AA61649-00

SERIES 6000
MACRO ASSEMBLER
GENERAL SPECIFICATION

July, 1972

Datacraft Corporation

1200 N. W. 70th Street P. O. Box 23550 Fort Lauderdale, Florida 33307 (305) 974-1700

LIST OF EFFECTIVE PAGES

TOTAL NUMBER OF PAGES IN THIS PUBLICATION IS 45
CONSISTING OF THE FOLLOWING:

Page No.	Change No.	Page No.	Change No.	Page No	Change No.
Title	Original				
A	Original				
i thru ii	Original				
1-1	Original				
2-1 thru 2-2	Original				
3-1 thru 3-5	Original				
4-1 thru 4-13	Original				
5-1 thru 5-5	Original				
6-1 thru 6-3	Original				
7-1 thru 7-3	Original				
A-1 thru A-9	Original				

Upon receipt of the second and subsequent changes to this technical document, personnel responsible for maintaining this publication in current status will ascertain that all previous changes have been received and incorporated. Action should be taken promptly if the publication is incomplete.

CONTENTS

CONTENTS (CONT'D.)

SECTION I
INTRODUCTION

1-1 SCOPE OF SPECIFICATION

This document contains a detailed description of the pseudo-operations, macro capabilities and formats employed by the Series 6000 Macro Assembler. Examples of usage of the various pseudo-operations are given as well as examples of macro definition and calling modes.

This specification does not contain instruction descriptions or assembler or computer operating instructions.

1-2 GENERAL DESCRIPTION

The Series 6000 Macro Assembler is a two-pass processor system. For the most part, source language statements provide a one-to-one map into machine language instructions and single or multiple data word configurations.

Source statements may contain a symbolic label, a mnemonic instruction or pseudo-operation, an operand field, and a comments field. Target language is Series 6000 machine language which is listed as a binary file whose record format is identical to the input record format of the Link Loader.

SECTION II
SOURCE LANGUAGE FORMAT

2-1 SCOPE

This section describes the format for source input statements to the Series 6000 Macro Assembler.

2-2 LABEL FIELD (COLUMNS 1-7)

The label field may contain an asterisk (*) in column 1 or a symbolic label, or may be left entirely blank.

An asterisk in column 1 causes all succeeding columns in the line to be treated as comments. No binary output will be generated. A plus (+) or minus (-) sign immediately following the asterisk affects the listing. Refer to Section 6 (Listing Options) for the exact effect.

When using a symbolic label, the first character must be alphabetic. Any succeeding characters may be alphabetic, numeric, or special symbols. When a label is encountered, it is assigned the address of the instruction or pseudo-operation being labeled.

NOTE

The arithmetic operators "+" and "-", a comma (,) or a blank may not be used in a symbolic label.

Examples (valid labels):

```
ENTRNCE     ***
EXIT2/2     BUC     Ø,K
T$A#43      TMA*    DACB1
```

2-3 OPERATION FIELD (COLUMNS 9-12)

The operation field may contain a three-character computer instruction mnemonic, or a three- or four-character pseudo-operation (see Section IV), or a one-to-six character macro reference (see Section V).

An asterisk (*) in column 12 is used to indicate an indirect memory reference except in the case of the computer input/output instructions, where the asterisk is used as an Override or Merge specification. These special cases are defined in the Appendix in Table A-14.

2-4 OPERAND AND COMMENTS FIELD (COLUMNS 15-72)

This field may contain an operand constant, address, etc., as dicated by the particular instruction. (Reference Tables A-4 through A-19 in the Appendix for a detailed listing.) The operand must begin in column 15 and must be separated from the comments by at least one blank column. For those instances where no operand is required, column 15 must remain blank.

2-5 SEQUENCE FIELD (COLUMNS 73-80)

The sequence field may contain any identification (e.g., a card sequence number) the user desires. This field will be ignored by the Assembler, but will be output as part of the listing.

SECTION III
OPERAND FORMATS

3-1 SCOPE

This section describes the operand formats used with the Series 6000 Macro Assembler. A list of the Series 6000 instruction mnemonics and their permissible operand formats is contained in Tables A-4 through A-19 in the Appendix.

3-2 CURRENT LOCATION (*)

An asterisk (*) in the operand field indicates that the location of the current instruction is to be used as the operand address.

3-3 SYMBOLIC LABELS

A symbolic label in the operand field indicates that the address associated with the label is to be used as the operand address. A symbolic label may be declared as a common variable by the use of the "COMM" pseudo-operation (reference: Section IV).

3-4 ABSOLUTE CONSTANTS

Absolute constants in the operand field indicate that the constant (octal or decimal) is to be used as the operand address.

3-5 ADDRESS ARITHMETIC

Any combination of current location (*), symbolic labels, or absolute constants may be joined by the plus (+) or minus (-) operators to define an address. Such a combination is referred to as an "operand expression".

An expression of the form XX*YYY may be used. This allows a limited multiplication capability. XX must be a constant and YYY must be a symbol whose mode is absolute. This form may be combined with other partial operand expressions by plus (+) or minus (-) operators.

EXPRESSION MODE

The mode (absolute or relative) of any given label is determined by the AORG or RORG pseudo-operation most recently encountered prior to the label being defined. Therefore, any label encountered after an AORG pseudo-operation is considered absolute and any label after an RORG is relative. The current location (*) tag is also subject to these rules.

In determining the mode of any operand expression, consider only relative labels. If the arithmetic operators (address arithmetic) preceding relative labels do not balance (i.e., the number of plus and minus signs are not equal), the resulting expression is relative. If the arithmetic signs preceding relative labels are in balance, the resulting expression is considered absolute.

Examples of relative and absolute expressions:

```
        RORG    0
CAT     ***
DOG     ***
MICE    ***
        .
        .
        .
        TMA     CAT+1           relative expression
        TMA     MICE-CAT+1      absolute expression
        TMA     MICE-CAT+DOG    relative expression
```

3-6 INDEXED ADDRESS REFERENCE

An operand address may be appended with an index reference by placing a comma (,) in the column to the immediate right of the operand address and following the comma with the index specification (I, J or K), e.g. TMA Ø,K

3-7 TEXT

Alphanumeric text may be used as an operand by the use of leading and following quotation marks ("......"). The text will be right-justified in the instruction operand and any unused portion of the operand will be filled with ZEROs. Text may not be used with address arithmetic.

Examples of operand text:

```
TOA     "AB"    will generate '62540502 }
COB     "Z"     will generate '00140132 }   These show the completely
TOB     "$"     will generate '00030044 }   assembled instruction.
```

3-8 LITERAL ADDRESS

A literal address is indicated by an equal sign (=), in column 15, followed by a constant. The literal address allows a data constant (see Section IV) to be assigned a nonprogram location and permits the address of that assignment to be used as the operand address.

NOTE

The expression mode (relative or absolute) of a literal address is the same as the mode of assembly upon encountering an END pseudo-operation (see Section IV).

If the machine representation of a constant is more than one word, the assigned address of the first word is used as the operand address. Any identical constants are assigned the same location to conserve storage.

Any memory reference instruction using a literal address may not be appended with indirect or index references.

Examples of literal addresses:

```
TMA     =B1B12B23
TMD     =12.5D-10
TMR     ="I/O CALL ERROR"
TMI     =/'22,40,'77/
```

3-9 EXTERNAL REQUEST

An unconditional external request is indicated by a "dollar sign" ($), in column 15, followed by one to six alphanumeric or special characters (except "+", "-", a comma or a "blank"). The first character must be alphabetic.

A conditional external request is indicated by two "dollar signs", in columns 15 and 16, followed by a one to six character name.

NOTE

A conditional external request will be satisifed by the link loader only if an unconditional external request of the same name has preceded it.

Address arithmetic may not be used with external requests; however, indirect and index references may be employed depending on the instruction.

Examples of external requests:

```
BSL     $FLOAT      unconditional external request
TMA*    $CAT,J      unconditional external request
BLL     $$A22       conditional external request
```

3-10 EXTERNAL EQUIVALENCE REQUEST

The purpose of an external equivalence request is to merge an externally defined data constant with the frame word of the instruction.

An external equivalence request is indicated by a number sign (#) in column 15, followed by one to six alphanumeric or special characters (except "+", "-", a comma, or a blank). The first character of the name must be alphabetic.

Examples of external equivalence requests:

```
TOA     #CBITS
OCW     #C/U
```

During the loading process, all external equivalence requests must have been preceded by their corresponding external equivalence definitions (refer to Paragraph 4-11).

3-11 MEMORY REFERENCING BOOLEAN INSTRUCTIONS (BIT PROCESSOR)

FORMAT I

```
MEN     X,Y
```

where: MEN is the three character mnemonic.
X is an absolute expression indicating the bit specification.
Y is an absolute expression indicating the non-negative displacement from the contents of base register V.

Examples of Format I

```
FBUF        BLOK    3
FLGSET1     EQIV    0
FLGSET2     EQIV    1
FLGSET3     EQIV    2
F1          EQIV    0
F2          EQIV    1
F3          EQIV    2
F4          EQIV    3
              .
              .
              .
F24         EQIV    23
              .
              .
              .
            TLO     FBUF
            TKV
            QBM     F1, FLGSET1
            QBM     3, FLGSET2
            QBM     19,2
              .
              .
              .
```

FORMAT II

```
MEN     X
```

where: MEN is the three character mnemonic.
X is an absolute expression indicating the bit specification and the displacement from the contents of base register V.

Examples of Format

```
B0          EQIV    0
B1          EQIV    256
B2          EQIV    257
B3          EQIV    258
.           .       .
.           .       .
.           .       .
B23         EQIV    278
FLAGBUF     BLOK    256
FLAGEND     EQIV    *
            AORG    0
FLAG1       EQIV    B0+*
FLAG2       EQIV    B1+*
                    .
                    .
                    .
FLAG24      EQIV    B23+*
            AORG    1

ALPHA       EQIV    B0+*
BETA        EQIV    B1+*
DELTA       EQIV    B2+*
                    .
                    .
                    .
            RORG    FLAGEND
            TLO     FLAGBUF
            TKV
            QBM     FLAG2
            QBM     DELTA
                    .
                    .
                    .
```

SECTION IV
PSEUDO-OPERATIONS

4-1 SCOPE

This section describes the pseudo-operations that will be processed by the Series 6000 Macro Assembler.

4-2 AORG

The AORG pseudo-operation sets the mode of all following labels to absolute and sets the program location of the next instruction to the contents of the absolute expression in the operand field. If a label is present, it is assigned the value of the operand expression. If symbolic labels are present in the operand expression, they must have been previously defined. If an operand error is encountered, the letter "O" will be placed in the error field of the list output and all subsequent binary output will be suspended.

Examples of AORG statements:

```
        AORG    Ø                   program location set to absolute Ø
A       DATA    Ø
        DATA    Ø
B       DATA    Ø
        .
        .
        .
        AORG    B-A+'1000           program location set to absolute ' 1002
```

4-3 RORG

The RORG pseudo-operation sets the mode of all following labels to relative and sets the next instruction's program location to the contents of the absolute or relative expression in the operand field. Initially, the mode of the assembly is set to relative and the program location counter is set to zero. If a label is present, it is assigned the value of the operand expression. If symbolic labels are present in the operand expression, they must have been previously defined. If an operand error is encountered, the letter "O" will be placed in the list output error field and all subsequent binary output will be suspended.

Examples of RORG statements:

```
        RORG    Ø                   program location set to relative Ø
        DATA    Ø
        DATA    Ø
A       DATA    Ø
        .
        .
        .
        RORG    A+1                 program location set to relative 3
```

4-4 BLOK

This pseudo-operation reserves a block of storage (n locations). (n = the value of the absolute expressions in the operand field.) If a label is present, it is assigned the first location in the block. If symbolic labels are present in the operand expression, they must have been previously defined. If an operand error is encountered, the letter "O" will be placed in the list output error field and all subsequent binary output will be suspended.

Examples of BLOK statements:

```
A     TMA     '1000,I
      TAM     '2000,I
      NOP
B     BSL     ROUTINE
      .
      .
      .
      BLOK    B-A+1       reserve 4 locations
      BLOK    100         reserves 100 locations
```

4-5 EQIV

This pseudo-operation assigns the value and mode of the operand expression to the label in the label field. All labels in the operand field must have been previously defined.

Examples of EQIV statements:

```
      RORG    Ø
A     ***
B     ***
X     EQIV    B           X is assigned a relative value of 1
Y     EQIV    '0403       Y is assigned an absolute value of '0403
Z     EQIV    B-A+1       Z is assigned an absolute value of 2
```

4-6 FORM

The FORM pseudo-operation specifies the inner fields to be created within a 24-bit word. The items of any subsequently-encountered formatted data constants will then be aligned (right-justified) within their respective inner fields (see Formatted Data Constant).

The format of the FORM pseudo-operation is as follows:

FORM $N_1, N_2, \ldots\ldots N_n$

where: N_k is a non-zero unsigned decimal integer

and $\sum_{i=1}^{n} N_i = 24 \qquad 1 \leq n \leq 24$

N_1 causes a left-justified inner field consisting of N_1 bits to be established.

N_2 causes a left-justified (to the N_1 subfield) inner field of N_2 bits to be established.

.
.

N_n causes a left-justified (to the N_{n-1} subfield) inner field of N_n bits to be established.

If an improper operand is specified, the letter "O" will be placed in the list output error field and a form of 24 will be established. If a symbolic label is encountered in the label field, it will not be assigned.

Examples of FORM statements:

```
FORM      12,12
.
.
.
DATA      /1,1/                  will generate '00010001
FORM      6,6,6,6
.
.
.
DATA      /1,1,1,1/              will generate '01010101
          9,1,4,5,5
.
.
.
DATA      /'330,0,'02,'15,'01/   will generate '33004641
```

4-7 DATA

When using the DATA pseudo-operation, the operand field may contain any number and combination of the data constants defined in the following paragraphs.

If more than one item is present (items are separated by commas), they will occupy sequential memory locations. If a label is present, it will be assigned the location of the first data items.

4-7.1 Single Integer

A single integer data constant consists of an optional sign (+ or -) and 1 to 7 decimal digits.

Examples of single integers:

```
DATA      +428      will generate '00000654
DATA      1         will generate '00000001
```

4-7.2 Double Integer

A double interger constant consists of an optional + or - sign and 1 to 12 decimal digits, followed by the letter "D". Two words will be stored in sequential memory locations.

Examples of double integers:

```
DATA    10234876293D    will generate '00002304-'02750605
DATA    1D              will generate '00000000-'00000001
DATA    -1D             will generate '77777777-'37777777
```

4-7.3 Single-Precision Real

A single-precision real data constant consists of an optional sign (+ or -), 1 to 7 decimal digits mixed with a decimal point and/or followed by a decimal exponent.

The exponent consists of the letter "E" preceding a decimal number (with optional + or - sign) between +37 and -37.

Two words will be stored in sequential memory locations.

Examples of single-precision real constants:

```
DATA    +24.4           will generate '30314632-'00000005
DATA    5.0E-10         will generate '21134060-'00000342
DATA    2.4E+5          will generate '35230000-'00000022
DATA    0E0             will generate '00000000-'00000201
DATA    1E0             will generate '20000000-'00000001
```

4-7.4 Double-Precision Real

A double-precision real constant consists of an optional sign (+ or -), 1 to 12 decimal digits mixed with a decimal point, and/or followed by a decimal exponent.

The exponent consists of the letter "D" preceding an optionally-signed decimal number between +37 and -37.

Two words will be stored in sequential memory locations.

Examples of double-precision real constants:

```
DATA    12.D0           will generate '30000000-'00000004
DATA    -1D-10          will generate '44406200-'23050337
DATA    5.56185D0       will generate '26176526-'15475003
```

4-7.5 Single-precision Fixed-Point

A single-precision fixed-point data constant consists of an optional sign (+ or -), 1 to 7 decimal digits mixed with an optional decimal point and followed by the letter "B" and an unsigned 2-digit decimal integer. This 2-digit number indicates the scaling position; i.e., a power of two by which the number is to be multiplied.

One word will be stored.

Examples of single-precision fixed-point constants:

DATA	15.2B5	will generate '00000746
DATA	1B6	will generate '00000100
DATA	-5.3B12	will generate '77725464

4-7.6 Double-Precision Fixed-Point

A double-precision fixed-point data constant consists of an optional sign 1 to 12 decimal digits mixed with an optional decimal point, and followed by the letter "X" and an unsigned 2-digit decimal integer. This 2-digit number indicates the scaling position; i.e., a power of two by which the number is to be multiplied.

Two words will be stored in sequential memory locations.

Examples of double-precision fixed-point constants:

DATA	15.2X5	will generate '00000000-'00000746
DATA	1X32	will generate '00001000-'00000000
DATA	-5.3X28	will generate '77777526-'14631464

4-7.7 Octal Constants

An octal data constant is designated by an apostrophe (') followed by 1 to 8 octal digits (0-7).

Examples of octal constants:

DATA	'77	will generate '00000077
DATA	'12345670	will generate '12345670

4-7.8 Binary Constants

A binary data constant is designated by the letter "B" followed by a decimal integer ($\leq$23) indicating the unitary bit position (0-23) that is to be set. Any number of binary constants may be strung together; however, no spaces or blanks are permitted between constants in a string.

Examples of binary constants:

DATA	B0B10B22	will generate '20002001
DATA	B23	will generate '40000000

4-7.9 Formatted Data Constant

The assembly format for formatted data constants is as follows:

$/A_1, A_2, \ldots\ldots, A_n/$

where: A_k is an octal constant, an optionally signed decimal constant, or text (consisting of 2 or less characters).

The constants within the slashes are truncated to match the number of bits specified in the most recently encountered FORM pseudo-operation. They are then placed in a 24-bit word according to the format specified in the FORM pseudo-operation.

Example:

```
FORM    19,5
DATA    /-1, "K"/          will generate '77777753
```

4-7.10 Text

Alphanumeric text may be designated as a data constant by the use of leading and following quotation marks ("....."). The text will be left-justified, 3 characters per word, in the generated word(s). Any unused bytes will be filled with blanks.

4-7.11 Truncated Text

Truncated alphanumeric text may be designated as a data constant by placing the letter "T" in front of the leading quotation marks. The text will be left-justified, 4 characters per word, in the generated word(s). Any unused poritions of a word will be filled with truncated blanks.

Examples of truncated text:

```
DATA    T"HOLD"
DATA    T"ALL GOOD MEN"
```

4-8 RDAT

The format for the RDAT pseudo-operation is as follows:

RDAT $X(C_1, C_2, \ldots\ldots, C_n)$

where X is a non-zero, unsigned, decimal constant indicating the number of times the following constant string is to be generated, and C_i is a valid DATA statement constant.

Examples of RDAT pseudo-operations:

```
RDAT    5('40,"ABCD",B1B13, 12.5)
RDAT    10(" ")
```

4-9 COMM

A COMM statement is of the form:

COMM $/X_1/A_1/X_2/A_2\ldots/X_n/A_n$

where each A is a non-empty list of symbolic labels which may be subscripted. Subscripting takes the form:

A(i)

where i is a non-zero, unsigned octal or decimal constant.

Each X is a one to six character name, the first character of which must be alphabetic. Any succeeding character may be alphanumeric or special characters except "+", "-", ",", " ", "/", or "(". X may also be empty indicating that blank common is the name. If X_1 is empty, the first two slashes are optional. Each X is a block name, a name that bears no relationship to any symbolic label having the same name.

In any given COMM statement, the entities occuring between block name X and the next block name (or the end of the statement if no block name follows) are declared to be in common block X. All entities from the beginning of the statement until the appearance of a block name, or all entities in the statement if no block name appears, are declared to be in "blank or unlabeled common".

A given common block name may occur more than once in a COMM statement or in a program unit. The assembler will string together in a given common block all entities so assigned in the order of their appearance. The first element of subscripted label will follow the immediately preceding entity, if one exists, and the last element of a subscripted label will immediately precede the next entity, if one exists.

If an operand error is encountered, all COMM statements within a program unit will be listed with an "O" in the list output error field and all binary output will be suspended.

Examples of the COMM statements:

```
COMM    /ALPHA/A,B/BETA/Y(50),Z
COMM    I,J/ALPHA/C('100),D
```

where common variable:	has an associated displacement of:	from block name:
A	0	ALPHA
B	1	ALPHA
Y	0	BETA
Z	50	BETA
I	0	(blank)
J	1	(blank)
C	2	ALPHA
D	'102	ALPHA

4-10 XDEF

The XDEF pseudo-operation causes an external definition to be issued. The XDEF statement has the following format:

```
XDEF        NAME, ADDR
```

where: NAME is the external name being defined, consisting of one to six alphanumeric or special characters (except "+", "-", "," or " ") the first of which must be alphabetic.

ADDR is the relative expression specifying the address to be associated with the name.

The XDEF statement must precede all other assembler statements other than comments. If ordering is not observed or if an operand error is encountered, the letter "O" will be placed in the list output error field and all subsequent binary output will be suspended.

Examples of the XDEF statement:

```
XDEF        SUB1,*
XDEF        TABLE1,TABLE1
XDEF        TABLE2,TABLE1+10
```

4-11 XEQV

The purpose of the XEQV pseudo-operation is to define a 24-bit external equivalence data constant.

The format of the XEQV pseudo-operation is as follows:

```
XEQV        NAME,X
```

where: NAME is the external equivalence name being defined, consisting of one to six alphanumeric or special characters (except "+", "-", a comma, or a blank), the first of which must be alphabetic.

X is an absolute expression specifying the data constant to be associated with the name.

Examples of XEQV pseudo-operations:

```
XEQV        CBITS,'301
XEQV        C/U, '0401
```

4-12 NAME

The NAME pseudo-operation defines, for the system, the name of the load module. The NAME statement has the following format:

```
NAME        XXXXXX
```

where: XXXXXX is the module name being defined, consisting of one to six alphanumeric or special characters (except "+", "-", "," or " ") the first of which must be alphabetic.

4-13 DAC

The DAC pseudo-operation is considered as a 16-bit memory reference instruction, however, any indirect or index specifications are placed in bit positions 21 through 23 to conform to the indirect memory reference format as defined in the Series 6000 Computer System Reference Manual.

Example of the DAC pseudo-operation:

```
DAC*        TABLE,I
```

4-14 BAC

The BAC pseudo-operation is used to create a byte address. The generated word can be loaded to index register I or J for use with the RBM, EMB, BBI, or BBJ instruction. The byte specification is placed in bits 22 and 23 of the generated word. The rest of the word is generated as either a 22-bit absolute value or a 16-bit relocatable memory reference. If the mode is absolute, the value may be positive or negative. External addresses are legal if the byte displacement is 2 or less. The format of the operand is:

base-address, byte-displacement from the initial character.

Examples of the BAC pseudo-operation:

```
BAC         0,1
BAC         ="0123456789",7
```

The second example generates a byte address that points to the character "7" in the literal string.

Example of Use:

The following sequence of instructions will set the buffer starting at byte 2 of location '100 through byte 3 of location '102 to blanks:

```
    TOB     " "
    TMI     XX
    RBM     '100 +3
    BBI     *-1
    :
    :
XX  BAC     -3, 2
```

4-15 ***

This pseudo-operation reserves one word of storage, which will be set to zero.

4-16 ZZZ

The pseudo-operation is considered as a 15-bit memory reference instruction containing an operation code of '00, which may be appended with indirect and index references.

4-17 OCTAL OPERATION CODE

An octal operation code is indicated by placing an apostrophe (') in column 9 and is to be followed by a two-digit octal constant. It will be considered as a 15-bit memory reference operation code and must include an operand. It may be appended with indirect or index references.

Examples of Octal Operation Codes:

```
'24          ALPHA + 10
'24*         BETA,I
```

4-18 HOLD

The HOLD pseudo-operation causes assembly to be suspended until an operator response is received. This is accomplished via a call to the system service $HOLD. (Refer to the appropriate operating system general specification.) The contents of the operand field will be listed on the operator communications device. The HOLD statement itself will not be listed, however, its presence will be reflected in the card sequence field.

The HOLD statement will be recognized within a conditional assembly block regardless of the skip condition (reference: Paragraph 4-21).

Examples of the Hold pseudo-operation:

```
HOLD         PLACE TAPE 2 IN READER
```

4-19 EJCT

EJCT pseudo-operation causes a Top of Form command to be issued to the list output device preceding the next line of output. The EJCT statement itself will not be listed; however, its presence will be reflected in the card sequence field.

4-20 IDEN

The IDEN pseudo-operation causes the entire Operand and Comments field (columns 15-72), followed by two blank lines, to be printed as the heading on each page of the list output device. A top of form command will be issued to the list output device prior to the next line. The IDEN statement itself will not be listed; however, its presence will be reflected in the card sequence field.

Example of the IDEN pseudo-operation:

```
IDEN        USER'S PROGRAM IDENTIFICATION
```

4-21 CONDITIONAL ASSEMBLY PSEUDO-OPERATIONS

These pseudo-operations cause all ensuing statements to be skipped by the assembler if the condition specified in the operand field is true. Assembly will resume upon encountering an ESKP (End Skip) pseudo-operation corresponding (i.e., on the same "nesting level") to the particular Skip pseudo-operation.

The relationship between Skip pseudo-operations and ESKP pseudo-operations is such that the first ESKP encountered defines the range of the last Skip encountered, the second ESKP defines the range of the next-to-last Skip, etc. This relationship is referred to as a conditional assembly block. All Skip pseudo-operations may be "nested" to any level.

If a skip condition is true for a specific conditional assembly block, any inner blocks (i.e., on a lower "nesting level") will be skipped regardless of their condition.

The conditional Skip pseudo-operations and the terminator pseudo-operation defined is as follows:

SKOS -- Skip if Operand Set -- will cause "skipping" if the value of the absolute expression in the operand field is non-zero.

SKOZ -- Skip if Operand Zero -- will cause "skipping" if the value of the absolute expression in the operand field is zero.

SKFS -- Skip if Flags Set -- will cause "skipping" if all system flags specified by the binary constant in the operand field are set (non-zero).

SKFZ -- Skip if Flags Zero -- will cause "skipping" if all system flags specified by the binary constant in the operand field are zero.

SKOP -- Skip if Operand Positive -- will cause "skipping" if the value of the absolute expression in the operand field is greater than zero.

SKON -- Skip if Operand Negative -- will cause "skipping" if the value of the absolute expression in the operand field is less than zero.

SKOB -- Skip if Operand Blank -- will cause "skipping" if column 15 if blank.

SKNB -- Skip if Operand Not Blank -- will cause "skipping" if column 15 is not blank.

SKOI -- Skip if Operands Identical -- will cause "skipping" if the two operands are identical. The operands must not contain a comma or a blank and there must be two operands. However, there is no other restriction on the characters in the operands, and they may in fact be null operands.

SKOD -- Skip if Operands Different -- will cause "skipping" if the two operands are not identical. The operands are subject to the same restrictions as in SKOI.

ESKP -- End Skip -- signifies the end of a conditional assembly block.

If a symbolic label is present in the label field of a conditional assembly or terminator pseudo-operation, it will not be assigned. If an imbalance exists between Skip pseudo-operations and their respective ESKP pseudo-operations, or of an operand error is encountered, assembly will be terminated and control will be returned to the system.

Examples of SKips that cause the code following them to be "skipped".

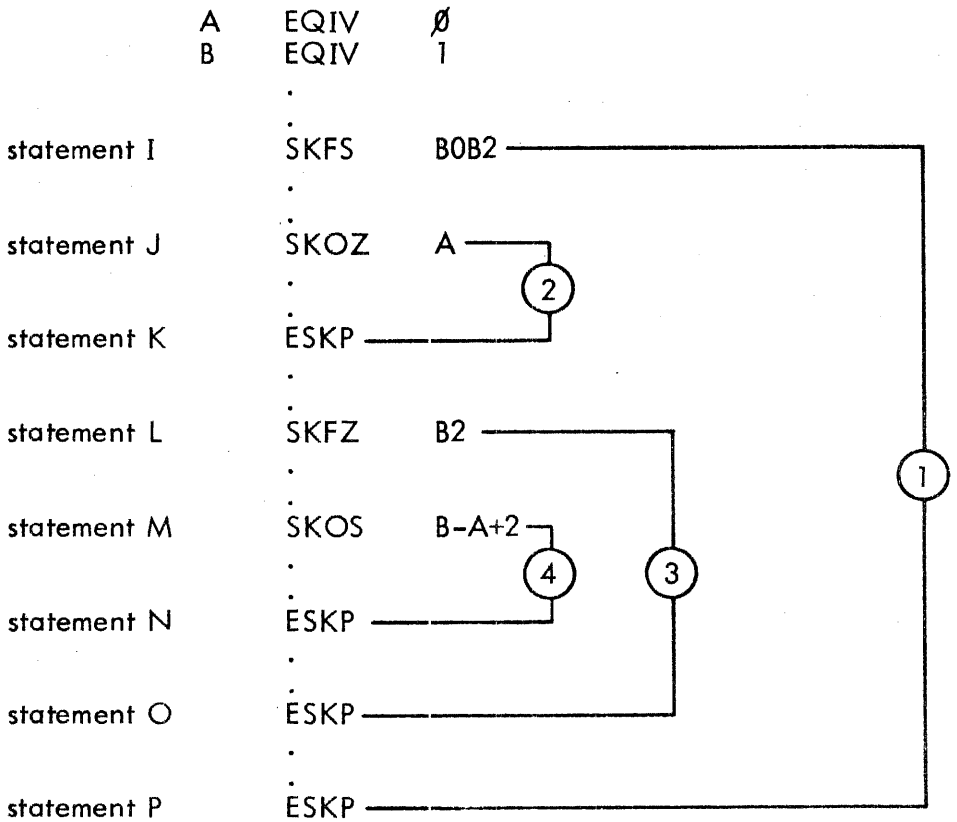
```
SKOP        4-3
SKON        3-4
SKOB
SKNB        $
SKOI        A(#,A(#
SKOD        ABC, ABCD
```

Examples of SKips that cause the code following them to be assembled.

```
SKOP        5-5
SKON        23
SKOB        Ø
SKNB
SKOI        A,B
SKOD        ,
```

Examples of "nested" Skip pseudo-operations:

Assume the system flags are set as follows: '00204631(i.e.,B0B3B4B7B8B11B16).



The preceding statements, considered individually, specify the following:

1) Statement I (SKFS) specifies that all statements between I and P are to be assembled;

2) Statement J (SKOZ) specifies that all statements between J and K are to be skipped;

3) Statement L (SKFZ) specifies that all statements between L and O are to be skipped;

4) Statement M (SKOS) specifies that all statements between M and N are to be assembled.

The next effect on the preceding examples, considered together, cause the following statements to be assembled:

1) Statements I + 1 through J - 1;

2) Statements K + 1 through L - 1; and

3) Statements O + 1 through P - 1.

Note that the SKOS at statement M is overridden by the SKFZ at statement L which is on a higher "nesting level".

4-22 END

The END pseudo-operation terminates assembly of the current program and reinitializes to accept another program or subprogram. If a label is present in the label field, it is assigned the address of the first available location following the literal table. If an optional operand (relative or absolute) is present in the operand field, its value will be presented to the link loader signifying the starting address of the program. This address will be passed to the operating system upon completion of the loading process. If no operand is present in the operand field, then the link loader will pass the starting load address of the program to the operating system as the starting address of the program.

4-23 END$

This pseudo-operation indicates that an End-of-File is to be written on the binary output logical device (B/O-5) and then backspaced over. This functioning permits the assembly to be followed by either a Link operation or other compilations, includes, etc. without manually having to write an EOF. In addition, the source output logical device (S/O-'10) is rewound (if the scratch option is set) and all I/O devices in use are closed. Exit is then made to the operating system. Labels or operands on the END$ statement are treated in the same manner as those encountered on an END pseudo-operation.

SECTION V
MACRO DESCRIPTION

5-1 SCOPE

The macro capability of the Series 6000 Assembler provides a powerful and useful method of defining instruction and data sequences at assembly time. The macros are user defined and provide flexibility in selectively generating machine code. The macro prototype and the calling parameters determine the machine code generated.

5-2 LANGUAGE SPECIFICATION

5-2.1 Basic Structure

A macro definition is of the form:

```
NAME    MACRO
        (Macro instructions)
        MEND
```

Macro calls are made by using the macro name as the op code, i.e.,

```
LABEL   NAME X,Y,Z,...
```

Example:

```
WAIT    MACRO
        TNK     '0700
        BLU     $I/O
        BON     *-2
        MEND
```

Then the programmer need only write the op code:

```
        WAIT
```

which will then be automatically expanded into the sequence:

```
        TNK     '0700
        BLU     $I/O
        BON     *-2
```

The macro is not a subroutine call. In-line code is generated when the macro is referenced. Macro definitions may appear anywhere in the program as long as they appear before the first reference to them.

5-2.2 Macro Parameters

Parameters on the Macro Call may be obtained by the Macro at the time of expansion by a structure of the form :DD, where DD is the parameter number. The label on the call to the Macro is available as :00; if no label is present, :00 has a null "value". In addition, the length (in characters) of a parameter may be obtained by a structure of the form $DD, where DD is as above. The maximum number of parameters in a single macro is 20. Note that parameter references are considered as character strings and may be concatenated with any character strings desired (see Example 3).

Example 1:

```
XYZ     MACRO
        TMA     :01
        AOA     :02
        TAM     :01
        MEND
```

then a call of the form

```
        XYZ     A,250
```

would be expanded into:

```
        TMA     A
        AOA     250
        TAM     A
```

Example 2:

```
ABC     MACRO
        TMA     :01
        AOA     $01
        MEND
```

then a call of the form

```
        ABC     ALPHA
```

would be expanded into:

```
        TMA     ALPHA
        AOA     05
```

Example 3:

```
MAC     MACRO
        TMA     A:01:02D,I
        AOA     10$01
        MEND
```

than a call of the form

```
        MAC     B,C
```

would be expanded into:

```
TMA     ABCD,I
AOA     1001
```

Since blanks and commas are used as delimiters, a special form has been implemented to allow blanks and/or commas within a parameter. When desired, a parameter can be enclosed within parentheses. When this is done, the actual parameter is assumed to be the character string beginning with the first character after the opening parenthesis and ending with the last character before the matching closing parenthesis.

Example 4:

```
R       MACRO
        TMI     :01
        TMJ     :02
        TMK     :03
        MEND
S       MACRO
        R       :01
        R       :02
        MEND
```

then a call of the form

```
        S       (A,B,C),(X,Y,Z)
```

would be expanded into:

```
TMI     A
TMJ     B
TMK     C
TMI     X
TMJ     Y
TMK     Z
```

5-2.3 Local Symbols

In order to avoid possible name conflicts, local names within Macros must be assured of being unique. To do this, a special label form is implemented. This of the form /XY, where XY consists of exactly two characters; X must be alphabetic and Y must be a character which is legal within a label. The generation number of the current Macro is appended after the pseudo-label to assure a unique label. The generation number of a Macro is a count of the number of macro expansions done within the program unit. If the generation number is less than 10000 then the pseudo-label is of the form XY/GGGG where GGGG is the generation number. If the generation number is greater than or equal to 10000 and less than 20000 then the pseudo-label is of the form X/YGGGG where GGGG is the generation number mod 10000. Creation of labels during generations greater than 19999 may cause erroneous results.

Example:

```
WAIT    MACRO
/WA     TNK     :0100
        BLU     $I/O
        BON     *-2
        MEND
```

than a sequence of calls of the form

```
        WAIT    '07
        WAIT    '05
                . . .
        (12345          intervening calls)
                . . .
        WAIT    '06
```

would be expanded into:

```
WA/0001 TNK     '0700
        BLU     $I/O
        BON     *-2
WA/0002 TNK     '0500
        BLU     $I/O
        BON     *-2
                . . .
        (12345          intervening macro references)
                . . .
W/A2348 TNK     '0600
        BLU     $I/O
        BON     *-2
```

5-2.4 Nested Macro Calls

Any Macro may, within its definition, call any other Macro, including itself. Thus a full recursive Macro structure is possible. Fifty levels of nesting and/or recursion are permitted.

Example:

```
FACTOR  MACRO
        SKOZ    :01
        MYO     :01
        FACTOR  :01-1,:02
        ESKP
        SKOS    :01
        TAM     :02
        ESKP
        MEND
```

than a call of the form

```
        FACTOR  3,X
```

would be expanded into:

```
        MYO     3
        MYO     3-1
        MYO     3-1-1
        TAM     X
```

Note that skips are not shown. This is because skips are not normally listed (see Section VI, Listing Options). For more involved listing of this macro call, see Paragraph 6-3.4.

5-2.5 Macro Library

A facility for keeping libraries of Macro definitions is part of the system. To reference a library, the pseudo-op MLIB is used. The operand field of the MLIB pseudo-op must contain an absolute expression whose value is a logical file number.

When the MLIB pseudo-op is encountered, the operand is evaluated. The operand is assumed to refer to a file containing source line images. This file is opened, rewound and then scanned for Macro definitions. If any Macro definitions are found, they are included as if they had been read from Symbolic Input. Any source images not part of a Macro definition (not between a MACRO and a MEND pseudo-op pair) are ignored. If a Macro definition is encountered that was previously defined via Symbolic Input, then the Macro definition in the library is ignored. When an End-of-File is detected, scanning stops and the file is closed.

SECTION VI
LISTING OPTIONS

6-1 SCOPE

This section describes the assembler facility for listing control.

6-2 LIST CONTROL STATEMENTS

To turn on the listing, columns 1 and 2 should contain an exclamation point and a plus sign, respectively (!+), and column 15 should be blank. This statement will not be listed; however, its presence will be retlected in the card sequence field.

To turn off the listing, columns 1 and 2 should contain an exclamation point and a minus sign, respectively (!-); and column 15 should be blank. This statement will not be listed; however, its presence will be reflected in the card sequence field.

Any errors detected during an assembly will be listed regardless of listing options. A non-zero error count will also be unconditionally listed. If the listing is off when the END or END$ pseudo-operation is encountered, the listing of the literal table is also suppressed.

There is no restriction on where or how often these cards may be included. Listing is initially on, so it is not necessary to precede an assembly with a list-on card to obtain a listing.

6-3 MACRO LISTING OPTIONS

In addition to the normal listing options, six listing options are available to aid in the readability of Macro expansions.

6-3.1 Options

NEST - When on, causes the listing of the Macro call itself (not the expanded code). This option only affects a Macro that has been called by a Macro, that is, a nested call.

MACRO - When on, causes the listing of the main Macro call (see NEST).

MEND - When on, causes the listing of all MEND's as part of the expansion.

CODE - When on, causes the listing of the expanded code produced by a Macro call.

MACDEF - When on, causes the listing of the Macro definition.

SKIP - When on, causes the listing of all SKip and ESKP pseudo-ops and any code that is SKipped; this only has an effect during the expansion of a macro.

6-3.2 Changing List Options

To turn on one or more options:

```
!+              OPT1,...,OPTN
Column 1        Column 15
```

To turn off one or more options:

```
!-              OPT1,000,OPTN
Column 1        Column 15
```

6-3.3 Default Options

The default options are:

```
NEST    - off
MACRO   - on
MEND    - off
CODE    - on
MACDEF  - on
SKIP    - off
```

6-3.4 Examples

The following is the list output of a sample run:

```
1                                       FACTOR  MACRO
2                                               SKOZ    :01
3                                               MYO     :01
4                                               FACTOR  :01-1,:02
5                                               ESKP
6                                               SKOS    :01
7                                               TAM     :02
8                                               ESKP
9                                               MEND
10                                              FACTOR  3,X
10      000000  60000003  0                     MYO     3
10      000001  60000002  0                     MYO     3-1
10      000002  60000001  0                     MYO     3-1-1
10      000003  15000010  1                     TAM     X
11                                      !+              NEST,SKIP,MEND
12                                              FACTOR  3,X
12                                              SKOZ    3
12      000004  60000003  0                     MYO     3
12                                              FACTOR  3-1,X
12                                              SKOZ    3-1
12      000005  60000002  0                     MYO     3-1
12                                              FACTOR  3-1-1,X
12                                              SKOZ    3-1-1
12      000006  60000001  0                     MYO     3-1-1
```

```
12                                                  FACTOR    3-1-1-1,X
12                                                  SKOZ      3-1-1-1
12                                                  MYO       3-1-1-1
12                                                  FACTOR    3-1-1-1-1,X
12                                                  ESKP
12                                                  SKOS      3-1-1-1
12      000007    15000010  1                       TAM       X
12                                                  ESKP
12                                                  MEND
12                                                  ESKP
12                                                  SKOS      3-1-1
12                                                  TAM       X
12                                                  ESKP
12                                                  MEND
12                                                  ESKP
12                                                  SKOS      3-1
12                                                  TAM       X
12                                                  ESKP
12                                                  MEND
12                                                  ESKP
12                                                  SKOS      3
12                                                  TAM       X
12                                                  ESKP
12                                                  MEND
13      000010    00000000  0         X               ***
14      000011    00400000  6                       END$
```

SECTION VII
ASSEMBLER OPTIONS AND OUTPUTS

7-1 SCOPE

This section describes the information supplied to the assembler by the operating system and the list output formats and error messages generated by the assembler.

7-2 ASSEMBLER OPTIONS

The assembler requests the following information from the operating system. This information is provided by the system service $INFO (refer to appropriate system reference).

1) Lines — specifies the total number of lines to be presented to the list output device prior to intiating a Top-of-Form request. Included is the heading line followed by two blank lines.

2) Date — a three-word (9-character) date placed on the heading line of each list output page.

3) Options — a 24-bit word indicating the following conditions:

   B0 - if set, the assembler will present all statements encountered on pass 1 to the source output device and will receive its input from this device on pass 2. (Scratch Option).

   B1 - is set, only pass 2 of the assembler will be executed. If this option is set, it is assumed that the symbol table is intact from a previous run. This will be true only if a normal assembly (i.e., option B1 reset) was previously completed, and there were no intervening processor executions or re-loading of the assembler prior to assembly with option B1 set. This option may not be used when the source input includes macro definitions or references.

7-3 ASSEMBLER OUTPUTS

7-3.1 List Output Formats

The general format of the list output line of the assembler is as follows:

AAAA BBBBBB CCCCCCCC LE DDDD....

where: AAAA = a 4-digit (decimal) card sequence number

BBBBBB = a 6-digit (octal) program location

CCCCCCCC = an 8-character (octal) binary output frame word or EQIV value

L = a single octal character loader code for the output frame word

E = a single alphabetic character error code (may be "U", "M", "O", "L", "P", "F", "C", or " ")

DDDD.... = the source input statement

There are certain assembler pseudo-operations where some of the above information is meaningless and is therefore suppressed. These pseudo-operations and their respective list output formats are as follows:

```
AAAA    CCCCCCCC   E    LABEL   EQIV    OPERAND
AAAA               E            FORM    OPERAND
AAAA               E            COMM    OPERAND
AAAA    CCCCCCCC   LE           XDEF    OPERAND
AAAA    CCCCCCCC   LE           XEQV    OPERAND
AAAA               E            SKip    OPERAND
AAAA               E            ESKP    OPERAND
AAAA                       *COMMENT CARD
```

7-3.2 Error Codes and Messages

The following is a list of the Series 6000 Macro Assembler error codes and messages.

1) M — signifies that the operand contains a reference to a label that has a multiple definition or the attempted redefinition or a macro label.

2) U — signifies that the operand contains a reference to a label that has not been defined.

3) O — indicates one of the following:

   a) operand syntax error
   b) improper index specification
   c) improper indirect specification
   d) improper literal usage
   e) improper text usage
   f) improper expression mode
   g) improper external specification
   h) operand address exceeds permissible limits

4) C — signifies that the contents of the operation field contains an unrecognizable operation code.

5) L — signifies the absence of a required label in the label field.

6) F — signifies incorrect card format (i.e., column 8 must be blank on all statements other than comment cards).

7) P — signifies a macro parameter error:

   a) greater than 20 parameters in a single call (the rest are ignored).
   b) expansion of a statement caused column 72 to be exceeded.

c) on a parameter enclosed in parentheses the terminal right parenthesis is not followed by a comma or a blank (following characters are ignored up to a comma or blank).

8) ASSEMB 1 —— indicates that insufficient storage is available for the assembler symbol table and that the job in progress will be aborted upon issuing a RELEASE command. (Each label encountered in the label field requires three locations of storage. Every two words of literal code requires three locations of storage. Macro definitions requires approximately six locations per line of source code.)

9) ASSEMB 2 —— indicates that the number of distinct common block names encountered has exceeded 25. (The number of common variables is unlimited.) Upon issuing a RELEASE command the job will be aborted.

10) ASSEMB 3 —— indicates that an operand error encountered in a SKip pseudo-operation. Upon issuing a RELEASE command the job will be aborted.

11) ASSEMB 4 —— indicates that an excess number of ESKP pseudo-operations were encountered. Upon issuing a RELEASE command the job will be aborted.

12) ASSEMB 5 —— indicates that an insufficient number of ESKP pseudo-operations were encountered upon completion of pass 1. Upon issuing a RELEASE command the job will be aborted.

13) ASSEMB 6 —— indicates that the expansion of a macro resulted in an overflow of the macro temporary storage area. (Nesting level is too deep or the total number of characters in the macro calling parameters is too great.)

APPENDIX A
SUPPLEMENTAL INFORMATION

Table A-1. Definition of Loader Codes

Bit Config.	Name	Function	Use
000	Direct Load	Load the word into the memory location specified by the RPL. Increment RPL by 1.	Direct loading of constants absolute memory referencing instructions, absolute address constants both 15-bit and 16-bit, and non-memory referencing instructions.
001	Memory Reference 15-bit address	Increment the right most 15-bits of the word by BASE and perform a direct load.	Relative memory referencing instructions.
010	External Definition	Increment the address by BASE and enter into external table along with the next 2 words as a 6-character ASCII name. Satisfy all requests.	External names, linkages between independently assembled programs.
011	External Request B0=0: 15-bit address B0=1: 16-bit address B1=0: unconditional request B1=1: conditional request	If defined, replace address with table address; if undefined, set up for stringing. Bit 0 of the load word specifies a 15-bit or 16-bit address. Bit 1 specifies whether conditional or unconditional request.	Memory reference to external address.
100	Memory Reference 16-bit address	Add MAPBIT to the word and execute memory reference (001).	Relative memory referencing for 16-bit address instructions and all address constants.
101	Common Request 15-bit address	Add common address to this address and execute direct load. Next 2 words contain common name.	Memory reference to location within a common block for internal and/or external use.
110	Special Action	Use bits 16 thru 20 of word to determine type of action to be taken.	Define absolute mode, relocatable mode, end, end jump absolute or relative, internal string back. (See special action table, A-2.)
111	Common Request 16-bit address	Add the 16-bit common address and execute direct load. Next 2 words contain common name.	16-bit memory reference to location within a common block

Table A-2. Link Loader Special Action Codes

Bit Config.	Name	Function	Use
00000	ORG absolute	Set RPL to the 16-bit address in this word.	Define or redefine absolute loading address.
00001	ORG relative	Set RPL to BASE + 16-bit address in this word.	Define or redefine relative loading address.
00010	END	Execute end of module procedures.	Define end of program module.
00011	End Jump Absolute	Set starting location to 15-bit address of this word and execute end of module procedures.	Define end of program module and starting address or programs.
00100	End Jump Relative	Set starting location to BASE + 15-bit address of this word then execute end of module procedures.	Define end of program module and starting address of programs
00101	String	Replace the address of location specified by this 15-bit address with the present contents of RPL. Replace this address with the former address of the location specified. If this address and the address of the location are equal, terminate this string. If not, re-execute string.	Forward referencing for one-pass processors.
00110	External String Back	If defined, perform string; if undefined, link strings.	Efficiency
00111	Module Name Definition	If this is the first link module, use the 6-character ASCII name contained in the next two load words as the name of this load module. Otherwise, ignore.	To define a module name to the system.
01000	Common Definition	Calculate common origin using address as size, enter into external table. Next 2 words contain the common block name.	Defining a common block for internal and external use.
01001	Common Origin	Set the RPL to the address in this word added to the address of the common block specified by the next 2 load words.	Loading block data into common.

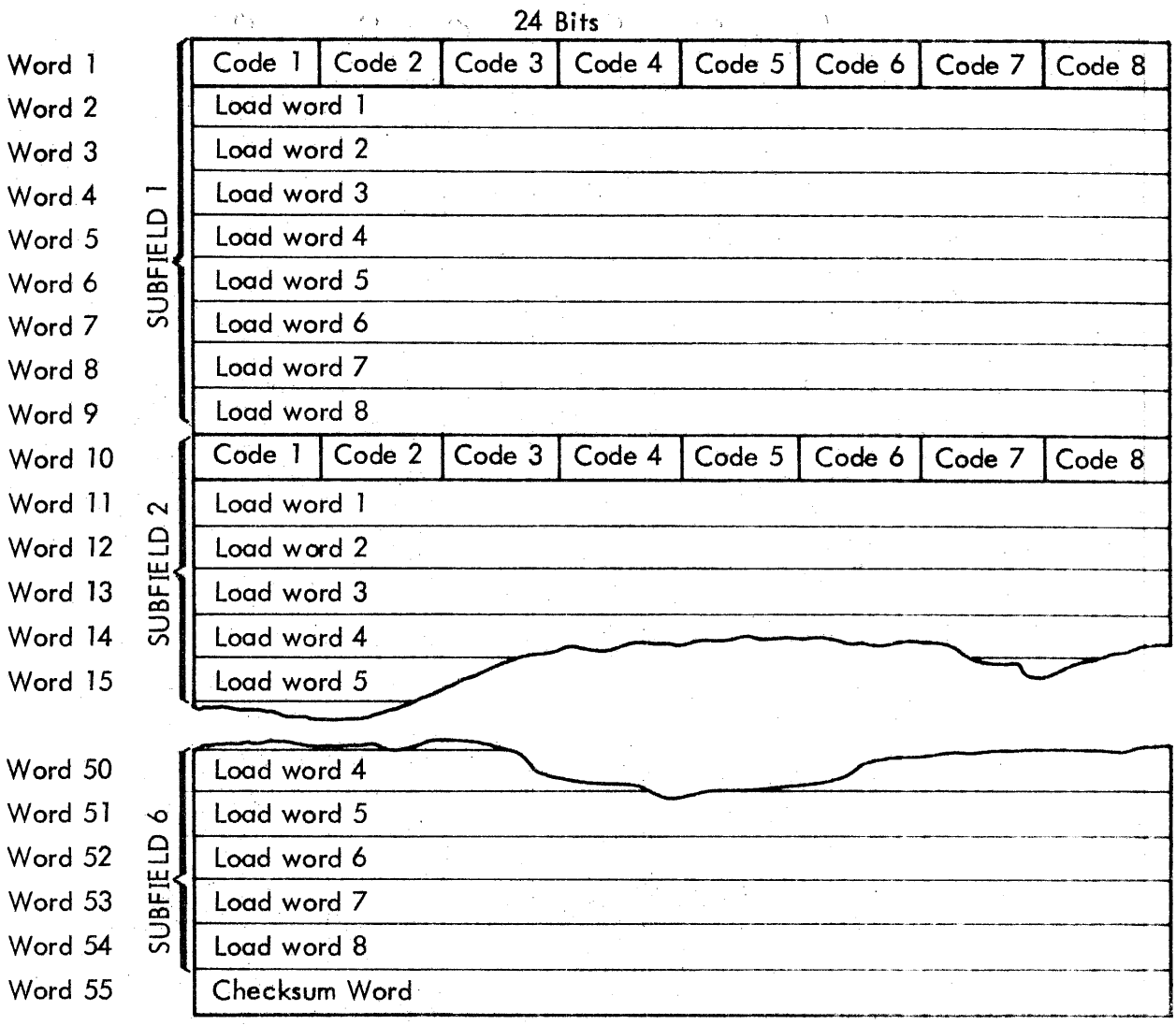
Table A-2. Link Loader Special Action Codes (Cont'd.)

Bit Config.	Name	Function	Use
01010	Source Error	Causes the Link Loader to abort loading.	To prevent loading and execution of programs that contain irrecoverable errors discovered by the Assembler.
01011	System Service Request	To provide linkage with system service routines.	If the External Request is found in the Link Loader's External Name table (i.e., has been previously defined), a BLJ instruction is inserted and the linkage is satisfied. If the External Request is not found in the Link Loader's table, the "System Service" table is scanned. If the External Request is found in the "System Service" table, a BLU instruction is inserted and the linkage is satisfied. If the External Request is not found in either table, a BLJ instruction is inserted and the Name is inserted in the Link Loader's External Name table. For the linkage to be satisifed, the external definition must now be loaded.
01101	External Equivalence Definition	The next two words contain a name, the third word contains a 24-bit constant. Enter into external table and set sign bit of first word of name to negative.	Defining a constant for external reference.
01110	External Equivalence Request	The next two words contain a name. The third word is an instruction frame and is ORed with the external table entry matching the name. A direct load is then performed.	Preparing an instruction, channel., Unit no., interrupt level, or group, all or part of which has been defined in another module.

Table A-3. Link Loader Input Record Format

Record length - 55 words, consisting of six 9-word subfields and a one-word hash-total checksum as the 55th word of the record.

The first word of each 9-word subfield contains eight 3-bit loader codes which determine the action to be taken for each of the following eight words in the subfield.



Tables A-4 through A-19 (located on the subsequent pages of this appendix) contain the permissible operand formats pertinent to the Series 6000 Macro Assembler.

*Table A-4.

15-Bit Positive Absolute or Relative Expression
External Requests (Conditional or Unconditional)
Common Requests
Index Specification
Indirect Specification

AAM, AEM, AUM, BUC, EXM, TBM, TDM, TRM, TXM, TrM

*Table A-5.

15-Bit Positive Absolute or Relative Expression
External Requests (Conditional or Unconditional)
Common Requests
Index Specification
Indirect Specification
Literal Address

AMA, AMB, AMD, AME, AMX
CMA, CMB, CME
DMA, DVM, DMX
IMA, IME
MYM, MMX
OMA
SMA, SMB, SMD, SME, SMX
TMR, TMB, TMD, TMX, TME, TMA, TMx
XMA

*Table A-6.

15-Bit Positive Absolute or Relative Expression
External Requests (Conditional or Unconditional)
Common Requests
Indirect Specification

AxM, BLx, CZM, HTx, RBM, TFM, TZM

*Table A-7.

15-Bit Positive Absolute or Relative Expression
External Requests (Conditional or Unconditional)
Common Requests
Indirect Specification
Literal Address

AMx, CMx, IMx, SMx, TMQ, EMB

*Refer to NOTES, page A-9.

*Table A-8

15-Bit Positive Absolute or Relative Expression External Requests (Conditional or Unconditional) Common Requests AOr, BWx, BNc, BOc, BcS, BcR, BOX, BBI, BBJ, BWx, DVO, MYO TOr, TNr

*Table A-9

16-Bit Positive Absolute or Relative Expression External Request (Conditional or Unconditional) Common Requests Indirect Specification BLL, BRL, BSL, BUL

*Table A-10

16-Bit Positive Absolute or Relative Expression External Request (Conditional or Unconditional) Common Request BJL

*Table A-11

16-Bit Positive Absolute or Relative Expression External Request (Conditional or Unconditional) Common Request Literal Address TLO

*Table A-12

8-Bit Absolute Expression External Request BLU

*Table A-13

11-Bit Absolute Expression (Channel Unit Specification) IAW, OAW, ISW, ODW

*Refer to NOTES, page A-9.

*Table A-14

11-Bit Absolute Expression (Channel Unit Specification) Indirect Specification (Merge, Override) IDW, OCW

*Table A-15

Unitary Bit Operands QBB, QSS

*Table A-16

8-Bit Absolute Positive Expression AOB, COB, DOB, OOB, SOB, TOB, XOB, TOC LRA, LRD, LLA, LLD, LAA, LAD RRA, RRD, RLA, RLD, RAA, RAD

*Table A-17

8-Bit Absolute Positive or Negative Expression AOM, AOW, AOX, COW, DOX, DV2, MOX, SOX, TOW, TOY, USP

*Table A-18

Double Operand (Bit Processor) First Operand: 5-Bit Absolute Expression less than or equal to 23 Second Operand: Blank or 8-Bit Positive Expression or: Single Operand: 13-bit absolute Expression DMH, DNH, FBM, OMH, ONH, QBM THM, TMH, XMH, XNH, ZBM

*Refer to NOTES, page A-9.

Table A-19

No Operand (Column 15 is Blank)

Arr, AAX, ADX
Crr, CDX, CZD, CZX, CZr
DAX, DDX, Drr, DVx, DVT
ESA, ESB, EZB
FAX, FXA, FNO
HIT, HLT, HSI, HXI
INX, Irr, IDX
Krr
MAX, MDX, MYr
NBB, NDD, Nrr, NSr, NXX, NHH, NOP
Orr
PBB, PDD, Prr, PXX
QBH
RCT, PPT, RSI, RXI, Rrr
SAX, SDX, SEX, SRE, SRT, SRX, Srr
TD1, TD2, TD3, TD4, TD5, TD6, T1D, T2D, T3D, T4D, T5D, T6D
TDL, TLD, TFH, TZH, TKV, TVK, TZD, TYA
Trr, TrB, TSr, TZr, TXD, TDX, TZX

UA1, UA2, UA3, UD1, UD2, UD3, UE1, UE2, UE3, UI1, UI2, UI3

Xrr

NOTES: (Tables A-4 thru A-19)

r - general register (I, J, K, E, A, or T)

x - index register (I, J, or K)

c - condition code (P, N, Z, or O)

External Equivalences are legal on all instructions except those with no operand.

Text is legal as an operand for most instructions, however, an attempt has been made to detect some instances where the use of text is meaningless. For example:

```
BSL         "A"
```