIXV(MP)+ 1;  END;

/*  <DO STATEMENT> ::= <DO_HEAD>     */
  DO;IF (LOC(MP) & "1")=0 THEN STEP=SHL(4,4); /* FIXED POINT ONE */
        ELSE DO; N=STORE_CONSTANT(1,4194304,3); /* FLOATING POINT ONE */
                 STEP=SHL(N,4) | 1;  END;
    DO_HEAD:
     Z=(LOC(MP) & "1") + (STEP & "1") + (DO_UNTIL & "1");
     IF Z=3 THEN Z=1;
      ELSE IF Z¬=0 THEN CALL ERROR('DO EXPRESSIONS ASSIGNMENT INCOMPATABLE',2);
     DO CASE Z;
        /* CASE 0 INTEGER */
        DO; EMITCK(1,MP); /* TAD */
            M=LOC(MP); /* SAVE MP FOR PASSING PARAMETERS */
            LOC(MP)=STEP;
            EMITCK(1,MP); /* TAD */
            LOC(MP)=M;
            EMITCK(3,MP); /* DCA */
```

```
                M=LOC(MP);
                LOC(MP)=DO_UNTIL;
                EMITCK(1,MP); /* TAD */
                EMIT(3616); /* CMA */
                LOC(MP)=M;
                EMITCK(1,MP); /* TAD */  END;
            /* CASE 1 REAL */
            DO; ADVANCE(11);
                AFLAGCK;
                EMITCK(5,MP); /* FGET */
                M=LOC(MP);
                LOC(MP)=STEP;
                EMITCK(1,MP); /* FADD */
                LOC(MP)=M;
                EMITCK(6,MP); /* FPUT */
                M=LOC(MP);
                LOC(MP)=DO_UNTIL;
                EMITCK(2,MP); /* FSUB */
                EMIT(568); /* FADD 70 */
                EMIT(0000); /* FEXT */
                LOC(MP)=M;
                EMIT(549); /* TAD 45 */  END;  END;
        ADVANCE(3);
        EMIT(4032); /* SMA CLA */
        P=SETLAB(SAVE_LABEL_ADDRESS,1);
        EMIT(2944+P); /* JMP I */
        CODE(SAVE_FIRST)=CB;  END;

/*  <DO STATEMENT> ::= <DO HEAD> , <EXPRESSION>    */
  DO;STEP=LOC(SP);  GO TO DO_HEAD;  END;

/*  .<DO HEAD> ::= <DO VARIABLE> , <EXPRESSION>    */
      DO_UNTIL=LOC(SP);

/*  <DO VARIABLE> ::= <DO LABEL> <VARIABLE> = <EXPRESSION>    */
  DO;N=LOOKUP(VAR(MP+1));
      IF N=0 THEN CALL ENTER1(VAR(MP+1),MP+1);
      IF (LOC(MP+1) & "1") + (LOC(SP) & "1")=1 THEN
         CALL ERROR('DO VARIABLE ASSIGNMENT INCOMPATABLE',2);
      DO CASE (LOC(SP) & "1");
         /* CASE 0 INTEGER ASSIGNMENT */
         DO; EMITCK(1,SP); /* TAD */
             EMITCK(3,MP+1); /* DCA */  END;
         /* CASE 1 REAL ASSIGNMENT */
         DO; ADVANCE(6);
             EMIT(2311); /* JMS I 7 */
             EMITCK(5,SP); /* FGET */
             EMITCK(6,MP+1); /* FPUT */
```

```
            EMIT(0000); /* FEXT */  END;  END;
      ADVANCE(2);
      EMIT(2944+CB-PAGE_BASE+1); /* JMP TO FIRST STATEMENT */
      SAVE_FIRST=CB; CB=CB+1; /* STUFF WITH ADDRESS OF FIRST STATEMENT */
      IF LAB(LOC(MP)+127)¬=0 THEN DO;  CALL ERROR(' ',2);
         OUTPUT='NESTED DO LOOPS MUST NOT END ON SAME LABELED STATEMENT';
         OUTPUT=' --OR- DUPLICATE LABEL'; OUTPUT=' ';  END;
      LAB(LOC(MP)+127)=CB; /* RETURN FOR INCREMENT */
      LOC(MP)=LOC(MP+1); /* SAVE THE VARIABLE */  END;

/*  <DO LABEL> ::= DO <NUMBER>    */
  LOC(MP),SAVE_LABEL_ADDRESS=FIND_LABEL(FIXV(SP));

/*  <GO STATEMENT> ::= <GOTO> <NUMBER>    */
  DO;M=FIND_LABEL(FIXV(SP));
      P=SETLAB(M,0);
      EMIT(2944+P); /* JMP I */ END;

/*  <GO STATEMENT> ::= <GO TRANSFER> <END GO> <VARIABLE>    */
  DO;IF (LOC(SP) & "1")=1 THEN CALL ERROR('VARIABLE MUST BE INTEGER TYPE',2);
      EMITCK(1,SP); /* TAD */
      ADVANCE(DT+3);
      EMIT(640+CB-PAGE_BASE+3); /* TAD */
      EMIT(1664+CB-PAGE_BASE+1); /* DCA */
      CB=CB+1;
      EMIT(2944+CB-PAGE_BASE); /* JMP - LABELS FOLLOW */
      DO I=0 TO DT-1;
         CODE(CB)=SHL(DIM(I),16);  CB=CB+1;  END;  END;

/*  <GOTO> ::= GO TO    */
  ;

/*  <GOTO> ::= GOTO    */
  ;

/*  <GO TRANSFER> ::= <GOTO> <PAREN> <NUMBER>    */
  DO;DT=0;  DIM(DT)=FIND_LABEL(FIXV(SP));  DT=DT+1;  END;

/*  <GO TRANSFER> ::= <GO TRANSFER> <COMMA> <NUMBER>    */
  DO;DIM(DT)=FIND_LABEL(FIXV(SP));  DT=DT+1;  END;

/*  <PAREN> ::= (    */
  ;

/*  <COMMA> ::= ,    */
  ;

/*  <END GO> ::= ) ,    */
```

```
    ;
/*  <DECLARATION LIST> ::= <DECLARATION> ;     */
    DFLAG=3;

/*  <DECLARATION LIST> ::= <DECLARATION LIST> <DECLARATION> ;     */
    DFLAG=3;

/*  <DECLARATION> ::= <DECLARATION TYPE> <VARIABLE>     */
    IF DFLAG=5 THEN CALL COMMON_CHECK(SP);

/*  <DECLARATION> ::= <DECLARATION TYPE> <VARIABLE LIST> <VARIABLE>     */
    IF DFLAG=5 THEN CALL COMMON_CHECK(SP);

/*  <DECLARATION> ::= <DATA DECLARATION> <NUMBER> /     */
    DO;CALL INSERT_DATA;
       IF (DIM(0) & "FF")¬=SHR(DIM(0),8) THEN CALL ERROR
          ('NUMBER OF VARIABLES AND DATA DO NOT MATCH',2);   END;

/*  <DECLARATION TYPE> ::= DIMENSION     */
    DFLAG=2;

/*  <DECLARATION TYPE> ::= INTEGER     */
    DFLAG=0;

/*  <DECLARATION TYPE> ::= REAL     */
    DFLAG=1;

/*  <DECLARATION TYPE> ::= COMMON     */
    DFLAG=5;

/*  <VARIABLE LIST> ::= <VARIABLE> ,     */
    IF DFLAG=5 THEN CALL COMMON_CHECK(SP-1);

/*  <VARIABLE LIST> ::= <VARIABLE LIST> <VARIABLE> ,     */
    IF DFLAG=5 THEN CALL COMMON_CHECK(SP-1);

/*  <DATA DECLARATION> ::= <DATA HEAD> /     */
    DT=1;

/*  <DATA DECLARATION> ::= <DATA DECLARATION> <NUMBER> ,     */
    CALL INSERT_DATA;

/*  <DATA HEAD> ::= <DATA> <VARIABLE>     */
    ;

/*  <DATA HEAD> ::= <DATA> <VARIABLE LIST> <VARIABLE>     */
    ;
```

```
/*  <DATA> ::= DATA     */
  DO;DT=1;
     DIM(0)=0; /* VARIABLE COUNTER */
     DFLAG=4;   END;

/*  <PROCEDURE BLOCK> ::= <PROCEDURE HEADING>  */
  DO;
    PROCEDURE_HEADING:
     P=SHR(FIXV(MP),8); /* PTABLE ENTRY */
     P=SHR(PTABLE(P) & "FFF000",12); /* OCTAL REFERENCE ON PAGE ZERO */
     CODE(P)=ENTRY & "FFF";  END;

/*  <PROCEDURE BLOCK> ::= <PROCEDURE HEADING> <DECLARATION LIST>  */
  GO TO PROCEDURE_HEADING;

/*  <PROCEDURE HEADING> ::= <PARAMLESS PROCEDURE>    */
  ;

/*  <PROCEDURE HEADING> ::= <PROCEDURE & PARAMETERS>    */
  ;

/*  <PARAMLESS PROCEDURE> ::= SUBROUTINE <IDENTIFIER> ;    */
  DO;ENTRY=STORE_CODE(0000);
     P=FIND_PROC(VAR(SP-1));  SFLAG=2;
     IF P=0 THEN P=SET_PROC(VAR(SP-1),2);
        ELSE DO; IF SHR(SHL(PTABLE(M),4),28) ¬=0 THEN
                    CALL ERROR('#PARAMETERS DOES NOT AGREE WITH PRIOR USE',2);
                 IF SHR(PTABLE(P),29) ¬= 2 THEN
                 CALL ERROR('PROCEDURE USED AS BOTH FUNCTION & SUBROUTINE',2);
              END;
    . PTABLE(P)=PTABLE(P)+ "10000000"; /* INDICATE PROCEDURE KNOWN */
     FIXV(MP)=SHL(P,8);  END;

/*  <PROCEDURE & PARAMETERS> ::= <PROCEDURE HEAD> <IDENTIFIER> ) ;    */
  DO;M=ENTER(VAR(SP-2),SET(VAR(SP-2)));
     PRT(M)=PRT(M) ! SHL(1,29);
     PRT(M)=PRT(M)|SHL(NEXT,8);
     IF NEXT-1 < PARMCELL THEN PARMCELL=NEXT-1;  NEXT=0;
     P=SHR(FIXV(MP),8); /* LOCATION OF PROCEDURE NAME */
     FIXV(MP)=FIXV(MP)+1;
     N=FIXV(MP) & "F";
     PTABLE(P)=PTABLE(P)+ "10000000"; /* INDICATE PROCEDURE KNOWN */
     IF (PTABLE(P) & "F000000")=0 THEN PTABLE(P)=PTABLE(P)|SHL(N,24);
       ELSE IF SHR(SHL(PTABLE(P),4),28) ¬=  (FIXV(MP) & "F") THEN
          CALL ERROR('PARAMETER COUNT DOES NOT AGREE WITH PRIOR USE',2); END;

/*  <PROCEDURE HEAD> ::= <PROCEDURE TYPE>    */
```

```
  ;

/*  <PROCEDURE HEAD> ::= <PROCEDURE HEAD> <IDENTIFIER> ,     */
  DO;M=ENTER(VAR(SP-1),SET(VAR(SP-1)));
     PRT(M)=PRT(M) | SHL(1,29);
     FIXV(MP)=FIXV(MP)+1;
     PRT(M)=PRT(M)|SHL(NEXT,8);
     NEXT=NEXT-1;  END;
/*  <PROCEDURE TYPE> ::= FUNCTION <IDENTIFIER> (     */
  DO;ENTRY=STORE_CODE(0000);  SFLAG=1;
     P=FIND_PROC(VAR(SP-1));
     IF P=0 THEN P=SET_PROC( VAR(SP-1) ,1);
       ELSE IF SHR(PTABLE(P),29) ¬=1 THEN
            CALL ERROR('PROCEDURE USED AS BOTH FUNCTION & SUBROUTINE',2);
     NEXT=PTABLE(P) & "FFF";
     FIXV(MP)=SHL(P,8) | SHL(1,4);
     M=ENTER1(VAR(SP-1),SP-1);
       /* SAVE OCTAL LOCATION OF VARIABLE FOR RETURN */
     ENTRY=ENTRY | SHL(PRT(M) & "FFF00",8);  END;

/*  <PROCEDURE TYPE> ::= SUBROUTINE <IDENTIFIER> (     */
  DO;ENTRY=STORE_CODE(0000);  SFLAG=2;
     P=FIND_PROC(VAR(SP-1));
     IF P=0 THEN P=SET_PROC( VAR(SP-1) ,2);
       ELSE IF SHR(PTABLE(P),29) ¬=2 THEN
            CALL ERROR('PROCEDURE USED AS BOTH FUNCTION & SUBROUTINE',2);
     NEXT=PTABLE(P) & "FFF";
     FIXV(MP)=SHL(P,8) | SHL(1,4);  END;

/*  <SUBROUTINE CALL> ::= <CALL> <VARIABLE>     */
  CFLAG=0;

/*  <CALL> ::= CALL     */
  CFLAG=1;

/*  <READ STATEMENT> ::= <READ HEAD> <VARIABLE> )     */
  DO;
  READ:
     M=LOOKUP(VAR(SP-1));
     IF M=0 THEN M=ENTER1(VAR(SP-1),SP-1);
     EMIT(2310); /* JMS I 6 */
     AFLAGCK;
     IF (LOC(SP-1) &"1")=1 THEN DO; EMITCK(6,SP-1); /* FPUT */
                                    EMIT(4096); /* FEXT */  AFLAG=0;
      /* CONVERT REAL TO INTEGER */  RETURN;  END;
     EMIT(0000); /* FEXT */     AFLAG=0;
     EMIT(2335); /* JMS I 37 */
     EMITCK(3,SP-1); /* DCA */  END;
```

```
   /*  <READ HEAD> ::= READ (     */
     ;

   /*  <READ HEAD> ::= <READ HEAD> <VARIABLE> ,     */
     GO TO READ;

   /*  <WRITE STATEMENT> ::= <WRITE HEAD> <EXPRESSION> )     */
     DO;
       WRITE_EXPRESSION:
         IF (LOC(SP-1) & "1")=1 THEN DO; AFLAGCK;
                                         EMITCK(5,SP-1); /* FGET */
                                         EMIT(0000); /* FEXT */  AFLAG=0;  END;
             ELSE DO; EMITCK(1,SP-1); /* TAD */
                      EMIT(2334); /* JMS I 36 */  END;
         EMIT(2309); /* JMS I 5 */  END;

   /*  <WRITE STATEMENT> ::= <WRITE HEAD> <STRING> )     */
     CALL EMIT_STRING(VAR(SP-1));

   /*  <WRITE STATEMENT> ::= <WRITE HEAD> <TAB EXPRESSION> )     */
     ;

   /*  <WRITE HEAD> ::= WRITE (     */
     DO;EMITCAR(141); /* RETURN */
        EMITCAR(138); /* LINE FEED */  END;

   /*  <WRITE HEAD> ::= WRITEON (     */
     ;

   /*  <WRITE HEAD> ::= <WRITE HEAD> <EXPRESSION> ,     */
     GO TO WRITE_EXPRESSION;

   /*  <WRITE HEAD> ::= <WRITE HEAD> <STRING> ,     */
     CALL EMIT_STRING(VAR(SP-1));

   /*  <WRITE HEAD> ::= <WRITE HEAD> <TAB EXPRESSION> ,     */
     ;

   /*  <TAB EXPRESSION> ::= TAB <EXPRESSION>     */
     DO;ADVANCE(2);
        EMIT(2332); /* JMS I 34 */
        EMIT(4096-FIXV(SP));  END;  END;
END SYNTHESIZE;

  /*                SYNTACTIC PARSING FUNCTIONS                          */
```

```
RIGHT_CONFLICT:
   PROCEDURE (LEFT) BIT(1);
      DECLARE LEFT FIXED;
      /*  THIS PROCEDURE IS TRUE IF TOKEN IS ¬ A LEGAL RIGHT CONTEXT OF LEFT*/
      RETURN ("C0" & SHL(BYTE(C1(LEFT), SHR(TOKEN,2)), SHL(TOKEN,1)
         & "06")) = 0;
   END RIGHT_CONFLICT;

RECOVER:
   PROCEDURE;
      /* IF THIS IS THE SECOND SUCCESSIVE CALL TO RECOVER, DISCARD ONE SYMBOL */
      IF ¬ FAILSOFT THEN CALL SCAN;
      FAILSOFT = FALSE;
      DO WHILE ¬ STOPIT(TOKEN);
         CALL SCAN;  /* TO FIND SOMETHING SOLID IN THE TEXT  */  END;
      DO WHILE RIGHT_CONFLICT (PARSE_STACK(SP));
         IF SP > 2 THEN SP = SP - 1;  /* AND IN THE STACK  */
         ELSE CALL SCAN;  /* BUT DON'T GO TOO FAR  */  END;
      OUTPUT = 'RESUME:' || SUBSTR(POINTER, TEXT_LIMIT-CP+MARGIN_CHOP+7);
   END RECOVER;

STACKING:
   PROCEDURE BIT(1);  /* STACKING DECISION FUNCTION */
      CALLCOUNT(1) = CALLCOUNT(1) + 1;
      DO FOREVER;    /* UNTIL RETURN  */
         DO CASE SHR(BYTE(C1(PARSE_STACK(SP)),SHR(TOKEN,2)),SHL(3-TOKEN,1)&6)&3;

            /*  CASE 0  */
            DO;    /* ILLEGAL SYMBOL PAIR  */
               CALL ERROR('ILLEGAL SYMBOL PAIR: ' || V(PARSE_STACK(SP)) || X1 ||
                  V(TOKEN), 1);
               CALL STACK_DUMP;
               CALL RECOVER;  END;

            /*  CASE 1  */
            RETURN TRUE;       /*  STACK TOKEN  */

            /*  CASE 2  */
            RETURN FALSE;      /* DON'T STACK IT YET  */

            /*  CASE 3  */
            DO;        /* MUST CHECK TRIPLES  */
               J = SHL(PARSE_STACK(SP-1), 16) + SHL(PARSE_STACK(SP), 8) + TOKEN;
               I = -1;  K = NC1TRIPLES + 1;  /* BINARY SEARCH OF TRIPLES  */
               DO WHILE I + 1 < K;
                  L = SHR(I+K, 1);
                  IF C1TRIPLES(L) > J THEN K = L;
                  ELSE IF C1TRIPLES(L) < J THEN I = L;
```

```
                    ELSE RETURN TRUE;  /* IT IS A VALID TRIPLE  */  END;
                RETURN FALSE;  END;
            END;     /* OF DO CASE  */
        END;    /*  OF DO FOREVER */
    END STACKING;

PR_OK:
    PROCEDURE(PRD) BIT(1);
        /* DECISION PROCEDURE FOR CONTEXT CHECK OF EQUAL OR IMBEDDED RIGHT PARTS*/
        DECLARE (H, I, J, PRD) FIXED;
        DO CASE CONTEXT_CASE(PRD);
            /*  CASE 0 -- NO CHECK REQUIRED  */
            RETURN TRUE;

            /*  CASE 1 -- RIGHT CONTEXT CHECK  */
            RETURN ¬ RIGHT_CONFLICT (HDTB(PRD));

            /*  CASE 2 -- LEFT CONTEXT CHECK  */
            DO;H = HDTB(PRD) - NT;
                I = PARSE_STACK(SP - PRLENGTH(PRD));
                DO J = LEFT_INDEX(H-1) TO LEFT_INDEX(H) - 1;
                    IF LEFT_CONTEXT(J) = I THEN RETURN TRUE;  END;
                RETURN FALSE;  END;

            /*  CASE 3 -- CHECK TRIPLES  */
            DO;H = HDTB(PRD) - NT;
                I = SHL(PARSE_STACK(SP - PRLENGTH(PRD)), 8) + TOKEN;
                DO J = TRIPLE_INDEX(H-1) TO TRIPLE_INDEX(H) - 1;
                    IF CONTEXT_TRIPLE(J) = I THEN RETURN TRUE;  END;
                RETURN FALSE;  END;
        END;  /* OF DO CASE  */
    END PR_OK;

  /*                          ANALYSIS ALGORITHM                                */

REDUCE:
    PROCEDURE;
        DECLARE (I, J, PRD) FIXED;
        /* PACK STACK TOP INTO ONE WORD */
        DO I = SP - 4 TO SP - 1;
            J = SHL(J, 8) + PARSE_STACK(I);  END;
        DO PRD = PR_INDEX(PARSE_STACK(SP)-1) TO PR_INDEX(PARSE_STACK(SP)) - 1;
            IF (PRMASK(PRLENGTH(PRD)) & J) = PRTB(PRD) THEN
                IF PR_OK(PRD) THEN
                DO;  7* AN ALLOWED REDUCTION */
                    MP = SP - PRLENGTH(PRD) + 1; MPP1 = MP + 1;
      IF CONTROL(BYTE('P')) THEN DO;
           S='                ' || V(HDTB(PRD)) || ' ::= ';
```

```
            DO I=MP TO SP;  S=S || V(PARSE_STACK(I)) || ' ';  END;
            OUTPUT=S;  END;
                CALL SYNTHESIZE(PRDTB(PRD));
                SP = MP;
                PARSE_STACK(SP) = HDTB(PRD);
                RETURN;  END;  END;
         /* LOOK UP HAS FAILED, ERROR CONDITION */
         CALL ERROR('NO PRODUCTION IS APPLICABLE',1);
         CALL STACK_DUMP;
         FAILSOFT = FALSE;
         CALL RECOVER;
      END REDUCE;

   COMPILATION_LOOP:
      PROCEDURE;
         COMPILING = TRUE;
         DO WHILE COMPILING;      /* ONCE AROUND FOR EACH PRODUCTION (REDUCTION)  */
            DO WHILE STACKING;
               SP = SP + 1;
               IF SP = STACKSIZE THEN
                  DO;CALL ERROR ('STACK OVERFLOW *** CHECKING ABORTED ***', 2);
                     RETURN;   /* THUS ABORTING CHECKING */  END;
               PARSE_STACK(SP) = TOKEN;
               VAR(SP) = BCD;
               IF NFLAG=1 THEN DO; FIXV(SP)=HOLD1; /* REAL NUMBER */
                                   FIXM(SP)=HOLD2|SHL(1,30);  END;
                          ELSE DO; FIXV(SP)=NUMBER_VALUE; /* INTEGER NUMBER */
                                   FIXM(SP)=0;  END;  NFLAG=0;
               CALL SCAN;  END;
            CALL REDUCE;
         END;     /* OF DO WHILE COMPILING  */
      END COMPILATION_LOOP;


   PRINT_SUMMARY:
      PROCEDURE;
         DECLARE I FIXED;
         CALL PRINT_DATE_AND_TIME ('END OF CHECKING ', DATE, TIME);
         OUTPUT = '';
         OUTPUT = CARD_COUNT || ' CARDS WERE CHECKED.';
         IF ERROR_COUNT = 0 THEN OUTPUT = 'NO ERRORS WERE DETECTED.';
         ELSE IF ERROR_COUNT > 1 THEN
            OUTPUT = ERROR_COUNT || ' ERRORS (' || SEVERE_ERRORS
               || ' SEVERE) WERE DETECTED.';
         ELSE IF SEVERE_ERRORS = 1 THEN OUTPUT = 'ONE SEVERE ERROR WAS DETECTED.';
            ELSE OUTPUT = 'ONE ERROR WAS DETECTED.';
         IF PREVIOUS_ERROR > 0 THEN
            OUTPUT = 'THE LAST DETECTED ERROR WAS ON LINE ' || PREVIOUS_ERROR
```

```
            || PERIOD;
        IF CONTROL(BYTE('D')) THEN CALL DUMPIT;
        DOUBLE_SPACE;
        CLOCK(3) = TIME;
        DO I = 1 TO 3;    /* WATCH OUT FOR MIDNIGHT */
           IF CLOCK(I) < CLOCK(I-1) THEN CLOCK(I) = CLOCK(I) +  8640000;  END;
        CALL PRINT_TIME ('TOTAL TIME IN CHECKER     ', CLOCK(3) - CLOCK(0));
        CALL PRINT_TIME ('SET UP TIME               ', CLOCK(1) - CLOCK(0));
        CALL PRINT_TIME ('ACTUAL CHECKING TIME      ', CLOCK(2) - CLOCK(1));
        CALL PRINT_TIME ('CLEAN-UP TIME AT END      ', CLOCK(3) - CLOCK(2));
        IF CLOCK(2) > CLOCK(1) THEN   /* WATCH OUT FOR CLOCK BEING OFF */
        OUTPUT = 'CHECKING RATE: ' || 6000*CARD_COUNT/(CLOCK(2)-CLOCK(1))
           || ' CARDS PER MINUTE.';
    END PRINT_SUMMARY;

MAIN_PROCEDURE:
    PROCEDURE;
        CLOCK(0) = TIME;  /* KEEP TRACK OF TIME IN EXECUTION */
        CALL INITIALIZATION;
        CLOCK(1) = TIME;
        CALL COMPILATION_LOOP;
        CLOCK(2) = TIME;
        /* CLOCK(3) GETS SET IN PRINT_SUMMARY */
        CALL PRINT_SUMMARY;
    END MAIN_PROCEDURE;

CALL MAIN_PROCEDURE;
RETURN SEVERE_ERRORS;

EOF EOF EOF
```