IBM 709/7090 DATA PROCESSING SYSTEMS BULLETIN

PRELIMINARY BULLETIN

FORTRAN ASSEMBLY PROGRAM (FAP) FOR THE IBM 709/7090 SUPPLEMENTARY INFORMATION FOR THE 32K VERSION

This bulletin was prepared by IBM Applied Programming as a supplement to the FORTRAN Assembly Program (FAP) for the IBM 709/7090, (Form J28-6098-1), and together with that publication, provides current specifications for FAP programs to be run on 32K 709 or 7090 Data Processing Systems, operating under either the FORTRAN or Basic Monitor (IBSYS).

Included herein are descriptions of new pseudo-operations, with sections on those connected with macro-operations and updating, and complete System Symbol and Combined Operations Tables. For additional information on FAP operating under the Basic Monitor (IBSYS), the reader is referred to the IBM preliminary bulletin, Basic Monitor (IBSYS), (Form J28-8086).

Coding is indicated throughout this bulletin by the use of all capital letters for actual coding; e.g., CLA.

Because of the preliminary nature of this bulletin, the material it contains, together with any necessary additions and corrections, will be published in a more complete form at a later date.

CONTENTS

THE FAP MACRO-OPERATION PROCESSOR

A FAP macro-operation is a type of pseudo-operation created by the programmer. The most significant property of an instruction specifying a macro-operation (a macro-instruction) is that it can generate one or more card images. The contents of the card images generated are virtually unrestricted and may include any machine operation, any pseudo-operation not restricted to the first card group (e.g., COUNT, ENTRY), or any macro-operation, and any field permitted on a FAP source card.

A macro-instruction can be regarded as an abbreviation for a sequence of card images. The sequence of card images generated by the macro-instruction is determined by the particular macro-definition corresponding to the macro-instruction code. Each macro-operation has its own definition, which consists of a heading card, a sequence of prototype card images, and an END card.

A prototype card image has a standard FAP source card format with location field, operation code, and variable field. Remarks may appear on a prototype card; however, the remarks will not normally appear following the variable field on the macro-generated card image. The fields of a prototype card may consist of: text, which will be reproduced as written; substitutable arguments; and (special) punctuation characters, which delimit arguments and, like text, are reproduced.

A field or subfield is text if it is longer than six characters or if it is a string of one through six characters, delimited by punctuation marks, which does not appear in the argument list of the macro-operation heading card. A field or subfield is a substitutable argument if it is a string of one through six characters which appears in the argument list of the macro-operation heading card.

A macro-generated sequence of card images consists of each of the prototype card images, with text and punctuation characters as on the prototype, but with substitutable arguments replaced by specific argument strings (of unrestricted length) whose position in the argument list of the macro-instruction corresponds to that in the argument list of the macro-definition heading card.

Examples of Programmer Macro-Operations

Suppose that the programmer has written a source program which will assemble, including the following sequences of instructions:

```
                                    ...
                                    ...
00127  0500 00 0 06365              CLA     FEDTAX
00130  0400 00 0 06367              ADD     STATAX
00131  0601 00 0 06370              STO     TOTTAX
                                    ...
                                    ...
00176  0500 00 0 03334  SUBCOM      CLA     XSUB1
00177  0400 00 0 03335              ADD     YSUB1
00200  0601 00 0 03336              STO     ZSUB1
                                    ...
                                    ...
01675  0500 00 0 05714              CLA     PART1
01677  0400 00 0 05715              ADD     PART2
01700  0601 00 0 05716              STO     TOTAL
                                    ...
                                    ...
```

The pattern of these three instructions might be designated by some BCD name, say QSUM, which could then be defined as follows:

```
                          QSUM  MACRO   V1,V2,V3
                                CLA     V1
                                ADD     V2
                                STO     V3
                          QSUM  END
```

The above sequence of five source cards generates no binary words in the object program, but constitutes the definition of the macro-operation which the programmer has chosen to call QSUM. The first card is the macro-definition heading card. It includes the name of the macro-operation in the location field and the argument list in the variable field. The next three cards are the prototype, in which V1, V2, V3, are substitutable arguments, identified in the heading card argument list. All of the other fields, CLA, ADD, STO, are text, since they do not appear in the argument list. The fifth card (END) marks the end of the range of the macro-definition. It will not terminate assembly.

Once the macro-operation QSUM has been defined in the source program, the sequence of CLA, ADD, STO, instructions may be replaced by a macro-instruction card which generates the sequence of instructions with their substitutable arguments replaced by the arguments in the variable field of the macro-instruction QSUM:

```
                                    ...
                                    ...
00127                               QSUM    FEDTAX,STATAX,TOTTAX
                                    ...
                                    ...
00176                     SUBCOM    QSUM    XSUB1,YSUB1,ZSUB1
                                    ...
                                    ...
01675                               QSUM    PART1,PART2,TOTAL
                                    ...
                                    ...
```

Note, in the above example, the following points:

1. The string QSUM which appears in the location field of the pseudo-operation MACRO is not a symbol, but a code for the macro-operation to be defined, and as such is entered into the Combined Operations Table. It may be the same as a location symbol appearing anywhere in the assembly, including symbols within the macro-definition.

2. The substitutable arguments V1, V2, V3, which appear in the variable field of the macro-definition heading card and also in various fields within the prototype, merely characterize the order of the expressions and character strings which may appear in the variable field of a later macro-instruction using the given macro-operation. If, on the macro-operation heading card, the order were to be changed to V3, V1, V2, then the macro-instruction

```
00127                          QSUM    TOTTAX,FEDTAX,STATAX
```

would cause generation of the same card images. Because the substitutable arguments are dummy names, they may be identical to strings used elsewhere in the program in location, operation, or variable fields, as symbols or operation codes, including the code for this or any other macro-operation. The programmer should exercise caution in constructing the prototype so the text will not be confused with substitutable arguments, as every string of six or fewer characters, in any field, is compared with the argument list. Special care should be taken with alphameric text, or with fields of VFD, DEC, or OCT pseudo-operations.

In the simple example of the macro-definition given above, the substitutable arguments appeared in address fields in the prototype and were replaced by symbols on the macro-generated cards. In general, substitutable arguments may appear in the location field, the operation field, in any of the subfields of the variable field, or as a heading character in any subfield. The substitutable arguments may be replaced by any valid FAP expression or appropriate alphameric character strings.

If a substitutable argument appears in an operation field, it may be a string of one through six characters; however, the code which replaces it must be a standard FAP operation code of three through six characters.

For example, the following definition could be written:

```
        QPOLY  MACRO   COEFF,LOOP,DEG,T,OP
               AXT     DEG,T
               LDQ     COEFF
        LOOP   FMP     GAMMA
               OP      COEFF+DEG+1,T
               XCA
               TIX     LOOP,T,1
        QPOLY  END
```

Here mnemonic character strings have been chosen to represent the substitutable arguments. Notice that LOOP appears in a location field, OP in an operation field, and that COEFF and DEG appear as symbols and within expressions in address subfields. Notice also that GAMMA is text - a symbol, and not a substitutable dummy argument - and presumably will be defined elsewhere in the program. Any use of the code QPOLY in a macro-instruction should be accompanied by an argument list of appropriate substitutions for the substitutable arguments. For example, LOOP should be replaced by a symbol, which should not be multiply defined, and OP should be replaced by a valid operation code.

A QPOLY macro-instruction might be written:

```
02031                  X015   QPOLY   C1-4,FIRST,5,4,FAD
```

The macro-instruction would cause the following six card images to be generated:

```
02031  0774 00 4 00005          AXT     5,4
02032  0560 00 0 06161          LDQ     C1-4
02033  0260 00 0 00135  FIRST    FMP     GAMMA
02034  0300 00 0 06167          FAD     C1-4+5+1,4
02035  0131 00 0 00000          XCA
02036  2 00001 4 02033          TIX     FIRST,4,1
```

The symbol X015 is defined as the location in which the first instruction (AXT) appears; each of the substitutable arguments is replaced by the corresponding argument in the macro-instruction argument list. The expression arising from the prototype address COEFF+DEG+1 is equivalent to C1+2.

Use of the macro-operation processor permits simulation of machine instructions of another computer, or extending the machine operation vocabulary of the 709/7090.

For example, STO can be modified to dump the information stored on each execution:

```
               0601 71 1 60000   .STO   OPSYN   STO
M                                STO    MACRO   A
                                        .STO    A
                                        SXA     *+2,4
                                        TSX     DUMP,4
                                        AXT     **,4
                                 STO    END
```

The M flag, appearing in the left margin of the assembly listing to point out a redefinition of an existing code, does not indicate an assembly error.

MACRO-DEFINING PSEUDO-OPERATIONS

MACRO

The pseudo-operation MACRO is used to name a macro-operation and to identify the arguments in the succeeding prototype. The constituents of the MACRO pseudo-instruction are:

1. Three to six BCD characters (not all zeros), appearing in the location field;
2. The operation code MACRO, appearing in the operation field; and
3. A list of substitutable dummy arguments, appearing in the variable field.

The Location Field

The character string in the location field is not a location symbol and will subsequently be used as a macro-operation code. If it is the same as any other machine operation, pseudo-operation, or macro-operation, the pseudo-operation will be flagged, the code will be redefined within the Combined Operations Table and the former definition lost.

The Argument List

The substitutable arguments in the macro-definition heading card argument list may be any legal FAP symbols, or may consist of all numeric characters (excluding all zeros). The substitutable arguments in a macro-definition may be separated by any of the following punctuation characters:

= + - * / ( ) $, '

and the argument list is terminated by the character "blank." After a punctuation character, suceeding punctuation characters or an explicit zero are ignored and do not result in a substitutable argument of zero.

Suggestive notation in a macro-definition argument list may be used. For example, in a macro-definition heading argument list

```
ALPHA MACRO  A(B+C)-D$E
```

is identical with

```
ALPHA MACRO  A,B,C,D,E
```

Extending the Argument List

The argument list of a macro-definition heading card may be extended by the use of the ETC pseudo-operation. In order that a following ETC card be recognized, it is necessary that the following conventions apply to the preceding card:

1. An unmatched left parenthesis exists in the variable field; or,

2. The variable field is terminated by a $ immediately followed by a blank or card column 73. This will not be confused with the use of this character to signal a heading character or a transfer vector name, as in neither of those cases will the $ be immediately followed by a blank; or

3. The variable field extends to card column 72.

If a card with an unmatched left parenthesis is not immediately followed by an ETC card, an assembly error will be flagged. If a card with a terminal $ is not followed by an ETC, the terminal $ is deleted from the macro-definition and ignored.

If the preceding card does not follow these conventions, an ETC card will be treated as the first card in the prototype, and an assembly error will usually result.

Alternative Form

An alternative form of the MACRO pseudo-instruction is the following:

1. Blanks, appearing in the location field;

2. The operation code MACRO, appearing in the operation field; and

3. Blanks, appearing in the variable field.

The above card is immediately followed by a card with these fields:

1. A FAP symbol, appearing in the location field;

2. Three to six BCD characters appearing in the operation field to be used as the macro-operation code; and

3. A list of substitutable arguments, appearing in the variable field.

A symbol, which may appear in the location field of the second card, is considered to be a substitutable argument, and not a location symbol. It is replaced by the corresponding argument in the location field of the macro-instruction card. This field in the macro-instruction is a specific argument, and not a location symbol. If it appears in the variable field of an instruction, it must be defined elsewhere in the program.

MOP

The pseudo-operation MOP is also used to define a macro-operation. The constituents of the MOP pseudo-instruction are:

1. Blanks, appearing in the location field;

2. The operation code MOP, appearing in the operation field; and

3. The macro-operation code, followed by the argument list, appearing in the variable field.

MOP is identical to MACRO except that the macro-operation code being defined appears as the first subfield in the variable field, followed by a punctuation character and the argument list.

The Prototype

The prototype follows the macro-definition heading card. It consists of a series of FAP instructions which use the substitutable arguments listed in the variable field of the preceding MACRO or MOP. The prototype must be followed by an END card with the macro-operation code in its location field or variable field, or with both the location field and the variable field blank.

To lend greater flexibility to macro-operations, parentheses and the apostrophe have been included in the list of special characters which may be used within the various fields of the macro-definitions. Thus, neither may be used as part of a substitutable argument.

Remarks cards with an asterisk in column 1, appearing within a macro-definition, will not appear in the expansions.

Heading characters in effect within the region in which the macro-definition appears do not apply to the definition.

The Location Field

A substitutable argument or any FAP symbol may appear in the location field of a prototype instruction.

The Operation Field

The following may appear in the operation field of a prototype instruction: any substitutable argument, any machine operation, any pseudo-operation not restricted to the first card group (e.g., COUNT, ENTRY),or any macro-operation code.

The Variable Field

A blank, if encountered before card column 72 on a prototype card (except the cards BCD, BCI, REM, TTL), is considered to terminate the variable field, and any information or commentary to the right of the blank will not be included in the macro-definition. If the blank appears following an unmatched left parenthesis, however, the blank does not terminate the scan. Blanks within parenthesis are considered to be text. If a matching right parenthesis is not encountered before card column 73, an ETC card must follow. Unmatched parentheses cause an assembly error.

Alphameric cards are scanned in full for substitutable arguments. If the variable field of BCD card (beginning in card column 12), or a BCI card (beginning in card column 12-16), commences with a non-blank, non-numeric character, the first subfield should be a substitutable argument for which a count will be substituted in the macro-instruction argument list.

The apostrophe (8-4 punch) may be used to concatenate (link) partial subfields in the operation field or in the variable field. It is possible to create a single subfield from a combination of arguments and text, as ' delimits an argument in the macro-definition prototype, but is itself not included in the macro-definition. The ' character cannot be used to concatenate subfields of lower level nested macro-definitions. For example, to delimit the count on a BCD prototype card, the following may be used:

```
ALPHA MACRO   A,B,C
      BCD A'0  'B' ERROR. CONDITION'C' IGNORED.
ALPHA END
```

The macro-instruction

```
02233                    ALPHA   7,FIELD,,
```

will cause the following card to be generated:

```
02233  006060263125        BCD 70  FIELD ERROR. CONDITION IGNORED.
```

The following illustrates macro-generation of an operation code by means of concatenation:

```
                    01226   J       TAPENO  A6B
                    02221   K       TAPENO  B1L

                            NAME4   MACRO   A,B,C,D,E,F,G
                            A       B'T'C'D E
                                    SD'F    G
                            NAME4   END

 03062
 03062  0766 00 0 01226    AX      NAME4   AX,W,D,J,,L,**
 03063  0776 00 0 00200            WTDJ
                                    SDL     **

 03064                              NAME4   AY,R,B,,5,N,K
 03064  0762 00 0 00225    AY      RTB     5
 03065  0776 00 0 02221            SDN     K
```

If parentheses are to be used as a part of text, the enclosed string must not appear in the macro-definition heading argument list, or the string will be considered a substitutable argument.

If a heading character of the form A$B is required, either A or B or both may be substitutable arguments.

If a transfer vector name is required of the form $NAME, NAME may be a substitutable argument.

If a literal of the form =A is required, A may be a substitutable argument for which a valid form of a literal must be substituted.

The variable field of any card in the prototype may be extended by the use of the ETC pseudo-operation. In order that a following ETC card be recognized, it is necessary to follow the conventions stated for extending a macro-definition heading card argument list (page 6). The programmer should exercise caution that a macro-generated card image which overflows column 72 has a variable field which is properly extended by an ETC card. Such variable fields are limited to those of a macro-instruction card, or a nested macro-definition heading card.

These conventions for ETC cards are different from those for ETC cards following CALL or VFD pseudo-instructions.

If a preceding card does not follow these conventions, an ETC card will be treated as is any other card in the prototype, and will be generated with arguments properly substituted. This ETC card may be used to extend the variable field of a CALL or VFD pseudo-instruction, provided it follows the conventions for CALL or VFD.

The macro-operation compiler will generate ETC cards, recognized by the macro-operation processor only, to follow any generated instruction whose variable field overflows card column 72.

Nesting Macro-Definitions

Macro-operation definitions may be nested by including a macro-definition heading card within a macro-definition prototype. The

various fields within the nested macro-operation will be scanned for substitutable arguments in the argument list of the outermost macro-definition. Those arguments common to both the outermost macro-definition and the nested macro-definition will be substitutable arguments for this inner macro-operation. Those intended to be substitutable arguments in the nested macro-operation, but not appearing in the outermost macro-definition argument list, will be considered as text within the outermost prototype. A macro-instruction using the outermost code will generate the nested macro-definition heading cards and prototypes of each of the macro-operations nested one level below, with all outer level arguments properly substituted. The name of the nested macro-operation may itself be an argument. This will cause the definition of a lower level macro-operation, the prototype of which will be scanned for substitutable arguments in the argument list of the current level definition, including those substituted in the outer expansion. Lower level macro-operations will not be defined until all higher level macro-operations within which they are nested have been expanded.

If macro-definitions are nested, the ends of the lower level prototypes must be marked with END cards bearing the name of the macro-operation in the location or the variable field. If no name appears in either field of the END card, the outermost macro-definition is terminated.

An example of nested macro-definitions is as follows:

```
NEST1 MACRO   A,B,C
NEST2 MACRO   A,D,E
NEST3 MACRO   B,D,F
      ...
      ...
NEST3 END
NEST2 END
NEST1 END
```

The prototype of a macro-definition may include macro-instructions, the macro-operations of which have not yet been defined; however, such lower level macro-operations must be defined prior to an appearance of the higher level macro-instruction. Circular definitions, which will result in a loop within the macro-operation processor, must be avoided by the programmer.

Restrictions on Macro-Definitions

The following restrictions apply to all macro-definitions:

1. When operating under the FORTRAN Monitor, all macro-definitions should be at the beginning of the program since the Error Records (which skip to END cards) may be confused by macro-definition END cards.

2. The effective limit of the number of macro-definitions or the total length of all macro-definitions is the length of tables which these share. Macro-operation codes are inserted in the Combined Operations Table, and the length of the Table must not exceed 1024 operation codes. Approximately 400 names are available to the programmer to insert macro-operation codes, or codes defined by OPD, OPSYN, or OPVFD pseudo-operations.

3. Macro-definitions share core storage with the symbol table; entries in one reduce the space available for the other. The space taken by macro-definitions is later occupied if required by the Symbolic Reference Table.

4. An argument list longer than 63 arguments will be truncated, but will not be flagged.

Macro-Instructions

A macro-instruction is used to generate, in line, the sequence of instructions given by the prototype, with substitutions for the arguments. The constituents of a macro-instruction are:

1. A FAP symbol, appearing in the location field;

2. A previously defined macro-operation, appearing in the operation field; and

3. A list of FAP symbols, expressions, alphameric character strings, or operation codes, appearing in the variable field.

The symbol in the location field of the macro-instruction will be defined as the location of the next instruction. A macro-instruction should not appear within the range of a DUP.

MAC

An additional pseudo-operation is available for use as a macro-instruction. The constituents of the MAC pseudo-instruction are:

1. A FAP symbol, appearing in the location field;

2. The operation code MAC, appearing in the operation field; and

3. The name of the macro-operation followed by the argument list appearing in the variable field.

MAC is identical to a macro-instruction with the code in the operation field, except that the code appears as the first subfield in the variable field, followed by a punctuation character and the argument list.

Punctuation

Only commas and parentheses may be used to separate arguments in the macro-instruction argument list. A single comma following a right parenthesis, or a single comma preceding a left parenthesis, is redundant and may be omitted. Consecutive commas define a null argument string; an explicit zero, if so desired, must appear in the argument list. A blank not within parentheses terminates the argument list. A pair of parentheses surrounding a string of characters in a macro-instruction argument string signifies that everything within the parentheses is to be substituted for the corresponding argument in the macro-definition prototype. Within such a pair of parentheses, nested parentheses, commas, and blanks are considered to be part of the string to be substituted. If a matching right parenthesis is not encountered before card column 73, an ETC card must follow. Unmatched parentheses cause an assembly error.

Parentheses to be included as part of a field to be substituted in a prototype must be enclosed within an outer pair of parentheses which will be deleted in the macro-instruction expansion.

For example, given the macro-definition

```
CALLIO MACRO   IOCOM,T1,OP,LABEL,T2,UNIT,PFX,ERRET
       TSX     (TAPE),4
       PZE     IOCOM,T1,OP
       PZE     LABEL,T2,UNIT
       IFF     0,ERRET
       PFX     ERRET
CALLIO END
```

the corresponding macro-instruction could be

```
03072                          CALLIO  CITIO,2,((RBEP)),CITLB,,CITTAP,,,
03072  0074 00 4 73406         TSX     (TAPE),4
03073  0 40004 2 06610         PZE     CITIO,2,(RBEP)
03074  0 00004 0 06614         PZE     CITLB,,CITTAP
                               IFF     0,
```

Note that TAPE should not be a substitutable argument; (RBEP) must be enclosed in an outer pair of parentheses; and that in the macro-instruction argument list, an explicit null argument bounded by a comma appears corresponding in position to the substitutable argument ERRET in the macro-definition argument list. This will cause the fourth word of the calling sequence to be omitted (see IFF on page 31).

As the character $ does not delimit an argument in the macro-instruction argument list, it may be used freely to indicate a heading character or a transfer vector symbol, to replace a substitutable argument. If the character $ is at the end of the argument list, a comma must be used to distinguish this from the character used to flag a following ETC card. As the character = does not delimit an argument in the macro-instruction argument list, it may be used freely to indicate a literal to replace a substitutable argument.

Argument Strings

The specific argument strings to be substituted must be given in the same order in the macro-instruction argument list as the substitutable arguments appear in the macro-definition heading argument list.

It is not necessary to restrict the length of an argument string to be substituted into a location field to six characters, or into an operation field to seven. Even an entire card image may be inserted into any field. No blank will be inserted following a location field longer than six characters, and the operation field, if any, will follow immediately.

Example:

```
                        NAME9 MACRO   XXX
                              XXX     REMARK
                        NAME9 END

04061                         NAME9   (CLA     B              )
04061  0500 00 0 06104         CLA     B               REMARK
```

Nested Macro-Instructions

It is possible to nest macro-instructions by including either a macro-operation code or a substitutable argument which will be replaced by a macro-operation, within an operation field in the prototype.

Example:

```
                        XXX   MACRO
                        XXX   END

                        COS   MACRO   OP
                              OP
                              TSX     $COS,4
                        COS   END

04155                         COS(COS(XXX))
04155                         COS(XXX)
                              XXX
04155  0074 00 4 00005        TSX     $COS,4
04156  0074 00 4 00005        TSX     $COS,4
```

Note that the null macro-operation XXX will cause the generation of no card images. The same effect could have been obtained by writing:

```
      -0 07774 0 04757   XXX   OPSYN   NULL
```

Note also that the assembler will assume a comma or an open parenthesis immediately following the operation code, as early as card column 11, to be the end of the operation field or the beginning of the variable field, respectively. Hence, suggestive notation, such as

```
                        COS(ALPHA)
```

may be used as a string to replace a substitutable argument.

The argument and sub-argument list of a macro-instruction may be extended by the use of the ETC pseudo-operation. In order that a following ETC card be recognized, it is necessary that the preceding card follow the conventions stated for extending a macro-definition heading card argument list (page 6). If the preceding card does not follow these conventions, an ETC card will be treated as the first card following the macro-generated card sequence, which usually results in an assembly error.

The Generated Cards

The generated cards are similar to the prototype cards, except that the substitutable arguments in the prototype will be replaced with the arguments appearing in the macro-instruction argument list. The heading characters in effect within the region in which the macro-instruction appears will be prefixed to all symbols shorter than six characters in the location and variable fields.

In the 709/7090 mode, the macro-compiler will generate the variable field beginning in card column 16, except for the cards BCD, BCI, REM, TTL. In the 704 mode and for the cards BCD, BCI, REM, TTL, the variable field will begin in card column 12. No note is taken of the card column in which the variable field begins on a prototype card. At least one blank will separate the variable field from an operation code which extends beyond these card columns.

Created Symbols

If arguments are missing from the end of the argument list of a macro-instruction, symbols will be created to fill the vacancies. These symbols take the form of ..001, ..002, to ..nnn, throughout the program. An explicitly null argument terminated by a comma will be treated as null; created symbols will be supplied only at the end of the argument string.

For example, given the macro-definition heading card

```
ALPHA MACRO   A,B,C,D
```

and the macro-instruction card

```
ALPHA   X,,
```

each appearance of the substitutable argument A will be replaced by X, each appearance of the substitutable argument B will be omitted, as the argument is explicitly void, and each appearance of the substitutable arguments C and D will be replaced by the symbols..nnn created to replace the omitted arguments at the end.

If more than 9999 symbols are to be created, the programmer must re-origin created symbols using the ORGCRS pseudo-operation, or assembly will be terminated.

NOCRS

The pseudo-operation NOCRS is used to suppress the creation of symbols to replace specific argument strings missing from the end of a macro-instruction argument list. The constituents of the NOCRS pseudo-instruction are:

1. Blanks, appearing in the location field;

2. The operation code NOCRS, appearing in the operation field; and

3. Blanks, appearing in the variable field.

ORGCRS

In order to alter the form of created symbols, the pseudo-operation ORGCRS may be used. This pseudo-operation also reinstates the creation of symbols, if they had been suppressed by NOCRS. The constituents of the ORGCRS pseudo-instruction are:

1. Blanks, appearing in the location field;

2. The operation code ORGCRS, appearing in the operation field; and

3. Blanks or one BCD character followed by three digits, appearing in the variable field.

The BCD character in the variable field, if any, will replace the second dot (.Annn); the digits, if any, will be the origin of a new set of created symbols. This origin will be one lower than the first symbol actually created. If the BCD character is desired, the three digits must be stated explicitly: Annn; but if it is not desired, nnn is sufficient.

Example:

```
MONSTR MACRO   DAVID,ALLEN,ROBERT,SCOTT
       CAL     DAVID
       TZE     SCOTT
       STA     ROBERT
       ALS     18
       ORA     ROBERT
 SCOTT SLW     ALLEN
       RMT
ROBERT BSS     1
       RMT
MONSTR END
```

In this macro-definition, the transfer address SCOTT and the storage address ROBERT must be unique for each appearance of the macro-operation in a macro-instruction. However, neither is required outside of the resulting expansion. Hence the assembler may be permitted to assign a location symbol, by omitting the corresponding arguments in the macro-instruction argument list. The pseudo-instruction ORGCRS is used to alter the format of the created symbol to .Nnnn:

```
                                        ORGCRS   N150

04170                                   MONSTR   MARLYN,HERB
04170 -0500 00 0 00312                  CAL      MARLYN
04171  0100 00 0 04175                  TZE      .N152
04172  0621 00 0 06000                  STA      .N151
04173  0767 00 0 00022                  ALS      18
04174 -0501 00 0 06000                  ORA      .N151
04175  0602 00 0 00313  .N152           SLW      HERB
```

The pseudo-operation BSS will be assembled later in the program (see RMT below).

```
06000                    .N151  BSS      1
```

IRP

The pseudo-operation IRP is used within a prototype to iterate a series of instructions within the set of generated instructions. The constituents of the IRP pseudo-instruction are:

1. Blanks appearing in the location field;

2. The operation code IRP, appearing in the operation field; and

3. A FAP symbol, appearing in the variable field.

The symbol in the variable field must be the name of a single substitutable argument appearing in the macro-definition argument list. An IRP card must precede the instructions to be iterated, and another IRP card with blank location and variable fields must follow the instructions. Both IRP cards must be within the range of the prototype.

The argument to be substituted (appearing in the macro-instruction argument list) is a string of sub-arguments separated by commas and enclosed in parentheses. The number of these sub-argument strings will be the number of iterations of the enclosed cards, and each iteration will be made with the corresponding sub-argument string substituted for the dummy argument. If no argument was given in the variable field of the first IRP, no iterations will be made; one argument causes one iteration, etc.

For example, to compute the sum of squares, the following macro-definition can be written:

```
SUMSQ MACRO   T,B
      STZ     T
      IRP     B
      LDQ     B          1)
      FMP     B          2)
      FAD     T          3)
      STO     T          4)
      IRP
SUMSQ END
```

The four instructions marked are to be iterated. To compute $A=X^2+Y^2+Z^2$, the following coding could be used:

```
04234                          SUMSQ   A,(X,Y,Z)
04234  0600 00 0 06016          STZ     A
                               IRP     X,Y,Z
04235  0560 00 0 06062          LDQ     X
04236  0260 00 0 06062          FMP     X
04237  0300 00 0 06016          FAD     A
04240  0601 00 0 06016          STO     A
04241  0560 00 0 06064          LDQ     Y
04242  0260 00 0 06064          FMP     Y
04243  0300 00 0 06016          FAD     A
04244  0601 00 0 06016          STO     A
04245  0560 00 0 06205          LDQ     Z
04246  0260 00 0 06205          FMP     Z
04247  0300 00 0 06016          FAD     A
04250  0601 00 0 06016          STO     A
```

An IRP pseudo-instruction cannot occur explicitly within the range of an IRP; the first nested IRP will cause the termination of the range, and the second, reopening of another range. However, a macro-instruction within the range of IRP pseudo-instructions may itself cause pairs of IRP pseudo-instructions to be generated at a lower level.

Note that the macro-operation compiler will not generate an ETC card for an IRP pseudo-instruction whose sub-argument string does not fit on one card, but will process the string internally.

The IRP pseudo-operation is undefined outside of a macro-operation.

RMT

The pseudo-operation RMT is used to bracket a block of coding which will not be assembled as encountered, but will be assembled when called for later in the coding. The constituents of the RMT pseudo-instruction are:

1. Blanks, appearing in the location field;

2. The operation code RMT, appearing in the operation field; and

3. Blanks, appearing in the variable field.

Macro-instructions may require assignment of temporary storage, definitions of constants, closed subroutines, or other allocations of memory. Such storage may be assigned within the macro-operation, in which case it must be bypassed by transfer instructions; or the programmer can keep track of the storage requirements

and define the necessary symbols whenever convenient. The pseudo-operation RMT provides a means by which such storage may be automatically assigned later in the assembly, at any point the programmer may specify.

A remote sequence is defined as all the source cards, or macro-generated cards, bracketed by a pair of RMT pseudo-instructions. Remarks cards with an * in column 1 appearing within a remote sequence will not appear in the expansion. A RMT pseudo-instruction cannot appear explicitly within the range of a remote sequence; the nested pair will cause the termination of the range, and the reopening of another range. However, a macro-instruction within the range of a remote sequence may itself cause pairs of RMT pseudo-instructions to be generated at a lower level. A remote sequence can be nested in a macro-operation, and macro-operation prototypes not including remote sequences may be nested to any desired depth within a remote sequence. When the remote sequence is assembled, any macro-definitions nested within will be defined and any macro-instructions nested within will be expanded.

Remote sequences may be defined outside of macro-operations, but should be used sparingly, as they may result in termination of assembly due to macro-definition table overflow. Little is gained by the use of this pseudo-operation outside of macro-operations.

Example:

The following remote sequence can be written:

```
        RMT
        CLA     *
XX      DEC     1E1
YY      DUP     1,2
        PZE
ZZ      BSS     10
        COS     XXX
        RMT
```

Subsequently, a RMT * pseudo-instruction will cause all waiting remote sequences to be assembled. The constituents of a RMT * pseudo-instruction are:

1. Blanks, appearing in the location field;

2. The operation code RMT, appearing in the operation field; and

3. An asterisk, appearing in the variable field.

RMT * and macro-instructions may be nested; however, the depth of nesting is limited as follows: each macro-instruction

within a remote sequence, or conversely, each RMT * generated by a macro-instruction, requires three locations in the Level Table. The total length of this Table is 112 locations, and the most severe use of this Table, an alternate nesting of macro-instructions and RMT *, will terminate assembly if there are more than 37 such nestings.

For example, if the remote sequence above had been defined, the following would be assembled following a RMT *:

```
04306                                  RMT     *
04306  0500 00 0 04306                  CLA     *
04307 +205400000003            XX      DEC     1E1
                    04310      YY      DUP     1,2
04310  0 00000 0 00000                  PZE
04311  0 00000 0 00000
04312                          ZZ      BSS     10
04324                                  COS     XXX
                                       XXX
04324  0074 00 4 00005                 TSX     $COS,4
```

If any remote sequences are waiting at the end of an assembly, they will be assembled following the program END card. As remote sequences may include macro-definitions, and macro-instructions may include remote sequences, a significant amount of coding may follow the END card. This generated coding will precede any literals.

Heading characters for remote sequences are those in effect at the time of definition, and not at the time of assembly. A remote sequence defined or assembled in a multiply-headed region may be improperly headed, and caution should be exercised.

UPDATING SYMBOLIC FAP TAPES

FAP permits reading an input tape other than Logical Tape 5 (the System Input Tape) and writing an updated blocked or unblocked symbolic tape, with optional deletion of assembly. This updating facility has been included in Pass 1 of FAP.

Update pseudo-instructions are listed in the pre-processor update listing, but will normally not appear on an update output tape or in the assembly listing. However, if the variable field of the pseudo-instruction is in error and is flagged, it will appear in the assembly listing and will be copied onto the update output tape.

UPDATE

In the FAP language, the UPDATE pseudo-operation is used to initiate the updating mode, assign update input and output tapes, and determine the mode of assembly. The constituents of the UPDATE pseudo-operation are:

1. Blanks, appearing in the location field;

2. The operation code UPDATE, appearing in the operation field; and

3. Two symbolic expressions and two single characters, all separated by commas, appearing in the variable field.

The first expression in the variable field is a logical tape number, which may be void or zero (no tape provided), or the number of the tape containing the symbolic input to be updated (Monitor tapes 1 through 8 are excluded). Successive use of the UPDATE pseudo-operation will permit multi-reel input.

The second expression in the variable field is the logical tape number of the tape on which an updated symbolic input for a future assembly will be written. Unflagged update pseudo-operations will not be copied onto this tape. Successive use of UPDATE will permit subdividing of, or extracting from, an input tape.

The first single character in the variable field is any non-void, non-zero character, which indicates that the updated symbolic tape is to be unblocked. This will result in a significant increase in assembly time. If the character is void or zero, the updated symbolic tape will be blocked 14 words per card, 16 cards per block. Control cards and END cards are always unblocked.

The second character is any non-void, non-zero character which indicates that assembly is not required. In effect, this character

reduces the FAP processor to an updating and/or blocking routine. No table entries are made, and Pass 2 is omitted. The only cards recognized are update pseudo-operations (and END), and those with an asterisk in column 1. The second character must appear within the first card group, and having once appeared, neither it nor its preceding comma may reappear on another UPDATE card.

Cards with an * in card column 1 are considered to be control cards and are unblocked if they follow another such card, or any update pseudo-operation or END. They are considered as comments cards and are blocked if they follow any card other than an update pseudo-operation card or another *-type control card.

Note that corrections and additions are made on the basis of serialization in card columns 73 through 80. A matching card on the Source Input Tape will replace the card on the update input tape; a nonmatching card will be inserted in sequence. A blank is sequenced following the character * and preceding the character /. If card columns 75 through 80 on either the source or update input tapes are all blank, the serialization is taken to be 00000000 for sequencing, and the card is used immediately. All cards inserted or replaced are labeled as such, and appear on the preprocessor assembly listing.

Example:

```
UPDATE  11
```

This merges correction cards appearing on the Source Input Tape with those appearing on Logical Tape 11, and reassembles. The Update Input Tape is on the same channel as the Source Input Tape; this is the fastest combination of tape assignments for a two-channel installation.

Example:

```
UPDATE 11,12
```

This merges correction cards appearing on the Source Input Tape with those appearing on Logical Tape 11, writes a blocked symbolic output tape on Logical Tape 12, and assembles.

Example:

```
UPDATE  ,12,,D
```

This copies the succeeding cards which are on the Source Input Tape, and blocks them onto Logical Tape 12.

Example:

```
UPDATE  11,,,D
```

This effectively spaces the tape and checks the sequence of the cards on Logical Tape 11.

NUMBER

The NUMBER pseudo-operation is used to reserialize columns 73-80 of the symbolic cards which are output on the Listing Tape and/or Update Output Tape.

The constituents of the NUMBER pseudo-instruction are:

1. Up to six BCD characters appearing in the location field;

2. The operation code NUMBER, appearing in the operation field; and

3. A number less than 32768, appearing in the variable field.

The six BCD characters will be left justified in card columns 73-78 with blanks omitted, and the number will be right justified in card columns 76-80. The programmer must insure that the BCD and numeric fields do not overlap for the length of the program. If the variable field is omitted, reserialization is suspended, and old serials (if any) appearing in card columns 73-80 will be maintained. In order to reserialize from 0, an explicit zero must appear in the variable field, in addition to any characters which may appear in the location field.

DELETE

The DELETE pseudo-operation will cause the deletion of one or more cards, of, or up through, a card of matching serialization. If matching serialization does not exist, deletions will be made up to, but not including, the next card of logically higher serialization. The constituents of the DELETE pseudo-instruction are:

1. Blanks appearing in the location field;

2. The operation code DELETE, appearing in the operation field;

3. THRU, when required, appearing in the variable field; and

4. A serialization number of eight characters, appearing in card columns 73-80.

Deleted cards appear, labeled as deleted, on the pre-processor listing.

IGNORE

IGNORE is identical with DELETE except that the cards deleted will not appear on the pre-processor listing. IGNORE has two forms: IGNORE and IGNORE THRU, which are similar to DELETE and DELETE THRU.

SKIPTO

The SKIPTO pseudo-operation will cause the deletion of one or more cards up to, but not including, a card of matching serialization.

The constituents of the SKIPTO pseudo-instruction are:

1. Blanks, appearing in the location field;

2. The operation code SKIPTO, appearing in the operation field;

3. Blanks, appearing in the variable field; and

4. A serialization number of eight characters, appearing in card columns 73-80.

Cards deleted will not appear on the pre-processor listing. Cards of higher serialization will not cause this operation to terminate.

ENDFIL

The ENDFIL pseudo-operation is used during an update, with or without assembly, to cause an end-of-file to be written on the update tape so addressed.

The constituents of the ENDFIL pseudo-instruction are:

1. Blanks, appearing in the location field;

2. The operation code ENDFIL, appearing in the operation field; and

3. A FAP expression or logical tape number appearing in the variable field.

If the logical tape number in the variable field is that of the current update output tape, so specified by an UPDATE pseudo-

instruction, the last partial block of card images waiting to be output will be written on the update output tape before the end-of-file is written. If the variable field is blank, the update output tape is assumed.

REWIND

The REWIND pseudo-operation is used during an update with or without assembly to cause the addressed update tape to be rewound.

The constituents of the REWIND pseudo-instruction are:

1. Blanks, appearing in the location field;

2. The operation code REWIND, appearing in the operation field; and

3. A FAP expression or logical tape number, appearing in the variable field.

If the logical tape number in the variable field is that of the current update output tape, specified by an UPDATE pseudo-instruction, the last partial block of card images waiting to be output will be written on the update output tape before the tape is rewound. If the variable field is blank, the update output tape is assumed. If the update input or output tape is rewound, no update operation pertaining to it will be executed unless it is addressed by a subsequent UPDATE pseudo-instruction.

UNLOAD

The UNLOAD pseudo-operation is used during an update with or without assembly, to cause the addressed update tape to be rewound and unloaded.

The constituents of the UNLOAD pseudo-instruction are:

1. Blanks, appearing in the location field;

2. The operation code UNLOAD appearing in the operation field; and

3. A FAP expression or logical tape number, appearing in the variable field.

If the logical tape number in the variable field is the current update output tape, specified by an UPDATE pseudo-instruction, the last partial block of card images waiting to be output will be written on the update output tape before the tape is rewound and unloaded. If the variable field is blank, the update output tape is assumed. If the update input or output tape is rewound and un-

loaded, no update operation will be performed on it unless it is addressed by a subsequent UPDATE pseudo-instruction. In the latter case, the operator should be informed by a PRINT pseudo-instruction to ready the tape sufficiently in advance to avoid delaying the assembler.

SKPFIL

The SKPFIL pseudo-operation is used during an update with or without assembly to cause the update tape which it addresses to be spaced forward until an end-of-file is passed.

The constituents of the SKPFIL pseudo-instruction are:

1. Blanks, appearing in the location field;

2. The operation code SKPFIL, appearing inthe operation field; and

3. A FAP expression or logical tape number, appearing in the variable field.

If the logical tape number in the variable field is that of the current update output tape, as specified by an UPDATE pseudo-instruction, the last partial block of card images waiting to be written will be placed on the update output tape before the tape is spaced. If the variable field is blank, the update input tape is assumed.

UMC

The UMC pseudo-operation is used to output card images generated by a macro-instruction onto a symbolic update output tape and to delete all macro-definition and macro-instruction cards. A subsequent appearance of UMC will cause a reversion to the normal output of macro-definition and macro-instruction cards; alternate appearances cause alternate switching from writing to not writing macro-generated images.

The constituents of the UMC pseudo-instruction are:

1. Blanks, appearing in the location field;

2. The operation code UMC, appearing in the operation field, and

3. Blanks, appearing in the variable field.

The programmer should exercise caution in the use of this pseudo-operation, in that a macro-definition may be deleted, but a macro-instruction for the same code may remain. As the remaining remote

sequences will be expanded after the END card is written, they must be inserted before the END card by the use of the RMT * pseudo-operation. The serial numbers associated with deleted macro-definition cards will be deleted; all cards generated by the macro-instruction will be serialized with the serial number associated with the macro-instruction card. Access to any of these is difficult. An IRP pseudo-operation in an expression is never copied on the update output tape.

ENDUP

The ENDUP pseudo-operation signals the termination of a FAP update with assembly deleted. The constituents of the ENDUP pseudo-operation are:

1. Blanks, appearing in the location field;

2. The operation code ENDUP, appearing in the operation field;

3. Blanks, appearing in the variable field; and

4. A serialization of eight characters, appearing in card columns 73-80.

If assembly is deleted, END cards will be unblocked, but will not terminate the pass. This makes it possible to update more than one program at a time. If the assembly is not deleted, ENDUP will be undefined. A card on the update input tape of matching serialization will be deleted, and no card of such serialization will appear on the updated output tape. Blank serialization on the ENDUP card will cause immediate termination of the update.

Tape Positioning

Because matching serialization is not required for any control card, it will be necessary, for proper spacing of tapes, to have symbolic cards of matching serialization on the Source Input Tape to position the update input tape past those cards of logically higher serialization which precede the card which is next to be updated (such cards are referred to as "spacers" below). It is also necessary to have either an END card of matching serialization on the source input tape or an ENDUP card (for an update without assembly) which is properly serialized, or without serialization, immediately following a properly serialized END card on the Source Input Tape. This insures that the update tape is properly positioned at the termination of FAP Pass 1. It should be noted that if a multi-reel update job has been terminated during Pass 1 due to source tape or machine error, update input and output tapes may be positioned incorrectly. Caution should be exercised in restarting or continuing. For a single input and single output tape, the Error Records will attempt to reposition these tapes for another try.

In order to update a FORTRAN source deck, the output tape must be unblocked and the assembly deleted. ENDUP is the pseudo-operation to terminate the job.

In case of illegible cards on the Source Input Tape, the assembly is terminated and the Machine Error Record will skip to the end of the job, to the END card, or to the beginning of the assembly, at the operator's option. If a card (or block of cards) on the update input tape is illegible, the card (or cards) will be omitted, the next source correction card read, a message printed on- and off-line, and the job will continue. It is possible to reinsert lost cards during a later update. The omission of such cards will usually result in an assembly error, but the update input tape will be properly spaced for the next assembly, unless an END card or a "spacer" card is read erroneously.

Update Examples

The following sequence of cards would cause an end-of-file, an END TAPE card, and another end-of-file to be written on the updated Symbolic Tape 12, this tape to be rewound, and Logical Tape 11 to be unloaded:

```
*       FAP
        UPDATE  ,12,,D
        ENDFIL
*       END TAPE
        ENDFIL
        REWIND
        UNLOAD  11
        ENDUP
```

The following cards could surround an input deck on the Source Input Tape to obtain a blocked, serialized Tape 12 for a later assembly:

```
 *      DATE    1/2/62
 *      JOB IDENTIFICATION
 *      PACK
 *      FAP
 *      PAGE TITLE CARD
        UPDATE  ,12
  F01   NUMBER  0
 *      DATE    1/2/62
 *      JOB IDENTIFICATION
 *      PACK
 *      FAP
 *      PAGE TITLE CARD
        COUNT   5000
        ABS
        ...
        ...
        ...
        END
 *      FAP
        UPDATE  ,12
        ENDFIL
 *      END TAPE
        ENDFIL
        UNLOAD
        END
   (END OF FILE)
 *      END TAPE
   (END OF FILE)
```

The following cards on the Source Input Tape might be used to update and reserialize symbolic cards on Logical Tape 9, with assembly deleted and with the updated output on Logical Tape 10:

```
*       FAP
        UPDATE  9,10,,D
F01     NUMBER  0
        CAL     ERASE                                   F0101010
        DELETE                                          F0101200
        NZT     SYMBL                                   F0101461
        TRA     ERROR
F02     NUMBER  0                                       F0199999
        DELETE                                          F0201730
        DELETE  THRU                                    F0201840
        DELETE                                          F0202040
        IGNORE  THRU                                    F0204170
F03     NUMBER  0                                       F0299999
        END                                             F0301760
        ENDUP
```

ADDITIONAL SYMBOL DEFINING PSEUDO-OPERATION

SET

In order to define a symbol, permitting it to be redefined later, the pseudo-operation SET may be used. The constituents of SET are:

1. A FAP symbol, appearing in the location field;

2. The operation code SET, appearing in the operation field; and

3. A FAP expression, appearing in the variable field.

The symbol in the location field is defined to have the value of the FAP expression in the variable field. This expression must satisfy all the rules of writing expressions. If the symbol had previously been defined by the SET pseudo-operation, it will be redefined. If the symbol had previously been defined, but not by a SET pseudo-operation, the symbol will be redefined, but an error flag will appear in the left margin of the listing. This is not an assembly error. The SET pseudo-operation will override any prior means of definition. This pseudo-operation is useful in providing a value for a parameter used within a macro-operation, which is not accessible through the argument list.

ADDITIONAL PROGRAM LINKING PSEUDO-OPERATION

EXTERN

The EXTERN pseudo-operation is used to insert symbols in the transfer vector.

The constituents of the EXTERN pseudo-instruction are:

1. Blanks, appearing in the location field;

2. The operation code EXTERN, appearing in the operation field; and

3. A list of FAP symbols, of one through six BCD characters, separated by commas, appearing in the variable field.

A symbol appearing in an EXTERN list, if already in the transfer vector, will be ignored. If a symbol has been previously defined in the location field of a machine instruction or a symbol defining pseudo-operation, it will be multiply defined. EXTERN is undefined in an absolute assembly.

ADDITIONAL LIST CONTROL PSEUDO-OPERATIONS

INDEX

The INDEX pseudo-operation is used to list a table of contents of important locations within the assembly.

The constituents of the INDEX pseudo-instruction are:

1. Blanks, appearing in the location field;

2. The operation code INDEX, appearing in the operation field; and

3. A list of FAP symbols, of one through six BCD characters, separated by commas, appearing in the variable field.

The first appearance of an INDEX card will cause the message

TABLE OF CONTENTS

to be listed. Each subfield of an INDEX pseudo-instruction will cause the symbol, and its definition, to be listed. The listing of the INDEX card itself is governed by the mode of the PCC control card.

INDEX pseudo-instructions may appear anywhere in the source program, and need not be grouped. The listing generated by INDEX pseudo-instructions will be inserted where the pseudo-instructions appear. However, the message TABLE OF CONTENTS will appear but once; and, for the most meaningful commentary, INDEX pseudo-instructions should be grouped at the beginning of the source program, interspersed with appropriate remarks cards.

PMC

Alternate appearances of PMC pseudo-operations with blanks in the variable field cause and suppress listing; ON or OFF give absolute control.

The constituents of the PMC pseudo-instruction are:

1. Blanks appearing in the location field;

2. The operation code PMC, appearing in the operation field; and

3. Blanks, or ON, or OFF, appearing in the variable field.

Card images generated through the use of a macro-instruction are normally not listed, except for card images which are flagged by the assembler. In order to cause such card images and the octal instruction to be listed, the pseudo-operation PMC can be used.

Control cards generated by macro-instructions are listed only if both PMC and PCC modes are ON. The nesting level of macro-generated cards appears in card columns 81 through 84 of the listing.

NULL

The NULL pseudo-instruction is used to cause a card image to be listed in full, but to have no effect upon assembly.

The constituents of the NULL pseudo-instruction are:

1. Any BCD characters appearing in the location field;

2. The operation code NULL, appearing in the operation field; and

3. Any BCD characters, appearing in the variable field.

A possible use of the NULL pseudo-instruction is

```
-0 07774 0 04757   CODE  OPSYN   NULL
```

where CODE is a machine operation or pseudo-operation which is not defined in the assembler, the effect of which may be omitted from the assembly. For example,

```
-0 07774 0 04757   ENTRY OPSYN   NULL
```

will enable a proper absolute assembly of a subprogram, with ENTRY pseudo-instructions listed, but not affecting the program.

ADDITIONAL CONDITIONAL ASSEMBLY PSEUDO-OPERATION

IFF

The IFF pseudo-operation controls the assembly of the following card.

The constituents of the IFF pseudo-instruction are:

1. Blanks appearing in the location field;

2. IFF appearing in the operation field; and

3. A series of FAP symbols separated by commas appearing in the variable field.

The pseudo-operation IFF with P,A,B in the variable field provides conditional assembly of program segments according to the values of the parameters P,A,B. P is a FAP expression, and A and B are BCD symbols. The pseudo-operation results in assembly of the next instruction (and all ETC cards), only if

1. P is not 0 and A is identical to B.

2. P = 0 and A not identical to B.

P will be zero if it is undefined; P will be considered non-zero if it is relocatable. That the conditional assembly governed by IFF extends over only one card (and all ETC cards) is not a serious restriction, as the following card may be a macro-instruction which will expand to a sequence of any length, or it may be another IFF. Remarks cards with an * in column 1 following an IFF will be ignored; the instruction immediately following a block of such cards will be conditionally assembled.

The example of the CALLIO macro-operation (page 12) can be used to demonstrate IFF. As P=0, the next prototype card

```
PFX     ERRET
```

will be generated only if field A, replacing the substitutable argument ERRET, is not identical with field B, which is void. As in the macro-instruction argument list, field A is also a void field; it is identical with field B, and the next prototype card will not be generated.

An IFF pseudo-operation, and all cards under its control, will be copied on an update output tape.

For example, the following macro-definition can be written:

```
                                ADD3   MACRO   A,B,C
                                       CLA     A
                                       ADD     B
                                       IFF     0,C,AC
                                       STO     C
                                ADD3   END
```

If it is desired to store the result, the following macro-instruction can be written:

```
04552                          ADD3    X,Y,Z
04552  0500 00 0 06062         CLA     X
04553  0400 00 0 06064         ADD     Y
                               IFF     0,Z,AC
04554  0601 00 0 06205         STO     Z
```

If it is desired to leave the result in the Accumulator, the following macro-instruction can be written:

```
04622                          ADD3    X,Y,AC
04622  0500 00 0 06062         CLA     X
04623  0400 00 0 06064         ADD     Y
                               IFF     0,AC,AC
```

REVISED PSEUDO-OPERATIONS

The following changes have been made in existing pseudo-operations:

CALL

A CALL pseudo-instruction may be used in an absolute assembly. No transfer vector entry will be made, and the name of the subprogram in the first subfield of the variable field must be defined as any other symbol.

END

A symbol in the location field of an END pseudo-instruction will be defined as the last location used by the program, one location below the program break.

ENTRY

An explicit zero in the variable field of an ENTRY pseudo-instruction will cause the program card to indicate that the entry point to the main program is the first instruction following the transfer vector and linkage director.

LBL

If the variable field of a LBL pseudo-instruction is blank, serialization of binary cards will be discontinued. In order to serialize from 0, an explicit zero must appear in the variable field.

LOC
ORG

A LOC or ORG pseudo-instruction may be used in a relocatable assembly. If the expression in a variable field is COMMON or undefined, an assembly error will result. If the expression in the variable field is absolute or relocatable, the new origin, and any symbol in the location field of a LOC or ORG card, will always be assumed to be relocatable above the transfer vector and linkage director.

PCC

A PCC pseudo-instruction may now have ON or OFF as well as blanks in the variable field; the new forms give absolute switch control.

PRINT

PRINT is considered to be an update pseudo-operation, and will be listed in the pre-processor update listing. It will not appear on the assembly listing or the update output tape.

TAPENO

Any one of the following characters may be used to set the mode in a TAPENO pseudo-instruction:

B	Binary
D	Decimal
H	High density
L	Low density

ADDITIONAL REVISIONS

The locations of definitions of symbols defined by the following pseudo-operations will appear in the symbolic reference table: BOOL, COMMON, EQU, MAX, MIN, SYN, and TAPENO.

In addition, if a symbol is multiply defined by its appearance in the location field of more than one of the following operations, the

point of definition will be flagged in the left hand margin of the listing: BCD, BCI, BES, BOOL, BSS, CALL, COMMON, DEC, DUP EQU, IFEOF, MAX, MIN, OCT, SYN, TAPENO, VFD, or any machine instruction. This flag does not signify an assembly error. However, any reference to a multiply defined symbol in the variable field of a machine operation or pseudo-operation will be flagged as before, signifying an assembly error. It is possible to have two M flags in the left-hand margin opposite a single instruction: one referring to the point of definition (location field) and the other to the point of use (variable field).

APPENDIX I: COMBINED OPERATIONS TABLE

Pseudo-Operations

The following is a list of pseudo-operations in the Combined Operations Table. Those marked with @ have been added or extended, and are described in this bulletin; all others are described in detail in FORTRAN Assembly Program (FAP) for the IBM 709/7090 (Form J28-6098-1).

	Operation Code	Purpose
	704	Set mode of assembly
	7090	Set mode of assembly
	9LP	Set card format
	ABS	Set card format
	BCD	Data generating
	BCI	Data generating
	BES	Storage allocating
	BOOL	Symbol defining
	BSS	Storage allocating
@	CALL	Program linking
	COMMON	Storage allocating
	COUNT	Assembler information
	DEC	Data generating
@	DELETE	Update information *
	DETAIL	List control
	DUP	Data generating
	EJECT	List control
@	END	Assembler information
@	ENDFIL	Update information
@	ENDUP	Update information
	ENTRY	Program linking
	EQU	Symbol defining
	ETC	Continue variable field *
@	EXTERN	Program linking
	FUL	Set card format
	HEAD	Symbol defining
	HED	Symbol defining
	IFEOF	Program linking
@	IFF	Conditional assembly
@	IGNORE	Update information *
@	INDEX	List control
@	IRP	Macro-operation control
@	LBL	Binary card labelling
	LIST	List control
@	LOC	Storage allocating
@	MAC	Macro-instruction

	Operation Code	Purpose
@	MACRO	Macro-definition
	MAX	Symbol defining
	MIN	Symbol defining
@	MOP	Macro-definition
@	NOCRS	Macro-operation control
@	NULL	List control
@	NUMBER	Update information
	OCT	Data generating
	OPD	Opcode defining
	OPSYN	Opcode defining
	OPVFD	Opcode defining
@	ORG	Storage allocating
@	ORGCRS	Macro-operation control
@	PCC	List control
@	PMC	List control
@	PRINT	Update information
	REF	List control
	REM	List control
@	REWIND	Update information
@	RMT	Deferred assembly
@	SET	Symbol defining
@	SKIPTO	Update information *
	SKP	List control
@	SKPFIL	Update information
	SPACE	List control
	SPC	List control
	SST	Symbol defining
	SYN	Symbol defining
@	TAPENO	Symbol defining
	TCD	Set card format
	TITLE	List control
	TTL	List control
@	UMC	Update information
	UNLIST	List control
@	UNLOAD	Update information
@	UPDATE	Update information
	VFD	Data generating

The following pseudo-operations have been deleted:

ENDFUL	(use ABS)
FAP	(use COUNT; not to be confused with *FAP Monitor Control Card)

* Pseudo-operations which do not appear in the Combined Operations table.

Machine and Extended Machine Instructions

The following is a list of machine instructions and extended machine instructions, type, and permissible fields in the Combined Operations Table. Each is described in detail in the 7090 Reference Manual (Form A22-6568) and/or in the 704 Reference Manual (Form A22-6500 1). Those marked @ have been added or extended.

In the machine instruction list, the description of the machine instruction, the fields permitted and/or machine required, the mode of assembly, and other information is coded as follows:

INSTRUCTIONS

TYPE	INSTRUCTION FORMAT	USAGE
A	0 00000 0 00000	18 bit decrement field
B	0000 xx 0 00000	No decrement field
C	0000 00 0 00000	8 bit decrement field
D	0000 xx 000000	18 bit address field
E	0760 xx 0 x0000	13 bit address field
K	00 0000 0 00000	4 bit prefix field

OTHER CODES

CODE	INTERPRETATION
N	Not significant, or may change operation
P	Permissible
R	Required
T	Operation code defined or required address satisfied by use of TAPENO character
(I/O)	I/O unit address defined by operation mnemonic
4	Permissible in 704 mode only
9	Permissible in 7090 mode only
8	Decrement field not longer than 8 bits required

An "X" as the fourth character of an operation code indicates a variable channel operation, for which character the channel designation A to H, or a properly defined TAPENO character, must be substituted. Unit record equipment should not use this character.

Violation of these rules will be flagged in the left hand margin of the listing, but this flag will not indicate an assembly error with deletion of relocatable binary cards. No 709 mode exists; hence while assembling in the 7090 mode, drum instructions will be flagged while indirect addressing of I/O commands will not.

Operation codes which are on a "Request Price Quotation" basis are indicated by RPQ in the mode column.

Op Code	Type	Addr	Tag	Dec	Ind Addr	Mode
ACL	B	R	P		P	
ADD	B	R	P		P	
ADM	B	R	P		P	
ALS	B	R	P			
ANA	B	R	P		P	
ANS	B	R	P		P	
ARS	B	R	P			
AXC	B	R	R			9
AXT	B	R	R			9
@BRA	A	R	(2)	(2)		
@BRN	A	R	(2)	(2)		
BSF	B(I/O)	R	P			9
BSFX	B(I/O)	RT	P			9
BSR	B(I/O)	R	P			9
BSRX	B(I/O)	RT	P			9
BST	B(I/O)	R	P			4
BTT	E	R	P			9
BTTX	E	NT	N			9
CAD	B	R	P			4
CAL	B	R	P		P	
CAQ	C	R	P	8		9
CAS	B	R	P		P	
CFF	E	N	N			
CHS	E	N	N			
CLA	B	R	P		P	
CLM	E	N	N			
CLS	B	R	P		P	
COM	E	N	N			
CPY	B	R	P			4
@CPYD	A	R	N	R	P	9
@CPYP	A	R	N	R	P	9
CRQ	C	R	P	8		9
@CTL	K	R	N		P	9
@CTLR	K	R	N		P	9
@CTLW	K	R	N		P	9
CVR	C	R	P	8		9
DCT	E	N	N			
DRS	B	R	P			RPQ
DVH	B	R	P		P	
DVP	B	R	P		P	
EAD	B	R	P		P	RPQ
EAXM	E	N	N			RPQ
ECA	B	R	P		P	RPQ
ECQ	B	R	P		P	RPQ
ECTM	E	N	N			9
EDP	B	R	P		P	RPQ
EFTM	E	N	N			9
ELD	B	R	P		P	RPQ
EMP	B	R	P		P	
ENB	B	R	P		P	9
ENK	E	N	N			9
ERA	B	R	P		P	9
ESB	B	R	P		P	RPQ

Op Code	Type	Addr	Tag	Dec	Ind Addr	Mode
ESNT	B	R	P		P	9
EST	B	R	P		P	RPQ
ESTM	E	N	N			9
ETM	E	N	N			
ETT	E	N	N			4
ETTX	E	NT	P			9
EUA	B	R	P		P	RPQ
FAD	B	R	P		P	
FAM	B	R	P		P	9
FDH	B	R	P		P	
FDP	B	R	P		P	
FIVE	A	P	P	P		9
FMP	B	R	P		P	
FOR	A	P	P	P		
FOUR	A	P	P	P		9
FRN	E	N	N			9
FSB	B	R	P		P	
FSM	B	R	P		P	9
FVE	A	P	P	P		
HPR	B	N	N			
HTR	B	R	P		P	
@ICC	K(1)	N	N			9
IIA	B	N	N			9
IIL	D	R				9
IIR	D	R				9
IIS	B	R	P		P	9
IOD	B(I/O)	N	N			4
IOT	E	N	N			9
IOXY(N)	A	R	P	R	P	9
... (3)	A	P	P	P		
LAC	B	R	R			9
@LAR	K	R	N	N	P	9
LAS	B	R	P		P	9
LAXM	E	N	N			RPQ
LBT	E	N	N			
@LCC	K	R	N	N	P	9
LCHX	BT	R	P		P	9
LDA	B	R	P		P	4
LDC	B	R	R			9
LDI	B	R	P		P	9
LDQ	B	R	P		P	
LFT	D	R				9
LFTM	E	N	N			9
LGL	B	R	P			
LGR	B	R	P			9
@LIP	K	N	N	N		9
@LIPT	K	R	N	N	P	9
LLS	B	R	P			
LNT	D	R				9
LRS	B	R	P			
LSNM	E	N	N			9
LTM	E	N	N			
LXA	B	R	R			

Op Code	Type	Addr	Tag	Dec	Ind Addr	Mode
LXD	B	R	R			
MON	A	P	P	P		
MPR	B	R	P		P	
MPY	B	R	P		P	
MSE	E	P	P			
MTH	A	P	P	P		
MTW	A	P	P	P		
MZE	A	P	P	P		
NOP	B	N	N			
NTR	A	R	P	P		4
NZT	B	R	P		P	9
OAI	B	N	N			9
OFT	B	R	P		P	9
ONE	A	P	P	P		
ONT	B	R	P		P	9
ORA	B	R	P		P	
ORS	B	R	P		P	
OSI	B	R	P		P	9
PAC	B	N	R			9
PAI	B	N	N			9
PAX	B	N	R			
PBT	E	N	N			
PDC	B	N	R			9
PDX	B	N	R			
PIA	B	N	N			9
PON	A	P	P	P		
PSE	E	R	P			
PSLX	BT	R	P		P	RPQ
PTH	A	P	P	P		
PTW	A	P	P	P		
PXA	B	N	R			9
PXD	B	N	R			
PZE	A	P	P	P		
RCD	B(I/O)	P	P			
RCDX	B(I/O)	N	N			9
RCHX	BT	R	P		P	9
RCT	E	N	N			9
RDC	E	R	P			9
RDCX	E	NT	P			9
RDR	B(I/O)	R	P			4
RDS	B	R	P			
REW	B(I/O)	R	P			
REWX	B(I/O)	RT	P			9
RFT	D	R				9
RIA	B	N	N			9
@RICX	E	NT	P			9
RIL	D	R				9
RIR	D	R				9
RIS	B	R	P		P	9
RND	E	N	N			
RNT	D	R				9
RPR	B(I/O)	P	P			
RPRX	B(I/O)	N	N			9
RQL	B	R	P			

Op Code	Type	Addr	Tag	Dec	Ind Addr	Mode
@RSCX	BT	R	P		P	9
RTB	B(I/O)	R	P			
RTBX	B(I/O)	RT	P			9
RTD	B(I/O)	R	P			
RTDX	B(I/O)	RT	P			9
RTT	E	N	N			4
RUN	B(I/O)	R	P			9
RUNX	B(I/O)	RT	P			9
*** (3)	A	P	P	P		
bbb (3)	A	P	P	P		
@SAR	K	R	N	N	P	9
SBM	B	R	P		P	
SCHX	BT	R	P		P	9
SDH	B(I/O)	R	P			9
SDHX	B(I/O)	RT	P			9
SDL	B(I/O)	R	P			9
SDLX	B(I/O)	RT	P			9
SDN	B(I/O)	R	P			9
SEVEN	A	P	P	P		9
SIL	D	R				9
SIR	D	R				9
SIX	A	P	P	P		
SLF	E	N	N			
SLN	E	R	P			
SLQ	B	R	P		P	
SLT	E	R	P			
SLW	B	R	P		P	
@SMS	K	R	N		P	9
@SNS	K	N	N			9
SPR	E	R	P			
SPRX	E	R	P			9
SPT	E	R	P			
SPTX	E	R	P			9
SPU	E	R	P			
SPUX	E	R	P			9
SSLX	BT	R	P		P	RPQ
SSM	E	N	N			
SSP	E	N	N			
STA	B	R	P		P	
@STCX	BT	N	N			9
STD	B	R	P		P	
STI	B	R	P		P	9
STL	B	R	P		P	9
STO	B	R	P		P	
STP	B	R	P		P	
STQ	B	R	P		P	
STR	A	N	N	N		9
STT	B	R	P		P	9
STZ	B	R	P		P	
SUB	B	R	P		P	
SVN	A	P	P	P		
SWT	E	R	P			
SXA	B	R	R			9

Op Code	Type	Addr	Tag	Dec	Ind Addr	Mode
SXD	B	R	R			
TCH	A	R	P	N	P	9
@TCM	K(1)	R	N	R	P	9
TCNX	BT	R	P		P	9
TCOX	BT	R	P		P	9
@TDC	K	R	N	N	P	9
TEFX	BT	R	P		P	9
THREE	A	P	P	P		9
TIF	B	R	P		P	9
TIO	B	R	P		P	9
TIX	A	R	R	R		
TLQ	B	R	P		P	
TMI	B	R	P		P	
TNO	B	R	P		P	
TNX	A	R	R	R		
TNZ	B	R	P		P	
TOV	B	R	P		P	
TPL	B	R	P		P	
TQO	B	R	P		P	
TQP	B	R	P		P	
TRA	B	R	P		P	
TRCX	BT	R	P		P	9
TRS	B	R	P			RPQ
TSX	B	R	R			
TTR	B	R	P		P	
TWO	A	P	P	P		
@TWT	K	R	N	N	P	9
TXH	A	R	R	R		
TXI	A	R	R	R		
TXL	A	R	R	R		
TZE	B	R	P		P	
UAM	B	R	P		P	9
UFA	B	R	P		P	
UFM	B	R	P		P	
UFS	B	R	P		P	
USM	B	R	P		P	9
VDH	C	R	P	8		9
VDP	C	R	P	8		9
VLM	C	R	P	8		9
WDR	B(I/O)	R	P			4
WEF	B(I/O)	R	P			
WEFX	B(I/O)	RT	P			9
WPB	B(I/O)	P	P			
WPBX	B(I/O)	N	N			9
WPD	B(I/O)	P	P			
WPDX	B(I/O)	N	N			9
WPR	B(I/O)	P	P			
WPRX	B(I/O)	N	N			9
WPU	B(I/O)	P	P			
WPUX	B(I/O)	N	N			9

Op Code	Type	Addr	Tag	Dec	Ind Addr	Mode
WRS	B	R	P			
WTB	B(I/O)	R	P			
WTBX	B(I/O)	RT	P			9
WTD	B(I/O)	R	P			
WTDX	B(I/O)	RT	P			9
@WTR	A	R	N	N	P	9
WTS	B(I/O)	N	N			4
WTV	B(I/O)	N	N			
XCA	B	N	N			9
XCL	B	N	N			9
XEC	B	R	P		P	9
XIT	B	R	P		P	
@XMT	A	R	N	R	P	9
ZAC	B	N	(2)			
ZET	B	R	P		P	9
ZSA	B	R	(2)			9
ZSD	B	R	(2)			

Footnotes From the Combined Operations Table

(1) A count field in the low order position of the operation code is assembled from the fourth subfield of the variable field; e.g.,

```
ICC   ,,,4
```

(2) The following extended machine operations are included in the Combined Operations Table (listed above). They differ from the assembled machine operation codes only in that no flag will appear in the left hand margin of the listing for certain uncoded tag and decrement fields.

Mnemonic	As Assembled	Unflagged Omitted Fields
BRA	TXL	T,D
BRN	TXH	T,D
ZAC	PXD	T
ZSA	SXA	T
ZSD	SXD	T

(3) Alternate forms for PZE(. . . ,***,bbb).

Disk File Orders

The following disk file orders (see IBM 1301 Input/Output Control System for 1410 and 7000 Series Data Processing Systems, Form J28-8064-1) are included in the Combined Operations Table. The symbolic order should be written as

```
LOC     DORD     A,T,H,R
```

and will assemble as ten BCD digits in two successive locations. A subfield may be any valid FAP expression, but the expression may not be common or relocatable or an assembly error will result. Fields marked R are required; they will be flagged A if missing. They may be coded void by the use of ** or successive commas. Fields not so marked, if coded, will be flagged F and ignored. Neither of these flags will indicate an assembly error. All are marked with @ to indicate they are new.

MNEMONIC	Access and Module	Track	Head	Record	Mode
@DEBM	R				9
@DNOP	R	R	R	R	9
@DREL	R				9
@DSAI	R	R	R	R	9
@DSBM	R				9
@DSEK	R	R	R	R	9
@DVCY	R	R	R	R	9
@DVHA	R	R	R		9
@DVSR	R	R	R	R	9
@DVTA	R	R	R	R	9
@DVTN	R	R	R	R	9
@DWRC	R	R	R	R	9
@DWRF	R	R	R	R	9

APPENDIX II: SYSTEM SYMBOL TABLE, FORTRAN MONITOR

The following is a list of symbols and octal definitions, which can be defined with the use of the SST control card, for the assembler operating under the FORTRAN Monitor.

CORE ALLOCATION SYMBOLS

TOPMEM	77777	Top of available memory
BOTIOP	73400	Bottom of I/O package
BOTTOM	00144	Bottom of available memory
(PCBK)	00143	Object program program break, , common break
DATEBX	00142	Monitor date cell
LINECT	00141	Monitor job line count, , FORTRAN page number, label flag in prefix
FLAGBX	00140	Monitor flag cell

TAPE ASSIGNMENT SYMBOLS

SYSTAP	00001	System tape
FINTAP	00002	First intermediate tape
SINTAP	00003	Second intermediate tape
TINTAP	00004	Third intermediate tape
MINTAP	00005	Monitor input tape
MLSTAP	00006	Monitor listing tape
MBNTAP	00007	Monitor punch tape
MCHTAP	00010	Monitor intermediate chain tape
SNPTAP	00011	Snap tape
LIBTAP	00001	System library tape

ENTRY POINTS TO I/O PACKAGE

(LOAD)	73400	Call next record on system tape
(DIAG)	73403	Call diagnostic record, source or machine error
(TAPE)	73406	Initiate tape operation
(PRNT)	73411	Initiate on-line print
(PNCH)	73414	Initiate on-line punch
(READ)	73417	Initiate on-line card read
(STAT)	73422	Locate tape statistic tables
(REST)	73425	Restore console
(STDN)	73431	Set tape density
(SECL)	73447	Call source program error record
(MECL)	73450	Call machine error record
(DGLD)	73451	Restore memory and halt

PARAMETERS VARIABLE AT EDIT TIME

(ES1S)	73452	Physical sense switch corresponding to END card setting 1
(ES2S)	73453	Physical sense switch corresponding to END card setting 2

(ES3S)	73454	Physical sense switch corresponding to ECS 3
(ES4S)	73455	Physical sense switch corresponding to ECS 4
(ES5S)	73456	Physical sense switch corresponding to ECS 5
(ES6S)	73457	Physical sense switch unassigned
(PGCT)	73460	Listing page dimensions
(LIBT)	73461	System library tape assignment

COMMON COMMUNICATIONS REGION

(FGBX)	73467	Location of Monitor flag cell
(LNCT)	73470	Location of Monitor line count
(DATE)	73471	Job date
(SNCT)	73472	Snapshot tape file count
(MSLN)	73473	Flag for diagnostic record
(ENDS)	73474	End card settings
(SCHU)	73501	Data transmission error information

DEFINITIONS OF OPERATION MNEMONICS
TO INITIATE TAPE OPERATION

		Operation	Information	End File	Check
(WROW)	40031	Write	Row binary		Immediate
(RDEC)	40016	Read	Decimal	Permitted	Immediate
(WEFC)	40015	Write	End file		Immediate
(RBEC)	40014	Read	Binary	Permitted	Immediate
(WDNC)	40013	Write	Decimal		Immediate
(RDNC)	40012	Read	Decimal	Prohibited	Immediate
(WBNC)	40011	Write	Binary		Immediate
(RBNC)	40010	Read	Binary	Prohibited	Immediate
(RDEP)	40006	Read	Decimal	Permitted	Later
(WEFP)	40005	Write	End File		Later
(RBEP)	40004	Read	Binary	Permitted	Later
(WDNP)	40003	Write	Decimal		Later
(RDNP)	40002	Read	Decimal	Prohibited	Later
(WBNP)	40001	Write	Binary		Later
(RBNP)	40000	Read	Binary	Prohibited	Later
(SKDC)	20012	Skip	Decimal		Immediate
(SKBC)	20010	Skip Backspace	Binary		Immediate for BTT
(SKDP)	20002	Skip	Decimal		Later
(SKBP)	20000	Skip Backspace	Binary		Later Not significant
(CHKU)	10000	Delay and check last information transmitted			
(TPER)	04000	Error return for transmitted information found improper			
(SNAP)	01000	Dump panel and memory selectively on SNPTAP			

APPENDIX III: FAP OPERATING UNDER THE BASIC MONITOR (IBSFAP)

Introduction

FAP is available as an independent system called IBSFAP operating under the IBSYS Basic Monitor. IBSFAP is completely independent of the FORTRAN Monitor. Complete IBSYS specifications will be found in the IBM Basic Monitor (IBSYS) bulletin (Form J28-8086).

IBSFAP Operations

Control Cards

The following Basic Monitor card

1	16
$EXECUTE	IBSFAP

will call Pass 1 of IBSFAP (IBSFP1).

A minimal monitor in IBSFP1 recognizes the following control cards:

Format		Use
1	7 16	
$ID		Installation accounting request
$IBSYS		Return to Basic Monitor
	*ROW	Specifies off-line row-binary card output
	*PRINT	Specifies on-line assembly listing
	*CARDS COLUMN	Specifies on-line column-binary card output
	*CARDS ROW	Specifies on-line row-binary card output
	*remarks preceding *FAP	Specifies remarks to be printed by Monitor on-line
	*FAP	Specifies beginning of a FAP assembly or updating run

The cards beginning in column 1 are Basic Monitor cards, while the rest (preceded by an asterisk) are IBSFAP control cards.

Input/Output Units

Output from IBSFAP is always blocked on SYSOU1, for the 720 Printer, five lines per block, and binary punched output is placed on SYSPP1 in card image form. The on-line options supplement, but do not replace, these operations.

IBSFAP uses logical units for updating and referencing. The input/output unit correspondence to SYSUNI functions is as follows:

SYSUNI	Logical I/O Unit	Function
SYSLB1, (2, 3, or 4)	1	IBSFAP system
SYSUT2	2	Scratch
SYSUT4	3	Scratch
SYSCK1	4	Scratch, on-line card output only
SYSIN1	5	IBSYS input
SYSOU1	6	IBSYS BCD list output
SYSPP1	7	IBSYS binary card output
SYSUT1	8	Available for updating
SYSUT3	9	Available for updating
SYSCK2	10	Available for updating

For maximum running efficiency, it is suggested that SYSUT2 and SYSUT4 be on channels different from those of SYSIN1 and SYSOU1, especially on large assemblies or updatings, and that the update input tape be on the same channel as SYSIN1.

Example

The following would be the sequence of cards on SYSIN1 to do multiple assemblies:

Format		Function
1 7	16	
$DATE	mmddyy	
$ATTACH	(unit)	Any number of unit assignments as necessary
$AS	(function)	
$EXECUTE	IBSFAP	
*...		Monitor control cards, as required
*FAP		
*page heading remarks		
COUNT	m	
...		Source input deck to be assembled
...		
END		
*...		Monitor control card, as required

*page heading remarks

```
                COUNT       m
                ABS
                ...                 Additional source input decks to
                                      be assembled
                ...
                END
$IBSYS                              Return to IBNUC
```

Special IBSFAP Feature

The pseudo-operation, SST, provides IBNUC and IOEX symbolic definition entries as follows:

1. All the one-entry points starting with SYSTRA through SYSLDR in IBNUC, plus the current value of SYSEND and SYSORG.
2. All the SYSUNI functions.
3. The communication region to IOEX from ACTIV to TRPSW as follows:

IOEX COMMUNICATION TABLE

IOEX Symbol	Equivalent Assembly Symbol, If Used	Function
	ENTRY POINTS TO IOEX SUBROUTINES	
ACTIV	(ACTIV	Activate Routine & Test
ACTIV+1	(ACTVX	Activate Routine Without Test
NDATA	(NDATA	Non-Data Select & Test
NDATA+1	(NDSLX	Non-Data Select Without Test
MWR,PROUT	(PROUT	Message Writer
PUNCH	(PUNCH	Hollerith Punch
FRCHX	(FRCHX	Free Channel X
PAWSX	(PAWSX	Error Pause
PAUSX	(PAUSE	Operator Action Pause
STOPX	(STOPX	Dead Stop Entry
SYMUNI	(SYMUN	Symbolic Unit Conversion
BCVDEC-1	(DECVD	Binary to Decimal Conversion--AC Decrement
BCVDEC	(DECVA	Binary to Decrement Conversion--AC Address
CKWAT	(CKWAT	Checkpoint Wait
BCD5-1	(BCD5R	Octal to BCD Bits 3-17, MQ
BCD5	(BCD5X	Octal to BCD Bits S-14, MQ
CVPRT	(CVPRT	Convert and Append to Tape Message
STOPD	(STOPD	Dead Stop Location

IOEX Symbol	Equivalent Assembly Symbol, If Used	Function
	INDIRECT REFERENCES TO CHANNEL DATA TABLES	
CHXAC	(CHXAC	Channel Activity
URRXI	(URRXI	Redundancy Count
RCTXI	(RCTXI	Redundancy Control
RCHXI	(RCHXI	Reset Load Channel
TCOXI	(TCOXI	Channel Delay
TRCXI	(TRCXI	Tape Redundancy Test
ETTXI	(ETTXI	End Tape Test
TEFXI	(TEFXI	End File Test
TRAPX	(TRAPX	Current Traps Enabled
	TRAP DATA	
TRAP	(TRAPS	Current traps enabled
COMM	(COMMM	Store Channel Results
LTPOS	(LTPOS	Tape Position
IOXSI	(IOXSI	Sense Indicators
	ACTIVITY SWITCHES	
CHPSW	(CHPSW	Checkpoint
TRPSW	(TRPSW	Trap

PUBLICATIONS

Following is a list of IBM publications which may be of interest to the reader:

REFERENCE MANUALS

Form Number	Title
A22-6528-1	IBM 7090 Data Processing System
A22-6536	IBM 709 Data Processing System
C28-6054-2	709/7090 FORTRAN Programming System
C28-6066-3	709/7090 FORTRAN Operations

GENERAL INFORMATION MANUALS

D22-6508-2	IBM 709/7090 Data Processing System
F28-8043	IBM Commercial Translator
F28-8053-1	COBOL
F28-8074-1	FORTRAN

BULLETINS

G22-6505-1	IBM 7090 Data Processing System
J28-6098-1	FORTRAN Assembly Program (FAP) for the IBM 7090
J28-6114-1	32K 709/7090 FORTRAN: Double-Precision and Complex Arithmetic
J28-6132	Advance Specifications: 7090 FORTRAN and FORTRAN Assembly Program (FAP)
J28-6133	32K 709/7090 FORTRAN: Source Language Debugging at Object Time
J28-6135	32K 709/7090 FORTRAN: Adding Built-In Functions
J28-6166	SHARE 7090 9PAC: Part 1 - Introduction and General Principles
J28-6167	SHARE 7090 9PAC: Part 2 - The File Processor
J28-6168	SHARE 7090 9PAC: Part 3 - The Reports Generator
J28-6169	IBM 709/7090 Commercial Translator Processor
J28-6173	IBM 7000/1400 Output Editing System-Preliminary Reference Manual
J28-6174	S-Program for the IBM 7090: Preliminary Specifications
J28-6184	IBM 7094 Programs and Programming Systems
J28-8086	7090 Operating Systems: Basic Monitor (IBSYS)

IBM
International Business Machines Corporation
Data Processing Division, 112 East Post Road, White Plains, N. Y.

Printed in U. S. A. J28-6186