SANYO
ICON

ICON/PICK Assembly Language Manual

ICON INTERNATIONAL, Inc.

764 East Timpanogos Parkway
Orem, Utah 84057 - 6212
Telephone: 801 225-6888
Fax: 801 226-0651
Telex: 323938 ICONSYS

LANGUAGE MANUAL

ICON/PICK Assembly Language

SANYO ICON

ICON INTERNATIONAL, Inc.
764 East Timpanogos Parkway
Orem, Utah 84057-6212
Telephone: 801 225-6888
Fax: 801 226-0651
Telex: 323938 ICONSYS

ICON/PICK Assembly Language Manual
Edition A
September 1986

Icon Order Number: 172-041-001 (Manual Assembly, includes binder, tabs, and pages)
171-010-002 (Pages only)

Trademarks

Record of Changes

ICON/PICK Assembly Language Manual
Order No. 171-010-002

Date	Update†	Change	Entered By *
Sep. 1986	A0	Initial publication of Revision A	
Apr. 1989	A1	Changed Covers, Title Page, Disclaimer Page, and Record of Changes Page to include new corporate logo and correct copyright	

† An update number has two parts: a capital letter and an Arabic numeral. (See update number A0 above.) The capital letter refers to the revision of the manual and the Arabic numeral refers to the sequence of changes made to that particular revision.

The first publication of all manuals is always designated as Revision A and is presented as A0. After the number of changes made to a particular manual warrants a new edition, the revision letter is changed to the next capital letter. For example, the revision after Revision A will be Revision B, and the publication will be represented as B0.

The second part of the update number, the Arabic numeral, gives the consecutive order of changes made to each revision. Update number A1 represents the first change made to Revision A, update A2 is the second change, and so forth. When a new revision is issued, the numbering starts over (B0, B1, B2).

* The person who entered the updated pages into this manual.

TABLE OF CONTENTS

Chapter 1

THE ASSEMBLER

THE PICK SYSTEM

USER'S ASSEMBLY MANUAL

PICK SYSTEMS normally assumes responsibility for assuring the compatible coexistence of the total computer system. This is based on extensive planning and qualification testing of each component and of the integrated system. Because user written assembly code can bypass and disrupt normal software integrity controls, PICK SYSTEMS cannot ensure system integrity, compatibility, or performance once the user adds assembly language programs to the system as supplied by PICK SYSTEMS.

The PICK Virtual Assembly Language includes a wide range of very powerful constructs. It has many instructions designed specifically for data base management. There is an extensive software machine architecture that relies heavily on massive software conventions, because of which the virtual machine implementation is very efficient. This interprocessor dependence also creates a fragility in the system at the assembler code level. The inadvertant destruction of conventional interfaces can cause widespread damage to the integrity of the system software!!!

THIS MACHINE IS NOT WELL SUITED TO USER WRITTEN ASSEMBLY CODE!

User written assembly code is NOT SUPPORTED by PICK SYSTEMS. Time spent locating user problems that are found to be caused by user assembly code will be billed to the user!

1.1 PICK ASSEMBLER

The PICK Virtual Assembler is table-driven. It will translate an arbitrary source language into either another source language or into object code. The source item, or "mode" is an item in any file defined on the database. This mode can then be used to generate a formatted listing (using the MLIST verb) or can be loaded for execution (using the MLOAD verb). The file this item is in is dependent on the specific implementation.

1.2 SOURCE LANGUAGE

The source language accepted by the PICK Virtual assembler is a sequence of symbolic statements, one statement per source-item line. Each statement consists of a label field, an operation (or op-code) field, an operand field, and a comment field.

1.2.1 LABEL FIELD

The label field begins in column one of the source statement, and is terminated by the first blank or comma; there is no limit on its length. If the character "*" appears in the first column, the entire statement is treated as a comment, and is ignored by the assembler. The reserved characters *+-'= are the only ones that may not appear in the label field. An entry in this field is optional for all except a few opcodes. A label may not begin with a numeric character.

1.2.2 OPERATOR FIELD

The operator is the first non-blank field after either the initial blank or string of blanks, or after the blank or string of blanks after the label field. The operator string is called an op-code. Op-codes are pre-defined in the permanent op-code symbol file OSYM and consist of one or more alpha characters. Op-codes are usually mnemonics for the intended operation, either an assembly directive, an operation to be done by the target machine, or a macro which will expand into several primitive operators. Additionally, users may define new mnemonics or "macros" which expand into several machine instructions. This may be done by creating new entries in the OSYM file.

1.2.3 OPERAND FIELD

Operand field entries are optional, and vary in number according to the needs of the associated op-code. Entries are separated by commas and cannot contain embedded blanks (except for character string literals enclosed by single quotes). The operand field is terminated by the first blank encountered. The characters +-'* have special meaning in this field.

1.2.3.1 OPERAND FIELD EXPRESSIONS

Entries in the operand field may be a symbol, or a constant. A symbol is a string of characters that is either defined by a single label-field entry in the mode, or is an entry in the pre-defined permanent symbol file (PSYM). A constant may be one of the following forms:

*	-	Defines current value of the location counter.
N	-	(n decimal) - A decimal constant.
X'h'	-	(h hexadecimal) - A hexadecimal constant.
C'text'	-	Character string; any characters, including blanks and commas, may appear as part of "text"; a sequence of two single quotes ('') is used to represent one single quote in the text.

Arithmetic operators (+,-) may be used to combine two or more constants.

1.2.4 COMMENT FIELD

Any commentary information preceded by a blank may follow the operand field entries.

1.3 ASSEMBLING SOURCE CODE : 'AS' VERB

FORMAT:

```
AS filename itemname {(options}
```

The 'AS' verb will assemble the item in the file specified.

OPTION	MEANING
Q	specifies that error lines are not to be listed at the end of the assembly.
L	generate a listing (equivalent to the MLIST verb) during assembly.
P	routes listing to line-printer.

As the assembler processes, it will output an asterisk (*) as every ten source statements are assembled. At the end of pass-1 a new line is started and an asterisk is printed for each ten statements reassembled.

1.4 LISTING ASSEMBLY PROGRAMS : 'MLIST' VERB

FORMAT:

```
MLIST filename itemname {(options}
```

Options are separated by commas:

OPTION	MEANING
P	routes output to the line-printer.
M	prints macro-expansions of source statements.
E	prints error lines only; also suppresses the pagination and enters EDIT at the end of the listing.
S	suppress listing of the object code.
N-m	restricts listing to line numbers n through m inclusive

The listing is output with a statement number, location counter, object code and source code, with the label, op-code, operand and comment fields aligned. A page heading is output at the top of each new page.

Errors, if any, appear in the location counter/object code area; macro expansions appear as source code with the operation codes prefixed by a plus sign (+).

1.5 LOADING ASSEMBLED MODES : 'MLOAD' VERB

FORMAT:

```
-------------------------------------------------
|       MLOAD filename itemname {(options}      |
-------------------------------------------------
```

The assembled mode is loaded into the frame specified by the FRAME op-code statement.
If the load is successful, the message;

[216] MODE 'item-id' LOADED; FRAME = nnn SIZE = sss CKSUM = cccc

is returned, where

nnn is the 3-digit number of the frame into which the mode has been loaded. The number nnn is expressed in decimal.

sss is the number of bytes of object code loaded into the frame, expressed in hexadecimal (base 16) notation.

cccc is the byte check-sum for the object code in the loaded mode.

The mode will not load correctly if its size exceeds 512 bytes, or if a FRAME statement is not the first statement assembled in the mode. In either case, a message will be returned indicating the error.

1.6 VERIFYING A LOADED PROGRAM MODE : 'MVERIFY' VERB

FORMAT:

```
MVERIFY filename itemname {(options}
```

After assembling and loading a program, the verb MVERIFY is used to check the assembled program against the loaded program.

OPTION	MEANING
A	output columnar listing of all mismatches.
E	output errors only.
P	direct output to the printer.

EXAMPLES:

```
>MVERIFY SM EXAMPL1 [CR]

[217] MODE 'EXAMPL1' VERIFIED FRAME = 34 SIZE = 477

>MVERIFY SM EXAMPL2 [CR]

014 OC 18
[218] MODE 'EXAMPL2' HAS 1 BYTES OBJECT CODE MIS-MATCHES
```

The first example verifies, but the second does not. In Example #2, the system informs the user that one byte at byte address 14 should have a value of OC, not 18.

An "A" option will cause a columnar listing of all bytes which mismatch. Each value in the source file which mismatches will be listed, followed by the value in the executable frame.

EXAMPLE:

```
>MVERIFY SM EXAMPL3 (A) [CR]

LOC  XX  YY  LOC  XX  YY  LOC  XX  YY  LOC  XX  YY
014  OC  18  015  13  17  016  OE  OD  017  3A  3C
.
.

[218] MODE 'EXAMPL3' HAS 78 BYTES OBJECT CODE MIS-MATCHES
```

1.7 STRIPPING THE SOURCE CODE : 'STRIP-SOURCE' VERB

FORMAT:

```
---------------------------------------------------
|          STRIP-SOURCE filename item-list        |
---------------------------------------------------
```

The STRIP-SOURCE verb is used to remove the source code from Assembly Language programs. This frees large amounts of disc space back to the available space pool. Modes with source stripped out out can still be verified against the ABS.

After the verb has been invoked, the user is prompted with:

DESTINATION FILE:

The file-name where the stripped object code is to be stored should then be entered.

EXAMPLE:

```
>STRIP-SOURCE PROG * [CR]
DESTINATION FILE-SPROG [CR]
```

Here the file PROG containing source programs is stripped and copied to the file SPROG.

The first six lines of the source item will be copied without source code stripping. Standard Pick Systems convention for source modes has the "FRAME" statement in line 1, and other descriptive information in lines 2 through 6; this information is maintained through the STRIP-SOURCE process.

Chapter 2

MACHINE INSTRUCTIONS

THE PICK SYSTEM

USER'S ASSEMBLY MANUAL

2.1 PICK ASSEMBLY LANGUAGE

This section lists PICK machine instructions and describes their execution. For each assembler mnemonic, a list of the different permutations of the instruction is given.

Some assembly instructions are actually macros which expand to more than one opcode, and many of the instructions use elements not explicitly defined in the instruction. In particular, the accumulator and R15 are used by many of the macros.

In defining the op-codes the following set of symbolic operands are used:

SYMBOL	MEANING
b	BIT. A bit addressed relativly via a base register and a bit displacement.
c	CHARACTER. A byte addressed relatively via a base address register and displacement. (Also known as a CHR.)
d	DOUBLE-TALLY. A 4-byte field addressed relatively via a base register and displacement. (Also known as a DTLY.)
f	TRIPLE-TALLY. A 6-byte field addressed relatively via a base register and displacement. (Also known as a FTLY.)
h	HALF-TALLY. A 1-byte field addressed relatively via a base register and displacement. (Also known as a HTLY.)
l	LABEL. A label definition local to the current program frame.
m	MODE-ID. A 16-bit modal identificaton, comprised of a 4-bit entry point and a 12-bit frame number.
n	LITERAL. A literal or immediate value. The size of the assembled literal or value is dependent on the instruction in which the "n" is used.
r	ADDRESS-REGISTER. One of the sixteen Reality address registers (A/R's).
s	STORAGE REGISTER. A 6-byte field addressed relatively via a base register and a 16-bit word displacement.
t	TALLY. A 2-byte field relatively addressed via a base register and displacement. (Also known as a TLY.)

2.2 ARITHMETIC OPERATIONS

The following operations perform arithmetic on binary integers. Negative values are represented in two's complement form. One-byte and two-byte operands are sign extended to form a double word value before the operation is performed. The accumulator is a four-byte field (D0) for 1, 2 and 4-byte operands; the accumulator is a six-byte field (FP0) for 6-byte operands. Storage operands may not cross frame boundaries.

2.2.1 Load (LOAD)

LOAD d	LOAD f	LOAD h
LOAD m	LOAD n	LOAD t

The contents of the operand are loaded into the accumulator, with the high-order bit of the operand propagated left to fill the accumalator if necessary. One, two, and four byte operands are loaded into D0; 6-byte operands are loaded into FP0.

2.2.2 Load Extended (LOADX)

LOADX d	LOADX h	LOADX n
LOADX t		

The high-order bit (sign bit) of the operand is propagated left until there are 48 bits, which are loaded into the 6-byte accumulator (FP0).

2.2.3 Store (STORE)

STORE d	STORE f	STORE h
STORE s	STORE t	

The contents of the accumulator (H0, T0, D0 or FP0) replace the contents of the operand. The accumulator is not changed.

2.2.4 Zero (ZERO)

ZERO c	ZERO d	ZERO f
ZERO h	ZERO t	

The contents of the operand are replaced by zero.

2.2.5 One (ONE)

ONE d	ONE f	ONE h
ONE t		

The contents of the operand are replaced by a one.

2.2.6 Add to Accumulator (ADD)

ADD d	ADD f	ADD h
ADD n	ADD t	

The contents of the operand are added to the 4- or 6-byte accumulator. The result is placed into the accumulator.

2.2.7 Add Extended (ADDX)

ADDX d	ADDX h	ADDX n
ADDX t		

Same as for ADD, except that a 6-byte operand is generated by extending the sign bit of the original operand, and the result is in the 6-byte accumulator (FP0).

2.2.8 Increment Storage by One (INC)

INC d	INC f	INC h
INC t		

The contents of the operand are incremented by one.

2.2.9 Add to Storage (INC)

INC d,d	INC d,n	INC f,f
INC f,n	INC h,h	INC h,n
INC t,n	INC t,t	

The contents of the first operand are incremented by the contents of the second operand.

2.2.10 Subtract from Accumulator (SUB)

SUB d	SUB f	SUB h
SUB n	SUB t	

The contents of the operand are subtracted from the accumulator. The difference is placed into the accumulator.

2.2.11 Subtract Extended (SUBX)

SUBX d	SUBX h	SUBX n
SUBX t		

Same as for SUB, except that a 6-byte operand is generated by extending the sign bit of the original operand, and the result is in the 6-byte accumulator (FP0).

2.2.12 Decrement Storage by One (DEC)

DEC d	DEC f	DEC h
DEC t		

The contents of the operand are decremented by one.

2.2.13 Subtract from Storage (DEC)

DEC d,d	DEC d,n	DEC f,f
DEC f,n	DEC h,h	DEC h,n
DEC t,n	DEC t,t	

The contents of the first operand are decremented by the contents of the second operand.

2.2.14 Multiply (MUL)

MUL d	MUL f	MUL h
MUL n	MUL t	

The contents of the accumulator are multiplied by the operand. An 8-byte result is stored in the accumulator and accumulator extension (D0 and D1). The sign of the product is determined by the rules of algebra, that is, if the accumulator and the operand had the same sign before the operation, the result will be positive. Otherwise, the result will be negative.

2.2.15 Multiply Extended (MULX)

MULX d	MULX h	MULX n
MULX t		

Same as for MUL, except that a 6-byte operand is generated by extending the sign bit of the original operand.

2.2.16 Divide (DIV)

DIV d	DIV h	DIV n
DIV t		

The sign bit of the accumulator (D0) is extended into the accumulator extension (D1) to form a 64 bit dividend. The accumulator is then divided by the operand, forming a 32 bit quotient and a 32 bit remainder. The quotient replaces the contents of the accumulator and the remainder replaces the contents of the accumulator extension. The sign of the quotient is determined by the rules of algebra. The sign of the remainder is the sign of the dividend. The contents of the operand are not changed.

Note that the DIV instruction with a "f"-type operand is an extended divide; see next.

2.2.17 Divide Extended (DIVX)

```
DIVX d              DIVX f              DIVX h
DIVX t
```

Same as for DIV, except that a 6-byte operand is generated by extending the sign bit of the original operand; the result is in the 6-byte accumulator (FP0), and the remainder is in FPY.

2.2.18 Negate (NEG)

```
NEG d               NEG f               NEG h
NEG t
```

The sign of the operand is changed (two's complement form.)

2.2.19 Move (MOV)

```
MOV d,d             MOV e,e             MOV f,f
MOV h,h             MOV m,t             MOV n,d
MOV n,f             MOV n,h             MOV n,t
MOV t,t
```

These instructions move a 1- 2- 4- or 6-byte number from one location in storage to another.

2.3 CHARACTER INSTRUCTIONS

2.3.1 Move Character to Character (MCC)

```
MCC c,c          MCC c,r          MCC h,r
MCC n,c          MCC n,r          MOV r,c
MCC r,h          MCC r,r
```

The byte addressed by the first operand is moved to the byte addressed by the second operand.

2.3.2 Move Character to Incrementing Character (MCI)

```
MCI c,r          MCI n,r          MCI r,r
MCI s,r          MCI s,s
```

The second operand is incremented to point to the next byte in storage, and the byte addressed by the first operand is moved to the byte addressed by the second operand.

2.3.3 Move Character Incrementing and Count (MCI)

```
MCI n,r,d        MCI n,r,h        MCI n,r,n
MCI n,r,t
```

The second operand is incremented to point to the next byte in storage. The byte addressed by the first operand is moved to the byte pointer to by the second operand. This process continues until the number of bytes specified by the third operand has been moved. At least one byte is always used, and if the third operand is initially zero, 65,536 bytes will be moved. This instruction uses the accumulator.

2.3.4 Move Incrementing Character to Character (MIC)

```
MIC r,c          MIC r,h          MIC r,r
```

The first operand is incremented to point to the next byte in storage, and the byte then pointed to by the first operand is moved to the byte addressed by the second operand.

2.3.5 Move Incrementing Character to Incrementing Character (MII)

```
MII r,r
```

Both operands are incremented to point to the next byte in storage, then the byte pointed to by the first operand is moved to the byte pointed to by the second operand.

```
MII r,r,d        MII r,r,h        MII r,r,n
MII r,r,t
```

Identical to the operation above, with additional functionality. This process continues until the number of bytes specified by the third operand has been moved. If the third operand is initially zero, no data is moved. This instruction uses the accumulator.

2.4 LOGICAL INSTRUCTIONS

2.4.1 Logical Or (OR)

```
OR c,n          OR h,n          OR r,n
OR r,r
```

The byte in storage referenced by the first operand is logically or'ed with the mask byte referenced by the second operand. The byte referenced by the second operand is unchanged.

2.4.2 Logical Exclusive Or (XOR)

```
XOR c,n         XOR r,n         XOR r,r
```

The byte in storage referenced by the first operand is logically exclusive-or'ed with the mask byte referenced by the second operand. The byte referenced by the second operand is unchanged.

2.4.3 Logical And (AND)

```
AND c,n         AND r,n         AND r,r
```

The byte in storage referenced by the first operand is logically and'ed with the mask byte referenced by the second operand. The byte referenced by the second operand is unchanged.

2.4.4 Shift (SHIFT)

```
SHIFT r,r
```

The byte pointed to by the first operand is shifted right one bit. A zero (0) bit is shifted in on the left. The shifted byte replaces the byte pointed to by the second operand, or it replaces the original byte if only one operand is specified.

2.5 BRANCHING INSTRUCTIONS

2.5.1 Branch Unconditionally (B)

B l

A branch is taken to the label. The label must reside in the same program in the same frame as the branch instruction.

2.5.2 Enter External Mode (ENT)

ENT m

A branch is taken to the entry point specified by the mode-id. The high order 4 bits of the mode-id (m) are the entry point number (0-15). The remaining 12 bits of the mode-id are the FID of the frame to be branched to.

ENTI ENT* t

The ENTI* (Enter Indirect) instruction branches to the entry point defined by the low order 2 byte of the accumulator (T0).

ENT* branches to the entry point specified by the operand. The operand is loaded into T0, and an ENTI instruction is performed.

2.5.3 Subroutine Call (BSL)

BSL l BSL m

The BSL (Branch and Stack Location) instruction is used to program subroutine calls in assembly language.

The stack pointer (element RSCWA in the process' PCB) is incremented by 4, and the DEBUGGER is entered with a "RTN STK FULL" abort if the stack overflows. Otherwise, the address of the instruction following the BSL instruction, is moved to the 4-byte field in the process' PCB pointed to by the return stack pointer. Next, a branch is taken to the entry point (BSL m), or program label (BSL l).

BSLI BSL* t

BSLI executes a branch and stack location which branches to the entry point defined by the mode-id in the low order 2 bytes of the accumulator (T0).

BSL* executes a branch to the entry point specified by the operand. The operand is loaded into T0, and an BSLI instruction is performed.

2.5.4 Return from Subroutine (RTN)

```
RTN
```

A branch is made to the address stored in the last entry in the return stack, and the stack is popped one entry. The stack pointer (RSCWA) is decremented by 4, and if it underflows the stack, the DEBUGGER is entered with a "RTN STK EMPTY" abort.

2.5.5 Branch character instructions

All the branch character instructions perform a LOGICAL comparison on the two operands, that is, the bytes are treated as unsigned 8-bit fields rather than signed two's complement fields. Therefore, the lowest character in the range is X'00' and the highest is X'FF' (the segment mark).

2.5.6 Branch Character Equal (BCE)

```
BCE c,c,l        BCE c,r,l        BCE n,r,l
BCE r,c,l        BCE r,n,l        BCE r,r,l
```

The character (byte in storage) addressed by the first operand is compared with the character addressed by the second operand. If the two characters are equal, a branch is taken to the label specified by the third operand. The label must be inside the same frame as the BCE instruction.

2.5.7 Branch Character Unequal (BCU)

```
BCU c,c,l        BCU c,r,l        BCU n,r,l
BCU r,c,l        BCU r,n,l        BCU r,r,l
```

Same as BCE, except that the branch is taken if the two characters are unequal.

2.5.8 Branch Character Low (BCL)

```
BCL c,c,l        BCL c,r,l        BCL n,r,l
BCL r,c,l        BCL r,n,l        BCL r,r,l
```

The byte in storage referenced by the first operand is compared with the byte referenced by the second operand. Both bytes are treated as 8-bit unsigned numbers. If the byte addressed by the first operand is numerically less than the byte addressed by the second operand, a branch to the label specified by the third operand is taken. The label must be inside the same frame as the BCL instruction.

2.5.9 Branch Character Less than or Equal (BCLE)

```
BCLE c,c,l          BCLE c,r,l          BCLE n,r,l
BCLE r,c,l          BCLE r,n,l          BCLE r,r,l
```

Same as BCL, except that the branch is taken if the first operand is numerically less than or equal to the second operand.

2.5.10 Branch Character High (BCH)

```
BCH c,c,l           BCH c,r,l           BCH n,r,l
BCH r,c,l           BCH r,n,l           BCH r,r,l
```

Same as BCL, except that the branch is taken if the first operand is numerically greater than the second operand.

2.5.11 Branch Character High or Equal (BCHE)

```
BCHE c,c,l          BCHE c,r,l          BCHE n,r,l
BCHE r,c,l          BCHE r,n,l          BCHE r,r,l
```

Same as BCH, except that the branch is taken if the first operand is numerically higher than or equal to the second operand.

2.5.12 Branch Character Numeric (BCN)

```
BCN r,l
```

If the character pointed to by the register is numeric (i.e, between "0" and "9" inclusive,) then a branch is taken to the label, which must lie inside the same frame as the BCN instruction.

2.5.13 Branch Character Not Numeric (BCNN)

```
BCNN r,l
```

If the character pointed to by the register is not numeric, (i.e, not one of the characters 0, 1, 2, ... 9,) Then a branch is taken to the label, which must lie inside the same frame as the BCNN instruction.

2.5.14 Branch Character Hexadecimal (BCX)

```
BCX r,l
```

If the character pointed to by the register is hexadecimal, (i.e, in the range "0" - "9" inclusive or "A" - "F" inclusive,) then a branch is taken to the label, which must lie inside the same frame as the BCX instruction.

2.5.15 Branch Character Not Hexadecimal (BCNX)

```
BCNX r,l
```

If the character pointed to by the register is not hexadecimal, (i.e, outside the range "0" - "9" inclusive or "A" - "F" inclusive,) then a branch is taken to the label, which must lie inside the same frame as the BCNX instruction.

2.5.16 Branch Character Alphabetic (BCA)

```
BCA r,l
```

If the character pointed to by the register is alphabetic, (i.e, in the range of capital letters "A" - "Z" inclusive, or small letters "a" - "z" inclusive,) then a branch is taken to the label, which must lie inside the same frame as the BCA instruction.

2.5.17 Branch Character Not Alphabetic (BCNA)

```
BCNA r,l
```

If the character pointed to by the register is not alphabetic,, (i.e, outside the range "A" - "Z" inclusive or "a" - "z" inclusive,) then a branch is taken to the label, which must lie inside the same frame as the BCNA instruction.

2.5.18 Branch if Zero (BZ)

```
BZ c,l         BZ d,l         BZ f,l
BZ h,l         BZ s,l         BZ t,l
```

The branch is taken if the operand has a value of zero (0).

2.5.19 Branch if Not Zero (BNZ)

```
BNZ c,l        BNZ d,l        BNZ f,l
BNZ h,l        BNZ s,l        BNZ t,l
```

The branch is taken if the operand has any value other than zero (0).

2.5.20 Branch if Less than Zero (BLZ)

```
BLZ c,l        BLZ d,l        BLZ f,l
BLZ h,l        BLZ t,l
```

The branch is taken if the operand has a negative value.

2.5.21 Branch if Less than or Equal Zero (BLEZ)

```
BLEZ c,l       BLEZ d,l       BLEZ f,l
BLEZ h,l       BLEZ t,l
```

The branch is taken if the operand has a negative or zero (0) value.

2.5.22 Branch if Equal (BE)

```
BE d,d,l        BE d,n,l        BE f,f,l
BE f,n,l        BE h,h,l        BE h,n,l
BE n,d,l        BE n,f,l        BE n,h,l
BE n,t,l        BE t,n,l        BE t,t,l

BE m,t,l        BE s,s,l        BE t,m,l
```

The branch to the label is taken if the two operands contain the same number. The contents of both operands are treated as two's complement integers. If the operands are of the same size, and are identical, then the branch is taken. Otherwise, the sign bit (highest-order bit) of the smaller operand is extended to the left until the operands are the same size, and if the two equal size numbers are identical, then the branch is taken.

2.5.23 Branch if Unequal (BU)

```
BU d,d,l        BU d,n,l        BU f,f,l
BU f,n,l        BU h,h,l        BU h,n,l
BU n,d,l        BU n,f,l        BU n,h,l
BU n,t,l        BU t,n,l        BU t,t,l

BU m,t,l        BU t,n,l        BU t,t,l
```

The branch to the label is taken if the two operands contain different numbers. Smaller operands will be sign extended, as with BE.

2.5.24 Branch if Less than (BL)

```
BL d,d,l        BL d,n,l        BL f,f,l
BL f,n,l        BL h,h,l        BL h,n,l
BL n,d,l        BL n,f,l        BL n,h,l
BL n,t,l        BL t,n,l        BL t,t,l
```

The contents of both operands are treated as two's complement integers. The branch is taken if the number contained in the first operand is less than the number in the second operand.

2.5.25 Branch if Less than or Equal (BLE)

```
BLE d,d,l       BLE d,n,l       BLE f,f,l
BLE f,n,l       BLE h,h,l       BLE h,n,l
BLE n,d,l       BLE n,f,l       BLE n,h,l
BLE n,t,l       BLE t,n,l       BLE t,t,l
```

The contents of both operands are treated as two's complement integers. Smaller operands will be sign extended to match the size of larger operands. If the first number is less than or equal to the second number, a branch is taken to the label.

2.5.26 Branch if High (BH)

BH d,d,l	BH d,n,l	BH f,f,l
BH f,n,l	BH h,h,l	BH h,n,l
BH n,d,l	BH n,f,l	BH n,h,l
BH n,t,l	BH t,n,l	BH t,t,l

A branch is taken to the label if the number contained in the first operand is higher than the number contained in the second operand. Both numbers are treated as two's complement integers.

2.5.27 Branch if High or Equal (BHE)

BHE d,d,l	BHE d,n,l	BHE f,f,l
BHE f,n,l	BHE h,h,l	BHE h,n,l
BHE n,d,l	BHE n,f,l	BHE n,h,l
BHE n,t,l	BHE t,n,l	BHE t,t,l

A branch to the label is taken if the number in the first operand is higher than or equal to the number in the second operand. Both numbers are treated as two's complement integers.

2.5.28 Branch Decrementing Not Zero (BDNZ)

BDNZ d,l	BDNZ d,d,l	BDNZ d,n,l
BDNZ f,l	BDNZ f,f,l	BDNZ f,n,l
BDNZ h,l	BDNZ h,h,l	BDNZ h,n,l
BDNZ t,l	BDNZ t,t,l	BDNZ t,n,l

The first operand is decremented by one, or by the second operand if there are three operands. If the first operand is non-zero, then a branch is taken to the label.

2.5.29 Branch Decrementing Less than Zero (BDLZ)

BDLZ d,l	BDLZ d,d,l	BDLZ d,n,l
BDLZ f,l	BDLZ f,f,l	BDLZ f,n,l
BDLZ h,l	BDLZ h,h,l	BDLZ h,n,l
BDLZ t,l	BDLZ t,t,l	BDLZ t,n,l

The first operand is decremented by one, or by the second operand if there are three operands. If the first operand is decremented below zero (0), then a branch is taken to the label.

2.5.30 Branch Decrementing Less than or Equal Zero (BDLEZ)

BDLEZ d,l	BDLEZ d,d,l	BDLEZ d,n,l
BDLEZ f,l	BDLEZ f,f,l	BDLEZ f,n,l
BDLEZ h,l	BDLEZ h,h,l	BDLEZ h,n,l
BDLEZ t,l	BDLEZ t,t,l	BDLEZ t,n,l

The first operand is decremented by one, or by the second operand if there are three operands. If the first operand is decremented to or below zero (0), then a branch is taken to the label.

A string is a series of logically continuous characters in storage, which may extend over linked frame boundaries. String instructions can scan or move strings of variable length. Crossing of frame boundaries and attaching and detaching of registers used in string instructions is handled automatically and is transparent to the user.

Note that in the event that any of these instructions reaches an end of linked frame condition, there is a special tally called XMODE that may be used to intercept this exception condition and perform special processing. Usage of XMODE is discussed in the section SYSTEM SOFTWARE. If XMODE is zero when an end or beginning of linked frame set is reached, a trap to the DEBUGGER is executed resulting in a FORWARD LINK ZERO abort message.

Some of the string instructions contain an extra literal byte known as a "variant." The variant byte controls the byte-by-byte matching against preset delimiters. The format of the variant byte (for all instructions except SICD) is as follows:

BIT	MEANING
0 (Most significant)	1 = Stop on Match 0 = Stop on Mismatch
1	Compare with X'FF' (SM)
2	Compare with X'FE' (AM)
3	Compare with X'FD' (VM)
4	Compare with X'FC' (SVM)
5	Compare with character in SC0
6	Compare with character in SC1
7 (Least significant)	Compare with character in SC2

The most significant bit determines whether the instruction stops on a "match" condition (bit is set to "1"), or on a "mismatch" condition (bit is "0"). Only those characters whose corresponding bits (see table above) are set are checked to determine a match or mismatch. The first four characters are the system delimiters; the last three characters are variable and reside in the user's PCB.

Below are examples of variant bytes and their respective match conditions:

VARIANT	CONDITION
X'A0'	Stop on attribute mark (X'FE')
X'F0'	Stop on SM, AM or VM
X'01'	Stop on non-blank (If there is a blank in SC2)
X'A4'	Stop on AM or contents of SC0

2.6.1 Scan to Delimiter

SID r,n

This instruction is used to find the end of a string, or to scan a string to find the first or last occurrence of a character in the string. The register (r) is incremented to point to the next character (byte) in storage, and the byte pointed to is checked for a match using the variant byte (n). The scan continues until a match or mismatch condition, as defined by the variant, is reached. Note that the this instruction will alter the position of the register by at least one location.

2.6.2 Scan to Delimiter and Count

SIDC r,n

This instruction scans a string from a register to a delimiter, and keep a count of the number of bytes scanned. The register is incremented to point to the next byte in storage, the lower-order 2 bytes of the accumulator (T0) are decremented one, and the byte addressed by the register is checked for a match or mismatch condition as defined by the literal variant byte. The process continues until a match condition is met, at which time the number of bytes scanned is the difference between the value of T0 before and after the instruction. Note that this instruction will alter the position of the register by at least one location.

2.6.3 Scan to Count

SIT r

This instruction scans the register forward the number of bytes specified by the contents of T0. The register is incremented and T0 is decremented until T0 reaches 0.

This instruction is logically equivalent to the instruction "INC r,T0" ; however, the SIT instruction can be used to force usage of exception mode processing via XMODE (see SYSTEM SOFTWARE for XMODE usage) if it reaches the end of a linked frame set. If T0 is zero at the start of the instruction, it becomes a NO-OP and the register is not altered.

2.6.4 Scan to Count or Delimiter

SITD r,n

This instruction combines the functions of the SIT and SID, in that the string is scanned until EITHER a match condition, as determined by the variant byte, is reached, OR the count in T0 reaches zero. If the instruction terminates due to the match condtion being met, the difference in the ending and original values of T0 gives the number of bytes scanned. If T0 is zero at the start of the instruction, it becomes a NO-OP and the register is not altered.

2.6.5 Move String to Delimiter

MIID r,r,n

This instruction is generally used to move a string pointed to by a register up to and including the delimiter marking the other end of the string. Both registers are incremented by one, and the byte pointed to by the first register is moved to the location addressed by the second register. The byte moved is then checked for a match, using the variant byte. The process of incrementing, moving and checking continues until a match condition occurs. Note that this instruction will alter the position of the registers by at least one location.

2.6.6 Move string to Delimiter and Count

MIIDC r,r,n

This instruction moves a string from one register to the other up to a delimiter, and keeps a count of the number of bytes scanned. Both registers are incremented by one, and the byte addressed by the first is moved to the location pointed to by the second; T0 is decremented by one. The byte moved is the checked for a match, using the variant byte. This process is repeated until a match occurs. The number of bytes moved is the difference between the original value of T0 and its value at the termination of the instruction. Note that this instruction will alter the position of the registers by at least one location.

2.6.7 Move String to Count

MIIT r,r

This instruction is used to move a string of fixed length. T0 contains a byte count (up to 65,535) defining the number of bytes to be moved. If T0 is zero when the instruction is executed, no operation is performed. Otherwise, the registers are incremented by one, the byte addressed by the first register is moved to the byte addressed by the second register, and T0 is decremented by one. This process is repeated until T0 reaches zero.

2.6.8 Move String to Register

MIIR: r,r

This instruction is used to move a string between the first register and R15 to the location addressed by by the second register. The first register is checked against R15, and if they are equal, the instruction ends. Otherwise, the registers are both incremented to point to the next byte in storage, and the byte pointed to by the first register is moved to the byte pointed to by the second register. The first register is then checked against R15, and the cycle of compare, increment, and move is repeated until the first register and R15 are equal. Note that if R15 is not forward of and in the same string as the first register, this instruction will not terminate.

2.6.9 Move String to Count or Delimiter

MIITD r,r,n

This instruction combines the functions of the MIID and MIIT instructions. Both registers are incremented and a byte is moved from the first to the second register. The lower 2 bytes of the accumulator (T0) are decremented by one. If EITHER the byte moved matches a delimiter, as defined by the variant byte, OR if T0 is decremented to 0, the instruction terminates. If T0 is zero at the start of the instruction, it becomes a NO-OP and the register is not altered.

2.6.10 Scan, Counting Delimiters (SICD)

SICD r,n

This instruction can scan a variable number of delimiters.

The function of the instruction is to position the register at a specified point within a data structure containing several levels of delimiters in minimial number of instructions. To accomplish this, the register pointing to the scanned position is adjusted dependent upon the termination mode of the instruction, i.e. The register is decremented if the instruction terminates in the abnormal mode.

The low order 16 bits of the accumulator (T0) contain the delimiter count. The referenced register points to the byte preceeding where the scan is to be started. The variant byte specifies the scan mode and the termination criteria. The scan will unconditionally stop on a X'FF' character.

Variant byte functions:

BIT	MEANING
0	Bit set if count is to be decremented before instruction is started. This form is for ordinal positioning. I.e. in BASIC the first attribute within a dynamic array (e.g. EXTRACT(ITEM,1,0,0) is logically the beginning of the string.
1	Bit is zero if scan is to be terminated when a character is found which is greater than the delimiter. This format is used when scanning for system level delimiters. Logical character compares are used, i.e. X'FE' is > X'20'. If bit is set, scan to be terminated only when a character is found which is greater than the character contained in SC2. Note: if the delimiter character is also SC2 the state of this bit is not significant.
2	Scan delimiter is X'FE'
3	Scan delimiter is X'FD'.
4	Scan delimiter is X'FC'.
5	Scan delimiter is contained in SC0.
6	Scan delimiter is contained in SC1.
7	Scan delimiter is contained in SC2. See bit 1 above.

NOTE: If more than one scan delimiter is specified, the delimiter associated with the highest numbered bit will be used.

Upon termination of the instruction:

Normal: the count in T0 will be zero designating that the specified number of delimiters have been counted. The register is positioned on the delimiter. If the initial count is zero (or one with bit 0 set) the instruction will return immediately.

Abnormal: the count in T0 is decremented for each delimiter found. The count remaining in T0 will be the number of delimiters which must be inserted to create the logical data position. The register pointing at the data position is decremented by 1 byte, thus preparing for any subsquent string positioning commands. It should be noted that this convention allows multiple positioning commands to be executed without testing to determine if a data element is null, that is assuming that the element delimiters have a monotonic relationship.

Examples:

The following structure is used for discussion...

```
_E0^E11]E12^E2^E31]E321\E322]E4_
^  ^        ^  ^                 ^
|Ra|        |  |                 |re
   |Rb      |  |rd
            |Rc
```

Case 1 - Scan to attribute 3 - ENGLISH interface
R15 is positioned at Ra

```
LOAD   3             AMC COUNT
SICD   R15,X'20'     SCAN TO AM DELIMITER
```

At completion R15 will be positioned to Rd, and T0 = 0

CASE 2 - Scan to attribute 6 - BASIC interface
R15 is positioned at Rb

```
LOAD   6             AMC COUNT
SICD   R15,X'A0'     SCAN TO AM DELIMITER
```

At completion R15 will be positioned to Re, and T0 = 2

CASE 3 - Scan to attribute 3 / value 2 / subvalue 1 - ENGLISH interface

```
LOAD   3             AMC COUNT
SICD   R15,X'20'     SCAN TO AM DELIMITER
LOAD   2             VALUE POSITION
SICD   R15,X'90'     SCAN TO VM DELIMITER
LOAD   1             SUBVALUE POSITION
SICD   R15,X'88'     SCAN TO SVM DELIMITER
```

At completion R15 will be positioned to Rd, and T0 = 0


```
CASE 4  -  Scan to 10'th occurance of character in SC1;
           stop on any character which is
           greater than the character in SC2.
           (No data shown for this example.)

      LOAD   10
      SICD   R15,X'42'
```

2.6.11 Branch on comparing strings; BSTE and BSTU

```
BSTE r,r,n,l        BSTU r,r,n,l
```

This instruction compares two strings up to a delimiter, and execute the branch if the strings are equal. The function of the variant byte is to specify a lower boundary for the delimiter that is considered to terminate the strings, that is, any character that is found to be logically greater than or equal to the variant byte is considered to terminate the string. Note that the strings do NOT have to be terminated by the same delimiter!

Both registers are incremented by one, and the bytes addressed by them are compared logically. If the bytes are equal, AND if the bytes are logically lower than the variant byte specified in the instruction, the increment and comparison is repeated. If the bytes are unequal, AND both bytes are greater than or equal to the variant byte, the strings are considered equal, and the instruction terminates by taking the branch.

In other cases, the strings are considered unequal, and the instruction terminates by falling through to the next sequential instruction.

Note that a three-way branch (equal, low, high) condition on comparing two strings can be coded by following, for example, the BSTE instruction by a suitable BCL instruction such as:

```
      BSTE  R4,R5,X'FC',EQUAL
      BCL   R5,R4,LOW
HIGH  EQU   *
```

2.7 BIT INSTRUCTIONS

2.7.1 Set Bit (SB)

```
SB b
```

The referenced bit is set to an "on" (1 or true) condition.

2.7.2 Zero Bit (ZB)

```
ZB b
```

The referenced bit is set to an "off" (0 or false) condition.

2.7.3 Branch Bit Set (BBS)

```
BBS b,l
```

If the referenced bit is "on" (1), then a branch is taken to the label.

2.7.4 Branch Bit Zero (BBZ)

```
BBZ b,l
```

If the referenced bit is "off" (0), then a branch is taken to the label.

2.8 REGISTER INSTRUCTIONS

2.8.1 Load Absolute Difference (LAD)

LAD r,r　　LAD r,s　　LAD s,r
LAD s,s

This instruction computes the number of bytes between the byte in storage pointed to by the first operand and the byte pointed to by the second operand. The result is a non-negative integer in the low order 2 bytes of the accumulator (T0).

NOTE: This instruction is valid for unlinked frames only if the frames referenced by the two arguments are the same. The instruction is valid for unequal frame numbers only if both frames are in the same group of contiguously linked frames, and the difference between the frame numbers is less than 32.

2.8.2 Increment Address Register (INC)

INC r

The address register is incremented by one causing it to point to the next sequential byte. If the resulting address is not in the same buffer, then either:

A crossing frame limits error occurs if the register is in unlinked format, or

An attempt is made to attach the register to the first data byte of the frame pointed to by the forward link of the current frame. In this case, forward link zero and illegal frame id are errors which can be detected if they occur.

INC r,n　　INC r,t

The address register is incremented by n or the number in the tally. If the increment causes the register to cross a frame boundary, then crossing frame limit, forward link zero or illegal frame id will be reported as appropriate.

2.8.3 Decrement Address Register (DEC)

DEC r

The address of the register is decremented by one.

If the register is in linked format and originally pointed to the first data byte of the frame and the backward link of the current frame is zero, the register attaches to data byte zero of the current frame. Otherwise, an attempt is made to attach the register to the last data byte of the frame pointed to by the backward link of the current frame. Illegal frame id is an error which can be detected in this case.

DEC r,n　　DEC r,t

Same as the INC instruction, except that the second operand is subtracted from the register address.

2.8.4 Increment Storage Register (INC)

INC s   INC s,n   INC s,t

The displacement portion of the storage register is incremented by one, or by the two's complement integer contained in the second operand. Note that no address errors are detectable.

2.8.5 Decrement Storage Register (DEC)

DEC s   DEC s,n   DEC s,t

The displacement portion of the storage register is decremented by one, or by the two's complement integer contained in the second operand.

2.8.6 Set Register to Address (SRA)

SRA r,c   SRA r,d   SRA r,f
SRA r,h   SRA r,l   SRA r,s
SRA r,t

The register is set pointing to the first byte of the second operand.

2.8.7 Move Register to Register (MOV)

MOV r,r

The first operand replaces the second operand. All eight (8) bytes of the register are copied.

MOV r,s

The effective register of the A/R replaces the contents of the S/R. The A/R is not affected.

MOV s,r

The contents of the S/R replace the A/R. If the S/R is not legal, address errors may be detected at this time.

MOV s,s

The contents of the first S/R replace the contents of the second S/R. No address errrors are detectable.

2.8.8 Exchange Register with Register (XRR)

XRR r,r

The contents of the two registers are interchanged. All eight (8) bytes from each operand are copied to the other operand.

XRR r,s   XRR s,r   XRR s,s

These instructions expand into macros which use R15 and MOV instructions.

2.8.9 Setup Register (SETUP)

SETUP r,t,d   SETUP0 r,d   SETUP1 r,d

The setup instruction is similar to the move storage register to address register instruction. The operand one address register is 'setup' to the implied storage register with the second operand as a displacement and the third operand as a frame-id (FID).

If the SETUP0 or SETUP1 form is used, the S/R displacement is set to zero or one.

2.9 CONVERSION INSTRUCTIONS

Conversion operations are provided to convert decimal integers represented by ASCII characters into binary values, and to convert hexadecimal integers into binary values, and binary values to hexadecimal. All conversions involve a register string pointer. Similar to other string functions, this register points one byte before the string.

2.9.1 Move Binary to Decimal (MBD)

```
MBD d,r            MBD f,r            MBD h,r
MBD t,r            MBD n,d,r          MBD n,f,r
MBD n,h,r          MBD n,t,r
```

The binary integer in the first operand is converted to an ASCII string and stored starting one byte past the byte pointed to by the register. If only two operands are present, MBD creates a variable length string, storing only the significant digits. If the third operand (n) is specified, it contains the number of characters to be put in the string. The number will be padded on the left with blanks if necessary, and will make the string longer than n characters if necessary.

2.9.2 Move Binary to Hexadecimal (MBX and MBXN)

```
MBX c,r            MBX d,r            MBX f,r
MBX h,r            MBX s,r            MBX t,r
MBX n,d,r          MBX n,f,r          MBX n,h,r
MBX n,s,r          MBX n,t,r

MBXN n,d,r         MBXN n,f,r         MBXN n,h,r
MBXN n,s,r         MBXN n,t,r
```

MBX is used to output an ASCII string representing a hexadecimal number. The MBX instruction assumes that the low order byte of the accumulator (H0) contains the count of the number of characters to be output. Bit B7 (high order bit of H0) is set if the string is to be padded with leading zeroes. If the third parameter (n) is present, the instruction expands into a macro. The macro first loads the number n into H0, and sets B7 if the opcodes was MBXN.

2.9.3 Move Decimal to Binary (MDB)

```
MDB r,d            MDB r,f            MDB r,h
MDB r,t
```

The ASCII decimal character pointed to by the register is converted to a binary number and stored into the second operand. The second operand is multiplied by ten (10) and the binary equivalent of the number pointed to by the register is added to the second operand.

2.9.4 Move Hexadecimal to Binary (MXB)

MXB r,c	MXB r,d	MXB r,f
MXB r,h	MXB r,s	MXB r,t

The ASCII hexadecimal character pointed to by the register is converted to a binary number and stored into the second operand. The second operand is multiplied by sixteen (16) and the binary equivalent of the number pointed to by the register is added to the second operand.

2.9.5 Move Floating-Point String to Binary (MSDB and MSXB)

MSDB r	MSXB r

MSDB converts the signed floating point decimal string pointed to by the register to a 6-byte binary integer, scales the number up by SCALE (in the user's PCB,) and stores the signed integer result in the 6-byte accumulator (FP0). MSXB is identical to MSDB, except that it converts hexadecimal numbers.

Both these instructions are macros which first zero D0 and D1, then execute a MFD: (MSDB) or MFX: (MSXB) instruction. These instructions (MFD: and MFX:) require that: H7 contains the fractional digit count (0-15) in its low order 4 bits, the high order 4 bits of H7 are as follows: 0) unused 1) numeric found 2) you passed a decimal point 3) sign bit. H6 contains the integer digit count. The register points one byte before the string to be converted. FP0 is normally zeroed before using these instructions, since any value in FP0 will be multiplied by 10 (MSDB) or 16 (MSXB) each time a character is converted.

The string must be at least one digit long, and must be terminated by a system delimiter (X'FA' -- X'FF'). It may not contain more than one decimal point, more fractional digits than are specified in H6, or any non-numeric (MSDB) or non-hex (MSXB) characters. A leading plus sign (+) or minus sign (-) is legal, and the result in FP0 will be negative if the string started with a minus sign. If the required number of fractional digits are not present, FP0 will be scaled upward as necessary

After conversion, the register points to the system delimiter at the end of the string, and NUMBIT is set to one (1), unless any of the above conditions are violated, in which case the register points to the last character converted, and NUMBIT is zero (0).

During execution of the instruction, H6 is decremented by one for each digit found; if H6 goes to zero, the instruction is terminated, with the register pointing to the last character converted, and NUMBIT set to zero (0). In this case, the fractional digit count is ignored.

2.10 OTHER INSTRUCTIONS

The following operations are used to communicate with the MONITOR.

2.10.1 Read Input Queue (READ)

READ r

The next character from the terminal input queue replaces the byte addressed by the register. If the input queue is empty the process is suspended until a character is received from the terminal. Characters transmitted by the terminal are automatically queued in the PIB for the terminal.

2.10.2 Write to Output Queue (WRITE)

WRITE r

The byte addressed by the register is placed into the terminal output queue. If the queue is full, the process is suspended until there is room in the queue.

2.10.3 Release Time Quantum (RQM)

RQM

Upon execution of this instruction, the process gets de-activated and the next process is selected. This process will be reactivated after a small delay. The instruction is useful when you need to wait a short period for some external activity.

Chapter 3

SUPPORT SOFTWARE

THE PICK SYSTEM

USER'S ASSEMBLY MANUAL

3.1 SYSTEM SOFTWARE

3.1.1 Introduction

Assembly level programming in the PICK system is facilitated by a set of system subroutines that allow easy interaction with the disc file structure, terminal i/o, and other subroutines. These subroutines work with a standard set of addressing registers, storage registers, tallies, character registers, bits, and buffer pointers, collectively called "functional elements." In order to use any of these routines, therefore, it is essential that the calling routine set up the appropriate functional elements as required by the called routine's input interface.

The standard set of functional elements is pre-defined in the permanent symbol file (PSYM), and is therefore always available to the programmer. Included in the PSYM are most of the mode-id's (program entry points) for the standard system subroutines. There is no reason that a symbol internal to an assembly program cannot have the same name as a PSYM-file symbol, if the PSYM-file symbol is not also referenced in that program; such symbolic usage cannot be a "forward" reference in the assembly program. To avoid confusion, however, it is best to treat the entire set of PSYM symbols as reserved symbols.

3.1.2 Address Registers

All data referenced in the system is made indirectly through one of the sixteen address registers (A/R's). Registers zero and one have specifically defined meanings; the other fourteen may be considered general-perpose registers, with the limitation that system software conventions determine the usage of most A/R's. Registers zero and one should never be changed in any way by assembly programs. Register two always points to the SCB at logon time and after the debugger or the WRAPUP processor has been entered.

Register zero always addresses byte zero of the process's PCB; register one always addresses byte zero of the frame in which the process is currently executing. Thus all elements in the PCB may be relatively addressed using register zero as a base register. The more conventional way of setting up an A/R is to move a S/R into it. For example, the sequences below are functionally identical:

```
FRM100  ADDR  0,X'100'       DEFINE A LITERAL S/R
        .                    REFERENCING FRAME X'100'
        .
        .
        MOV   FRM100,R15
```

and

```
        SETUP0   R15,X'80000100'
```


3.1.3 Re-entrancy

In practiaclly all cases, the system software is re-entrant; that is, the same copy of the object code may be used simultaneously by more than one process. For this reason, no storage internal to the program is utilized; instead the storage space directly associated with a process is used; this is part of the process's Primary, Secondary, Tertiary (Debug), and Quadrenary Control Blocks. The Primary Control Block (PCB) is addressed via address register zero, the SCB via address register two. The Debug Control Block is used solely by the Debug processor, and should not be used by any other programs. The Quadrenary Control Block has no register addressing it; it is used by some system software (magnetic tape routines, for example) which temporarily set up a register pointing to it; its use is reserved for future software extensions.

A user program may utilize storage internal to the program if it is to be used in a non-re-entrant fashion; however, in most cases it will be found that the functional elements defined in the PSYM will be sufficient.

In some cases it may be required to set up a program to be executable by only one process at a time; that is, the code is "locked" while a process is using it, and any other process attempting to execute the same code waits for the first process to "unlock" it. The follwing sequence is typical;

```
        ORG  0
        TEXT X'01'      INITIAL CONDITION FOR LOCK BYTE
        CMNT *          (NOTE USAGE OF STORAGE INTERNAL
        CMNT *          TO PROGRAM)
        .
LOCK    MCC  X'00',R2   SET "LOCKED" CODE AT R2
        XCC  R2,R1      EXCHANGE BYTES AT R2 AND R1
        BCE  R2,X'01',CONTINUE
        CMNT *          OK TO CONTINUE; PROGRAM IS NOW LOCKED
        RQM  *          WAIT (RELEASE QUANTUM)
        B    LOCK       TRY AGAIN
        .
UNLOCK  MCC  X'01',R1   UNLOCK PROGRAM
```

3.1.4 Work-spaces or Buffers

There is a set of work-spaces, or buffer areas, that is pre-defined and available to each process. If the system conventions with regard to these buffers are maintained, they should prove adequate for the majority of assembly programming. There are three "linked" buffers, or work-spaces, of equal size, symbolically called the IS, the OS, and the HS. These are at least 3000 bytes in length each; more space for each area can be assigned to a process at LOGON time. There are five other work-spaces, known as the BMS, CS, AF, IB, and the OB, which may vary between 50 and 140 bytes in length, and are all in one frame. There is the TS, a one-frame unlinked work-space of 512 bytes, and the PROC work-space, 2000 bytes in length which is normally used by the PROC processor alone. Finally there are three additional frames (PCB+29 through PCB+31) that are unused by the system, and are freely available.

Each work-space is defined by a beginning pointer and an ending pointer, both of which are storage registers (S/R's). When the process is at the TCL level, all these pointers have been set to an initial condition. At other levels of processing, the beginning pointers should normally be maintained; the ending pointers may be moved by system or user routines. The address registers (A/R's) that are named after these work-spaces (IS, OS, AF, etc.) need not necessarily be maintained within their associated work-spaces; however, specific system routines may reset the A/R to its associated work-space. Note that, conventionally, a buffer beginning pointer addresses one byte before the actual location where the data starts. This is because data is usually moved into a buffer using one of the "move incrementing" type of instructions, which increment the A/R before the data movement.

Work-space	Location (offset from PCB)	Size	Linked?	Remarks
BMS	4 (disp.=0)	50	No	Normally contains an item-id when communicating with the disc file i/0 routines; typically, the item-id is copied to the BMS area, starting at BMSBEG+1; BMSBEG may be moved to point within any scratch area. BMSEND normally points to the last byte of the item-id; BMS (A/R) is freely usable except when explicitly or implicitly calling a disc file i/o routine
AF	4 (disp.=50)	50	No	This work-space is used by any system subroutine, though the AF A/R is used as a scratch register
CS	4 (disp.=100)	100	No	As above
IB	4 (disp.=200)	0-140	No	Used by terminal input routines to read data; IBBEG may be moved to point within any scratch area before use; IBEND conventionally points to the logical end of data; IB A/R is freely usable except when explicitly or implicitly calling a terminal input routine
OB	4 (disp.=201 +IBSIZE)	0-140	No	Used by terminal output routines to write data. OBBEG and OBEND should not be altered; they always point to the beginning and end of the OB area; OB (A/R) conventionally points one before the next available location in the OB buffer
TS	5	511	No	This work-space is not used by the system subroutines, other than those associated with the Conversion processor, though the TS A/R is used as a scratch register

PROC	6-9	2000	Yes	Used exclusively by the PROC processor for working storage; user-exits from PROC's may change pointers in this area
HS	10-15	3000+	Yes	Used as a means of passing messages to the WRAPUP processor at the conclusion of a TCL statement; may be used as a scratch area if there is no conflict with the WRAPUP history-string formats; HSBEG should not be altered; HSEND conventionally points one byte before the next available location in the buffer (initial condition is HSBEG=HSEND)
IS OS	16-21 22-27	3000+ 3000+	Yes Yes	These work-spaces are used interchangeably by some system routines since they are of the same size (and are equal in size to the HS); specific usage is noted under the various system routines ISBEG and OSBEG should not be altered, but may be interchanged if necessary; initially, ISEND and OSEND point 3000 bytes past ISBEG and OSBEG respectively (not at the true end. Additional work-space is assigned at LOGON time); IS and OS A/R's are freely usable except when calling system subroutines that use them.

3.1.5 Defining a Separate Buffer Area

If it is required to define a buffer area that is unique to a process, the unused frames PCB+29 through PCB+31 may be used. The following sequence of instructions is one way of setting up an A/R to a scratch buffer:

```
          .
          .
          LOAD      R0FID            GET PCB FID
          ADD       29               INC TO PCB + 29
          SETUP0    R6,D0            SETUP R6
          .
          .
```

Register three can now be used to reference buffer areas, or functional elements that are addressed relative to R3. None of the system subroutines use R3, so that a program has to set up R3 only once in the above manner. However, exit to TCL via WRAPUP WILL RESET R3 TO PCB+10.

3.1.6 Usage of XMODE

In several cases, the multiple-byte move instructions can be used (say, when building a table) even when it is not known whether there is enough room in the current linked set to hold the data. Normally, if the end of a linked frame set is reached, DEBUG is entered with a "forward link zero" abort condition. However, the tally XMODE may be set up to contain the mode-id of a user-written subroutine that will gain control under such a condition. This subroutine can then process the end-of-frame condition, and, by executing a RTN instruction, continue normal processing. Instructions that can be handled by this scheme are: INC register, MCI, MIC, MII, MIID, MIIR, and SID. Care should be taken in the case of MIIR to save register R15 in the subroutine. MIIT can be handled since the accumulator is saved in D1 by the debugger before it is used in transfering control via XMODE; therefore, D0 should be restored from D1 before returning from the XMODE trap.

For example:

```
         .
         .
         .
         MOV   XXX,XMODE    SET UP XMODE FOR NEXT
         CMNT  *            INSTRUCTION
         MII   R12,R13,SR4 COPY FROM R12 TO R13,
         CMNT  *            TILL R12=SR4
         ZERO  XMODE
         .
         .
         .

!XXX     EQU   *            ENTRY POINT FOR SUBROUTINE
         MOV   R15,SR1      SAVE R15
         SRA   R15,ACF      SET TO SAVE REGISTER NUMBER
         BCE   X'0D',R15,OK ENSURE TRAP WAS DUE TO R13
         MOV   0,XMODE      PREVENT DEBUG RE-ENTRY;
         ENT     5,DB1      NO! RE-ENTER DEBUG TO PRINT
         CMNT  *            "FORW LNK ZERO" MESSAGE
*
OK       SETUP R13,500,R13FID RESET DISPLACEMENT FIELD OF
         CMNT  *                R13, SINCE FIRMWARE HAS LEFT
         CMNT  *                IT IN A STRANGE STATE
*
* HANDLE END-OF-FRAME CONDITION HERE
*
         MOV   R13FID,RECORD SET UP INTERFACE
         BSL   GETSPC      GET ANOTHER OVERFLOW FRAME
         MOV   SR1,R15     RESTORE R15
         RTN   *           RETURN TO CONTINUE EXECUTION
         CMNT  *           OF MII INSTRUCTION
```


3.1.7 Initial Conditions

At any level in the system, the following elements are assumed to be set up; they should not be altered by any programs:

```
MBASE      D  + Contain the base-FID, modulo, and
MMOD       T  + separation of the M/DICT associated with
MSEP       T  + the process

USER       T    Used to indicate the status of the
                process, as follows:

               -1    Indicates the spooler process
                0    Indicates process not logged on
                1    Indicates the file-restore process
                2    Indicates a process which has been
                     logged off, and must release
                     work-space and go to MD0
                3    Indicates a process which must go
                     to LOGOFF after WRAPUP processing
                5    Indicates normal logged-on process.
```

3.1.8 Special PSYM Elements

Certain elements have a "global" significance to the system; in addition to those described above, they include the following:

```
Element        Description

HO        H  +
.            + Overlay  the accumulator and  extension;
.            + H7 is the high-order byte  of Dl; HO  is
.            + The low-order byte of DO
H7        H  +

INHIBITH  H    If non zero, the "BREAK" key on the terminal
               is  inhibited;  used by  processes  that
               should not be interrupted.  Conventionally, any
               process can increment INHIBITH to prevent
               BREAK KEY interuption. The subrouine DECINHIB
               should be used to decrement the inhibit half tally.

OVRFLCTR  D    Used by WRAPUP

RSCWA     T    Return-stack   current   word   address;
               contains the address one  byte past  the
               current entry in the stack; the stack is
               null if RSCWA=X'184'

SYSPRIV1  B    Indicates system privileges, level  one,
               if set

SYSPRIV2  B    Indicates system privileges,  level two,
               if set along with SYSPRIV1

TO        T  +
.            +
.            + Overlay the accumulator and extension
.            +
T3        T  +

XMODE     T    May  be  set   to   the  mode-id  of   a
               subroutine that is to  gain control when
               a "forward link zero" condition occurs
```


3.2 DOCUMENTATION CONVENTIONS

In the system software documentation, each routine is listed along with its entry point (as would be used in a DEFM statement); if the entry point is included in the standard PSYM file, it is followed by an asterisk (*). Unless otherwise specified, routines are meant to be called as subroutines, using a BSL instruction, and they return to the calling program via a RTN instruction. Be aware that there is no particular reason to believe that the referenced routine currently has the specified interface, name or location, or that it exists.

The Functional Description section for each routine briefly describes the action taken. The Input Interface, Output Interface, and Element Usage sections describe the functional elements used by the routine. The single letter following an element name describes its type: B=bit, C=character, H=half tally, T=tally (word), D=double tally, F=triple tally, R=address register, S=storage register. Even if not specified, the following elements may be destroyed by any routine.

Tallies	:	T4, T5
Double Tallies	:	Accumulator and extension (D0, D1), D2
Registers	:	R14, R15
Storage Registers	:	SYSR0, SYSR1, SYSR2

If no description follows an element name, it indicates that the element is used as a scratch element.

The system delimiters are symolically referred to as follows:

Hex. Value	Name and Description	
FF	SM	Segment Mark
FE	AM	Attribute Mark
FD	VM	Value Mark
FC	SVM	Secondary Value Mark
FB	SB	Start Buffer

3.3 SYSTEM SUBROUTINES

3.3.1 ATTOVF

ATTOVF is used to obtain a frame from the overflow space pool and to link it to the frame specified in double tally RECORD. The forward link field of the frame specified in RECORD is set to point to the overflow frame obtained, the backward link field of the overflow frame is set to the value of RECORD, and the other link fields of this overflow frame are zeroed.

Input Interface

RECORD	D	Contains the FID of the frame to which an overflow frame is to be linked

Output Interface

OVRFLW	D	Contains the FID of the overflow frame if obtained, or zero if no more frames are available

Element Usage

```
R15        R    Utility

INHIBITH B +
D0         D  + Used by GETOVF
R14        R  +
```

Subroutine Usage

GETOVF

Two additional levels of subroutine linkage required

3.3.2 BLOCK-SUB

This routine prints block letters on the terminal or line printer. It is used, for instance, by the TCL verbs "BLOCK-TERM" and "BLOCK-PRINT"; for more information, see the discussion of these verbs in the SYSTEM COMMANDS documentation.

Input Interface

IS	R	Points one before the first character to be output; the end of data is marked by the character pair SM Z (no space after the SM); if any element in the data string contains a SM, it must be terminated by a SB (see MD1B documentation, "Editing Features")
ZBIT	B	If set, output is directed to the terminal, otherwise output is passed to the spooler for line printer listing or other use

OBSIZE	T		Contains the maximum number of characters on each output line
OB	R		=OBBEG
SB0	B		If set, no test for terminal or printer output is made, terminal or printer characteristics are not initialized, the output device is not advanced to top-of-form, and the heading is not set null; all these actions take place if SB0 is reset
AFBEG	S	+	
BMSBEG	S	+	Point to scratch areas
HSEND	S	+	
LISTFLAG	B	+	
SMCONV	B	+	
NOBLNK	B	+	
LFDLY	T	+	As required by WRTLIN
PAGSIZE	T	+	
PAGSKIP	T	+	
PAGFRMT	B	+	

Output Interface

OB	R	=OBBEG
PAGINATE	B	=1
PAGHEAD	S	Points to a null page heading (SM) at HSEND if SB0=0

Element Usage

BITS	C	+	
SC0	C	+	
SC1	C	+	
SC2	C	+	
REJCTR	T	+	
C1	T	+	
CTR16	T	+	
CTR17	T	+	
CTR18	T	+	
CTR19	T	+	
D0	D	+	
D1	D	+	
BASE	D	+	
MODULO	T	+	Utility
SEPAR	T	+	
IR	R	+	
UPD	R	+	
BMS	R	+	
AF	R	+	
OB	R	+	
CS	R	+	
TS	R	+	

```
R15       R  +
SR4       S  +
.
.
SR22      S  +

CTR1      T     Used by CVTNIR

R14       R     Used by RETIX

T7        T  +  Used by WRTLIN
SYSR1     S  +
```

Subroutine Usage

RETIX; GBMS if the system file "BLOCK-CONVERT" is found; CVTNIR; WRTLIN; NEWPAGE if required; PRNTHDR if SB0=0; PCLOSEALL and SETLPTR if SB0=0 and ZBIT=0; SETTERM if SB0=1 or ZBIT=1

Six additional levels of subroutine linkage required if "BLOCK-CONVERT" is a "Q"-code item in the master dictionary, otherwise five levels required

Error Conditions

BLOCK-SUB exits to WRAPUP (MD995 or MD99) under the following conditions:

Error Number	Error type
520	Null input data
521	Too many characters (more than nine) in a word to block
522	BLOCK-CONVERT file missing or improperly defined in the master dictionary
523	Block output would exceed page width
524	An input character is not in the BLOCK-CONVERT file
525	An input character is improperly formatted in the BLOCK-CONVERT file

3.3.3 CONV - CONVEXIT

These entry points are used to call the entire conversion processor as a subroutine, which will perform any and all valid conversions specified in the conversion string. Other entry points may be used to perform certain specific conversions. Multilple conversion codes are separated by VM's in the conversion string. Conversion is called by the ENGLISH pre-processor to perform conversions on "input" data (in selection criteria), and by the LIST/SORT processor to perform "output" conversion.

CONV is the usual mode-id used to invoke conversion processing. CONVEXIT is the entry point to which any part of the conversion processor returns in order to check if more conversion is required (further VM's and conversion codes in the conversion string).

Input Interface

TSBEG	S		Points one before the value to be converted; the value is converted "in place", and the buffer is used for scratch space; therefore it must be large enoughto contain the converted value; the value to be converted is terminated by any of the standard system delimiters (SM, AM, VM, or SVM)
IS	R		Points to the first character of the conversion code specification string for CONV; for CONVEXIT, points at least one before the next conversion code (after a VM) or AM at the end of the string, or to the AM; the code string must end with an AM; initial semicolons (;) are ignored
MBIT	B		Set if "input" conversion is to be performed; reset for "output" conversion
DBIT	B	+	As required by TRANSLATE (see TRANSLATE
DAF1	B	+	documentation)
XBIT	B		As required by CFUNC (see CFUNC documentation)

Output Interface

TSBEG	S		Points one before the converted value
TS	R	+	Point to the last character of the
TSEND	S	+	converted value; a SM is also placed one past this location; TS=TSEND=TSBEG if a null value is returned
IS	R		Points to the AM terminating the conversion code(s)

Element		Conversions Where Used
DBIT	B	F,T
XBIT	B	F
GMBIT	B	F
WMBIT	B	F
SB10	B	All
SB12	B	All
DAF1	B	T
DAF9	B	T
SC2	C	C,D,F,T
T3	T	F,MD
T4	T	D,F,MD,MT
T5	T	D,F,MD,MT
T6	T	D,F,M
T7	T	F,MD
CTR1	T	C,F,G,T
CTR12	T	F
CTR13	T	F
CTR20	T	All
CTR21	T	D,MD,T
CTR22	T	D
CTR23	T	D,MD
CTR28	T	T
D1	D	C,F,MT,T
D2	D	D,F,MD,MT
D3	D	MT
D7	D	F
D8	D	F
D9	D	F
FP0	F	F,MD
FP1	F	F,MD
FP2	F	F,MD
FP3	F	F
FP4	F	F
FP5	F	F
FPX (SYSR0)	F	F,MD,T
FPY	F	F,MD
BASE	D	T
MODULO	T	T
SEPAR	T	T
RECORD	D	T
SIZE	T	T
NNCF	H	T
FRMN	D	T
FRMP	D	T
NPCF	H	T
XMODE	T	C,F,MT,T
IR	R	T
BMS	R	T
R14	R	D,MD,MT,MX,T
R15	R	All
SYSR1	S	T
SYSR2	S	T

S4	S	T
S5	S	F
S6	S	C,T
S7	S	All
SR0	S	C,F
SR1	S	F
SR4	S	C,T

Subroutine Usage

CVTHIS for "U" conversions; GCORR for "G" conversions; TRANSLATE for "T" conversions; CONCATENATE for "C" conversions; additional subroutines as used by routines listed under "Exits" below, and by user-written routines

The number of additonal levels of subroutine linkage required depends on the conversions performed - see the documentation for the various conversion routines for more specific information; note that for "F" conversions, CFUNC may call CONV recursively

User Conversion Processing

The conversion processor will pass control to a user-written routine if a "Uxxxx" code is found in the conversion string, where "xxxx" is the hexadecimal mode-id of the user routine. This routine can then perform special conversion before returning. The input interface for the user routine will be identical to that described in the preceding section; after performing the conversion the user routine should set up the output interface elements to be compatible with CONVEXIT, and then exit via an external branch to that point to continue the conversion process if multiple conversions are specified. Alternately, a RTN may be executed if this is not needed, or to prevent further conversions from being performed. Elements used by the regular conversion routines may safely be used by user routines; however, if additional elements are needed, a complete knowledge of the processor that called CONV (LIST, SELECTION, etc.) will be necessary.

Exits

To IDATE for "D" conversions on input (MBIT=1); to ODATE for "D" conversions on output; to ICONVMD or OCONVMD for "MD" conversion on input or output; to CFUNC for "F" conversions; to TIMECONV for "MT" conversions; to HEXCONV for "MX" conversions; all these routines, however, return to CONVEXIT

For output conversion, a null value returned causes an immediate end of conversion processing.

Error Conditions

CONV exits to WRAPUP after setting RMODE to zero under the following conditions:

705 Illegal conversion code

706	Illegal "T" conversion: format incorrect, filename cannot be found, etc.
707	DL/ID cannot be found for a "T" conversion file

WRAPUP is also entered without setting RMODE to zero under the following error conditions:

708	Value cannot be converted by a "T" conversion
339	Invalid format for input data conversion

3.3.4 DLINIT

DLINIT is used to obtain a block of contiguous overflow space for a file. After checking the input parameters and obtaining the necessary number of frames, if available, it enters DLINIT1 to initialize the frames (see DLINIT1 documentation). If not enough space is available for the file, DLINIT calls NOSPACE to find out if processing should be aborted (see NOSPACE documentation).

Input Interface

```
MODULO    T  + Contain the modulo and separation
SEPAR     T  + parameters for the file; if MODULO is
               initially less than or equal to zero, it
               is set to eleven; if SEPAR is initially
               less than or equal to zero, it is set to
               one, and if initially greater than 127
               it is set to 127
```

Output Interface

```
BASE      D    Contains the beginning FID of a
               contiguous block of size MODULO*SEPAR if
               the space is available, otherwise
               unchanged

OVRFLW    D    =BASE if the requested space is
               available, otherwise =0

RMBIT     B    Set if the requested space is obtained,
```

Element Usage

```
R14        R  +
R15        R  + Used by GETBLK
INHIBITSV2 B  +
D0         D  +
```

Subroutine Usage

GETBLK; NOSPACE if the requested space is unavailable

Three additional levels of subroutine linkage required

Exits

To DLINIT1 if the requested space is obtained; to NSPCQ (WRAPUP) from NOSPACE if the space is unavailable and processing is aborted by the user

3.3.5 DLINIT1

DLINIT1 initializes the link fields of a file as specified by its base, modulo, and separation parameters, and sets each group empty by adding an AM at the beginning (in the first data byte).

Input Interface

BASE	D	+ Contain the base, modulo, and separatio
MODULO	T	+ of the file; note - one frame is linked
SEPAR	T	+ even if MODULO is less than or equal to zero

Output Interface

R14	R	Points to the first data byte in the first frame of the last group in the file (set by LINK)
R15	R	Points to the last byte of the last frame of the last group in the file (set by LINK)
RECORD	D	=one greater than the FID of the last frame of the last group in the file
NNCF	H	=SEPAR-1

Frames are initialized as described above

Element Usage

CTR1	T	Utity
FRMN	D	+
FRMP	D	+ Used by LINK
NPCF	H	+

Subroutin.im 5 LINK

One additional level of subroutine linkage required

3.3.6 ENGLISH INTERFACE

Summary

It is possible to interface with the ENGLISH processor at several levels. A typical LIST or SORT statement passes through the Preprocessor and Selection processor before entering the LIST processor. All statements must pass through the first two stages, but control can be transferred to user-written programs from that point onward.

General Conventions

The ENGLISH processors use a compiled string that is stored in the IS work space. String elements are separated by SM's. There is one file-defining element in each string, one element for each attribute specified in the original statement, and special elements pertaining to selection criteria, sort-keys, etc. The formats of various string elements are as follows:

File Defining Element, at ISBEG+1:

```
SM D file-name AM base VM modulo VM separ AM conv AM
correl AM type AM just AM SM
```

Attribute Defining Element:

```
SM c attribute-name AM amc AM conv AM correl AM
type AM just AM SM

     c = A - regular or D2 attribute
         Q - D1 attribute
         Bx- SORT-BY, SORT-BY-DSND, etc.; "x" is from
             attribute one of the connective
```

Explicit Item-id's:

```
SM I item-id SM
```

End-of-string ELEMENT:

```
SM Z
```

The Selection Processor

This performs the actual retrieval of items which pass the selection criteria, if specified. Every time an item is retrieved, the processor at the next level is entered with bit RMBIT set; a final entry with RMBIT zero is also made after all items have been retrieved. If a sorted retrieval is required, the Selection processor passes items to the GOSORT mode, which builds up the sort-keys preparatory to sorting them. After sorting, GOSORT then retrieves the items again, in the requested sorted sequence.

A user program may get control directly from the Selection processor (or GOSORT if a sorted retrieval is required); the formats of the verbs are:

Line number	Non-sorted	Sorted
1	PA	PA
2	35	35

3	xxxx	76
4	xxxx	

where "xxxx" represents the mode-id of the user program. Note that in this method of interface, only item retrieval has taken place; none of the conversion and correlative processing has been done. For functional element interface, the column headed "Selection Processor" in the table shown later must be used.

Exit Convention: On all but the last entry, the user routine should exit indirectly via RMODE (using an ENT* RMODE instruction); on the last entry, the routine should exit to one of the WRAPUP entry points. Processing may be aborted at any time by setting RMODE to zero and entering WRAPUP. Bit SB0 must also be set on the first entry.

Special Exit From The LIST Processor

A user program may also gain control in place of the normal LIST formatter, to perform special formatting. The advantage here is that all conversions, correlatives, etc. Have been processed, and the resultant output data has been stored in the history string (HS area). The formats of the verbs then are:

Line number	Non-sorted	Sorted
1	PA	PA
2	35	35
3	4D	4E
4	xxxx	xxxx

where "xxxx" is the mode-id of the user program.

Output data is stored in the HS area; data from each attribute is stored in the string, delimited by AM's; multiple values and sub-multiple-values are delimited within an element by VM's and SVM's, respectively. Since the HS may contain data other than the retrieved item, the user program should scan from HSBEG, looking for a segment preceded by an "X"; all segments except the first are preceded by a SM. The format is:

X item-id AM value one AM ... AM value n AM SM Z

The program must reset the history string pointer HSEND as items are taken out of the string. In special cases, data may not be used until, say, four items are retrieved, in which case HSEND is reset on every fourth entry only. HSEND must be reset to point one byte before the next available spot in the HS work space, normally one before the first "X" code found.

The exit convention for the LIST processor is the same as for the Selection processor (see above).

Example: The following program is an example of one which prints item-id's (only) four at a time across the page.

```
001             FRAME  504
002             ZB     SB30            INTERNAL FLAG
003             BBS    SB0,NOTF        NOT FIRST TIME
004 * FIRST TIME SETUP
005             MOV    4,CTR32
006             SB     SB0
007 *
008 NOTF        BBZ    RMBIT,PRINTIT   LAST ENTRY
```

```
             BDNZ   CTR32,RETURN     NOT YET 4 ITEMS OBTAINED
             MOV    4,CTR32          RESET
    PRINTIT  MOV    HSBEG,R14
    LOOP     INC    R14
             BCE    C'X',R14,STOREIT FOUND AN ITEM
             BCE    C'Z',R14,ENDHS END OF HS STRING
    SCANSM   SCD    R14,X'C0'        SCAN TO NEXT SM
             B      LOOP
    STOREIT  BBS    SB30,COPYIT      NO FIRST ID FOUND
             SB     SB30             FLAG FIRST ID FOUND
             MOV    R14,SR28         SAVE LOCATION OF FIRST
             CMNT   *                "X"
    COPYIT   MIID   R14,OB,X'A0'     COPY ITEM-ID TO OB
             MCC    C' ',OB          OVERWRITE AM
             INC    OB,5             INDEX
             B      SCANSM
    ENDHS    BSL    WRTLIN           PRINT A LINE
             MOV    SR28,HSEND       RESTORE HS TO FIRST
             CMNT   *                "X" CODE
             DEC    HSEND            BACK UP ONE BYTE
             BBZ    RMBIT,QUIT
    RETURN   ENT*   RMODE            RETURN TO SELECTION
             CMNT   *                PROCESSOR
    QUIT     ENT    MD999            TERMINATE PROCESSING
             END
```

Element Usage

The following table summarizes the functional element usage by the Selection and LIST processors. Only the most important usage is described; elements that have various usages are labeled "scratch." a " " (blank) indicates that the processor does not use the element. Since the LIST processor is called by the Selectin processor, any element used for "memory" purposes (not to be used by others) in the former is indicated by a blank usage in the latter column.

In general, user routines may freely use the following elements:

- Bits : SB20 upwards
- Tallies : CTR30 upwards
- Double tallies: D3-D8
- S/R's : SR20 upwards

SB0 and SB1 have a special connotation: they are zeroed by the Selection processor when it is first entered, and not altered thereafter. They are conventionally used as first-time switches for the next two levels of processing. SB0 is set by the LIST processor when it is first entered, and user programs that gain control directly from Selection should do the same. SB0 may be used as a first-entry switch by user programs that gain control from the LIST processor.

An ENGLISH verb is considered an "update" type of verb if the SCP character (from line one of the verb definition) is B, C, D, E, G, H, I, or J. SCP characters of B, C, D, and I are reserved for future ENGLISH update verbs.

Bits	Selection Processor	LIST Processor
ABIT	scratch	non-columnar list flag
BBIT	first entry flag	
CBIT	scratch	scratch
DBIT	scratch	dummy control-break

EBIT	reserved	control-break flag
FBIT	reserved	scratch
GBIT	reserved	scratch
HBIT	reserved	scratch
IBIT	explicit item-id's specified	
JBIT	reserved	D2 attribute in process
KBIT	by-exp flag	by-exp flag
LBIT	scratch	left-justified field
MBIT	CONV interface; zero	zero
NBIT	scratch	scratch
OBIT	selection test on item-id	
PBIT	scratch	scratch
QBIT	scratch	scratch
RBIT	full-file-retrieval flag	
SBIT	selection on values (WITH)	
TBIT	scratch	print limiter flag
UBIT	scratch	reserved
VBIT	reserved	scratch
WBIT	scratch	reserved
XBIT	scratch	reserved
YBIT	left-justified value being tested	left-justified print limiter test
ZBIT	left-justified item-id	
SB0	unavailable	first entry flag, level one
SB1	unavailable	first entry flag, level two
SB2	reserved; zero	
SB4 through SB17	scratch or reserved	scratch or reseved
VOBIT	set for WRAPUP interface	
COLHDRSUPP	set if the corre-	
DBLSPC	sponding connective	
HDRSUPP	was found in the	
IDSUPP	input statement	
DETSUPP		
LPBIT		
TPBIT		
CBBIT		
PAGFRMT		
RMBIT	set on exit if an item was retrieved; zero on final exit	
WMBIT	FUNC interface	FUNC interface
GMBIT	FUNC intrface	FUNC interface
BKBIT	scratch	scratch
DAF1	set if SCP=B, C, D, E, G, H, I, or J	

DAF8	set if accessing a dictionary	
Tallies	Selection processor	LIST processor
C1;C3-C7	scratch	scratch
C2	contents of MODEID2	
CTR1-CTR4	scratch	scratch
CTR5	scratch	AMC of the current element in the IS
CTR6	reserved	scratch
CTR7	reserved	AMC corresponding to IR
CTR8	reserved	scratch
CTR9	reserved	scratch
CTR10	reserved	scratch
CTR11	reserved	scratch
CTR12	FUNC interface	current sub-value counter count
CTR13	FUNC interface	current value count
CTR14	reserved	scratch
CTR15	reserved	item size
CTR16	reserved	scratch
CTR17	reserved	reserved
CTR18	reserved	scratch
CTR19	reserved	sequence no for by-exp
CTR20-CTR23	CONV interface	CONV interface
CTR24	reserved	scratch
CTR25	reserved	scratch
CTR26	reserved	scratch
CTR27	reserved	current max-length
CTR28	reserved	scratch

Other storage	Selection processor	LIST processor
D9	count of retrieved items	
D7	FUNC interface	FUNC interface
FP1-FP5	FUNC interface	FUNC interface
RMODE	return mode-id (MD3)	
SIZE	item-size	scratch
SBASE SMOD SSEP	file base, modulo, and separation	
DBASE DMOD DSEP	dictionary base, modulo, and separation	

S/R's	Selection processor	LIST Processor
S1	points to the next explicit item-id	
S2-S9	scratch	scratch
SR0	points one before the item count field	
SR1	points to the correlative field	current correlative segment in the IS

SR2	scratch	scratch
SR3	reserved	scratch
SR4	points to the last AM of the item	
SR5	reserved	points to the next
SR6	points to the conversion field	current conversion field
SR7	reserved	scratch
SR8-SR12	reserved	reserved
SR13	GOSORT only: next sort-key	reserved
SR14-SR19	reserved	reserved
PAGHEAD	heading in the HS if HEADING was specified	generated heading in the HS

A/R's	Selection Processor	LIST Processor
AF	scratch	scratch
BMS	within the BMS area	scratch
CS		scratch
IB		scratch
OB		output data line
IS	compiled string	compiled string
OS		scratch
TS	within the TS area	within the TS area
UPD		within the HS area
IR	within the item	within the item

Work Space Usage	Selection Processor	LIST processor
AF	scratch	
BMS	contains the item-id	
CS	control break value string	
IB	output line	
IS	compiled string	
OS	scratch	
HS	heading data	heading data; attribute data for special exits
TS	scratch	current value in process

Additional Notes

1. If a full-file-retrieval is specified, the additional internal elements as used by GETITM will be used. If explicit item-id's are specified, RETIX is used for retrieval of each item.

2. Most elements used by the CONV and FUNC processors have been shown in the table; both may be called either by the Selection processor or the LIST processor.

3. Since the ISTAT and SUM/STAT processes are independently driven by the Selection processor, the element usage of these processors is not shown.

4. The section of the IS and OS used by the Selection and LIST processors is delimited by ISEND and OSEND respectively. The buffer space beyond these pointers is available for use by other programs.

3.3.7 GETBUF - G3 GETBUF - G3

These routines accept input data from the terminal and perform some editing on the characters obtained. GETBUF also prints an initial prompt character at the terminal before reading input. Control is returned when a non-editing control character is input, or when the number of characters specified in T0 or T1 are input.

Editing Features

Control-H	Logically backspaces the buffer pointer; echoes character in BSPCH
Control-X	Logically deletes the entire input buffer; echoes a CR/LF, and prints the prompt character
Control-R	Retypes the input line
Rubout	Ignored; the character is echoed, but is not stored in the buffer
Control-shift-K Control-shift-L Control-shift-M Control-shift-N Control-shift-O	These characters are converted to the internal delimiters SB, SVM, VM, AM, and SM, respectively; they echo as the characters [, /,], ^, and _

Note: the high order bit of all characters input is zeroed.

Input Interface

BSPCH	C	Contains the character to be echoed to the terminal when the back space key is pressed; required by G3
PRMPC	C	Character output as a "prompt" when input is first requested by GETBUF, and after certain editing operations by both GETBUF and G3
T0	T	Contains the maximum number of Characters accepted (for GETBUF only)
T1	T	Contains the maximum number of characters to be accepted (for G3 only)

R14 R Points one byte before the beginning of the input buffer area (for GETBUF only)

R15 R Points one byte before the beginning of the input buffer area (for G3 only)

Output Interface

R15 R Points to the control character causing return to the calling routine

Element Usage

D0

3.3.8 GETIB - GETIBX GETIB - GETIBX

GETIB and GETIBX are the standard termianal input routines. Register IBBEG points to a buffer area where the routine will input the data. Input continues to this area until either a carriage return or line feed is encountered, or until a number of characters equal to the count stored in IBSIZE have been input. The carriage return or line feed terminating the input line is overwritten with a segment mark (SM), and register IBEND points to this character on return. If the input is terminated because the maximum number of characters has been input, a SM will be added at the end of the line.

This routine calls GETBUF to read input data from the terminal, and then determines if the last character was a carriage return or line feed, and echoes a CR/LF to the terminal. If the last character was a control character (see GETBUF documentation), GETIB/GETIBX either accepts or deletes the character, depending on the value of bit CCDEL, and calls GETBUF again.

The entry GETIB also provides the facility for taking input from a stack instead of directly from the terminal (see below). This feature is used, for example, by the PROC processor to store input lines which are returned to requesting processors as if they originated at the terminal. If the last character in a stacked line is a " ", it is replaced with a SM. Terminal input resumes when the stacked input is exhausted. GETIBX does not test for stacked input.

Input Interface

CCDEL B If set, control characters are deleted from terminal input

IBBEG S Points one byte before the buffer area where input is to be stored; the buffer must be two bytes greater than IBSIZE

IBSIZE T Contains the maximum number of characters accepted for input

LFDLY	T	Contains (in the low-order byte) the number of "fill" characters (nulls) to be issued after a CR/LF echo to the terminal; required by PCRLF
PRMPC	C	Terminal prompt character; required by GETBUF
BSPCH	C	Contains the character to be echoed to the terminal when the back space key is pressed; required by G3
STKFLG	B	If set, GETIB tests for "stacked" input; terminal input will not be requested until stacked input is exhausted
STKINP	S	Points to the next "stacked" input line; lines are deliminated by AM's, with a SM indicating the end of the stack

Output Interface

IB	R	=IBBEG
IBEND	S	Points to a SM one byte past the end of input data (overwrites the CR or LF)
STKFLG	B	Zeroed if the end of stacked input was reached; not changed if initially zero
STKINP	S	Points to the next line of stacked input (or end of stack) if stacked input is being processed

Element Usage

R14	R
R15	R

Subroutine Usage

If no stacked input: GETBUF, G3, PCRLF (if CCDEL=1)

One additional level of subroutine linkage required

Error Conditions

if a stacked input line exceeds IBSIZE, the line is truncated at IBSIZE; the remainder of the line is lost.

3.3.9 GETITM GETITM

This routine sequentially retrieves all items in a file. It is called repetitively to obtain items one at a time until all items have been retrieved. The order in which the items are returned is the same as the storage sequence.

If the items retrieved are to be updated by the calling routine (using routine UPDITM), this should be flagged to GETITM by setting bit DAF1. For updating, GETITM performs a two-stage retrieval process by first storing all item-ids (per group) in a table, and then using this table to actually retrieve the items on each call. This is necessary because, if the calling routine updates an item, the data within this group shifts around; GETITM cannot simply maintain a pointer to the next item in the group, as it does if the "update" option is not flagged.

An initial entry condition must also be flagged to GETITM by zeroing bit DAF7 before the first call. GETITM then sets up and maintains certain pointers which should not be altered by calling routines until all the items in the file have been retrieved (or DAF7 is zeroed again).

Note the functional equivalence of the output interface elements with those of RETIX.

Input Interface

DAF7	B		Initial entry flag; must be zeroed on the first call to GETITM
DAF1	B		If set, the "update" option is in effect
DBASE	D	+	Contain the base, modulo, and separation
DMOD	T	+	of the file
DSEP	T	+	
BMSBEG	R		Points one prior to an area where the item-id of the item retrieved on each call may be copied
OVRFLCTR	D		Meaningful only if DAF1 is set; if non-zero, the value is used as the starting FID of the overflow space table where the list olrbem-ids is stored; if zero, GETSPC is called to obtain space for the table

Output Interface

RMBIT	B	+	
SIZE	T	+	
R14	R	+	(See RETIX documentation)
IR	R	+	
SR4	S	+	
XMODE	T	+	
SR0	S		=R14 if DAF1 is set, otherwise as set by GNSEQI
BMS	R		As set by RETIX if DAF1 is set, otherwise as set by GNSEQI
BMSEND	S		=BMS if DAF1 is set, otherwise unchanged

```
    DAF9       B     =0
```

Element Usage

```
    BASE       D  +
    MODULO     T  +
    SEPAR      T  +
    RECORD     D  + Used by GETITM and other subroutines for
    NNCF       H  + accessing file data
    FRMN       D  +
    FRMP       D  +
    NPCF       H  +

    OVRFLW     D    Used by GETSPC if DAF1 is set and
                    OVRFLCTR is initially zero
```

The following elements should not be altered by any other routine while GETITM is used:

```
    DAF1       B  + (See Input Interface)
    DAF7       B  +

    DBASE      D    Contains the beginning FID of the
                    current group being processed

    DMOD       T    Contains the number of groups left to be
                    processed

    DSEP       T    (Unchanged)

    SBASE      D  + Contain the saved values of DBASE, DMOD,
    SMOD       T  + and DSEP when the routine was first
    SSEP       T  + called

    NXTITM     S    Points one before the next item-id in
                    the pre-stored table if DAF1 is set,
                    otherwise points to the last AM of the
                    item previously returned

    OVRFLCTR   D    Contains the starting FID of the
                    overflow space table if DAF1 is set,
                    otherwise unchanged
```

Subroutine Usage

RCREC, GNSEQI; GNTBLI (local), RETIX, and GETSPC (if OVRFLCTR =0) if DAF1 is set

BMSOVF used with XMODE

Four additional levels of subroutine linkage required

Error Conditions

See RETIX documentation ("Exits"); GETITM, however, continues retrieving items until no more are present even after the occurance of errors

3.3.10 GETOPT GETOPT

This routine processes an option string consisting of single alphabetic characters and/or a numeric option, separated by commas. A numeric option consists of a numeric character or a pair of numeric characters separated by a hyphen. If the option string contains more than one numeric option, the last one will be used. Alphabetic options set the corresponding bits ("A" sets ABIT, etc.), but these bits are not zeroed upon entry. The option string begins one past the address pointed to by register IS, and must end with a right parenthesis (")").

Input Interface

IS	R	Points one before the option string

Output Interface

ABIT	B	+
.		+
.		+ Set as described above
.		+
ZBIT	B	+
NOBIT	B	Set if a numeric option is found, otherwise zeroed
RMBIT	B	Set if no errors are found in the option format, otherwise unchanged
D4	D	=value of the first number in a numeric option, if found, otherwise unchanged
D5	D	=value of the second number in a numeric option, if found; =D4 if a numeric option consists of a single number; otherwise unchanged
IS	R	Points to the last character processed (=")" if no format errors are found)
RMODE	T	=0 if a format error is found

Element Usage

D0 and D1

Subroutine Usage

CVTNIS if a numeric option is found

Two additional levels of subroutine linkage required

Exits

To MD995 with error 209 if a format error is found

3.3.11 GETOVF GETOVF

GETBLK
GETSPC

These routines obtain overflow frames from the overflow space pool maintained by the system. GETOVF and GETSPC are used to obtain a single frame; GETBLK is used to obtain a block of contiguous space (used mainly by the CREATE-FILE processor). Note that the link fields of the frame(s) obtained by a call to GETBLK are not reset or initialized in any way - this is a function of the calling routine. GETOVF and GETSPC zero all the link fields of the frame they return.

These routines cannot be interrupted until processing is complete.

Input Interface

D0	D	Contains the number of frames needed (block size), for GETBLK only

Output Interface

OVRFLW	D	If the needed space is obtained, this element contains the FID of the frame returned (for GETOVF and GETSPC) or the FID of the first frame in the block returned (for GETBLK); if the space is unavailable, OVRFLW=0

Element Usage

```
INHIBITSV2 B +
D0         D + Utility
R14        R +
R15        R +
```

Subroutine Usage

SYSGET (but not used by GETOVF if a frame is obtained from a multiple-frame block in the system overflow table); three internal subroutines; GETOVF called by GETSPC; NOSPACE called by GETSPC if no frames are available

One additional level of subroutine linkage required by GETOVF and GETBLK; three levels required by GETSPC

Exits

For GETSPC: to NSPCQ if no more frames are available and processing is aborted by the user; this is a function of NOSPACE

3.3.12 GETUPD GETUPD

GETUPD initializes the UPD register triad to point to the UPD work space (frame PCB+28).

Input Interface

None

Output Interface

UPD	R	+ Point to the first data byte of the
UPDBEG	S	+ frame 28 frames after the process's PCB
UPDEND	S	Points to the last byte of the above frame

3.3.13 GNSEQI GNSEQI

This routine gets the next sequential item from a file. If its pointer into the file (register NXTITM) is at the end of a group, it returns with bit RMBIT zero; otherwise it copies the item-id into the area specified by register BMS, updates NXTITM, sets RMBIT, sets registers pointing to the beginning and end of the item, and returns the item size in tally SIZE. If a non-hexadecimal digit is found in the item count field, or the computed item size is negative or zero, GNSEQI immediately returns to the routine which called it.

Input Interface

NXTITM	S	Points one before the beginning of the next item to be retrieved (or the AM at the end of the group)
BMS	R	Points one before the area to which the item-id is to be copied

Output Interface

RMBIT	B	Set if an item was successfully retrieved, otherwise zeroed
NXTITM	S	Points one before the following item or end-of-group AM if RMBIT is set, otherwise unchanged
BMS	R	Points to an AM after the copied item-id if the item was retrieved, otherwise unchanged
SR0	S	=the initial value of NXTITM if not at the end of the group, otherwise unchanged
SR4	S	=NXTITM if RMBIT is set, otherwise unchanged
IR	R	Points to the AM after the item-id if RMBIT is set; points to the AM before the item-id if SIZE is zero onegative; points to the AM indicating end of group data if there were no more items in the group when the routine was called; points to the character in error if a non-hexadecimal character is found in

		the item count field
SIZE	T	Contains the value of the item count field if RMBIT is set
XMODE	D	=0

3.3.14 GNTBLI GNTBLI

This routine retrieves the next entry from a table consisting of strings (typically item-ids) separated by AMs, and terminated by a SM. On each call, the routine checks if its pointer (register NXTITM) is at the end of the table. If it is, the routine exits with bit RMBIT zero; otherwise the next table element is copied into the buffer specified by register BMS, NXTITM is set pointing to the following element, and RMBIT is set.

Input Interface

NXTITM	S	Points one before the next table entry (or SM)
BMS	R	Points one before the area to which the table entry is to be copied

Output Interface

NXTITM	S	Points to the AM following the entry which was copied, if one was copied, otherwise one before the SM at the end of the table
IR	R	=NXTITM if an element was copied, otherwise NXTITM+1
BMS	R	Points to an attribute mark one past the end of the entry copy, if present, otherwise unchanged
RMBIT	B	Zeroed if NXTITM points to the end of the table when the routine is called, otherwise set

3.3.15 HGETIB

This routine accepts a line of input from the terminal, like GETIB, and also handles tabs if bit STKFLG is zero. A table of preset tab positions, in increasing order of column numbers, is assumed to be set up in tallies CTR8-CTR15. Up to 16 tab positions may be stored, two per tally, with unused positions set to zero. When a horizontal tab character (control-I, X'09') is encountered in the input string, the cursor is positioned according to the tab table, and the input line is filled with the appropriate number of blanks.

Input Interface

STKFLG	B	If set, the routine immediately enters GETIB, without processing tab characters; if set, GETIB tests for "stacked" input; terminal input will not be requested until stacked input is exhausted (see GETIB documentation)
IBBEG	S	Points one byte before the buffer area where input is to be stored; the buffer must be two bytes greater than IBSIZE
IBSIZE	T	Contains the maximum number of characters accepted for input
LFDLY	T	Contains (in the low-order byte) the number of "fill" characters (nulls) to be issued after a CR/LF echo to the terminal; required by TCRLF (and PCRLF)
PRMPC	C	Contains the terminal prompt character; required by GETBUF
BSPCH	C	Contains the character to be echoed to the terminal when the back space key is pressed; required by G3
CCDEL	B	If set, control characters are deleted from terminal input
STKINP	S	Points to the next "stacked" input line; lines are deliminated by AM's, with a SM indicating the end of the stack; meaningful only if STKFLG is set

```
CTR8     T  +
.           +
.           + Contain tab positions as described above
.           +
CTR15    T  +
```

Output Interface

IB	R	=IBBEG
IBEND	S	Points to a SM one byte past the end of input data (overwrites the CR or LF)
STKFLG	B	Zeroed if the end of stacked input was reached; not changed if initially zero
STKINP	S	Points to the next line of stacked input (or end of stack) if stacked input is being processed

Element Usage

```
D0      D  +
D1      D  +
R14     R  +
R15     R  + Utility
IB      R  +
CTR7    T  +
CTR16   T  +
```

Subroutine Usage

GETBUF; TCRLF; G3

Two additional levels of subroutine linkage required

3.3.16 HSISOS

This routine sets up the register triads for the HS, IS, and OS work spaces as described below. It does not link frames in the work spaces.

Input Interface

None

Output Interface

```
R2      R    Points to the Secondary Control Block
             (PCB+1)

HS      R  + Point to the beginning of the HS work
HSBEG   S  + space (PCB+10)
HSEND   S  +

IS      R  + Point to the beginning of the IS work
ISBEG   S  + space (PCB+16)

ISEND   S    Points to the last data byte in the
             primary OS work space (3000 bytes past
             ISBEG)

OS      R  + Point to the beginning of the OS work
OSBEG   S  + space (PCB+22)
```

OSEND	S	Points to the last data byte in the primary OS work space (3000 bytes past OSBEG)

The first byte in each work space is set to X'00'.

Element Usage

DO

3.3.17 INITTERM - RESETTERM

These routines are used to initialize terminal and line printer characteristics. RESETTERM is called from WRAPUP before reentering TCL; INITTERM is called from LOGON.

Input Interface

OBSIZE	T	Contains the value of the output (OB) buffer (RESETTERM only)
OBBEG	S	Points to the start of the OB buffer

Output Interface

```
TOBSIZE   T  +
TPAGSIZE  T  +
POBSIZE   T  + Initialized  to  default  values,  as by
PPAGSIZE  T  + SETUPTERM (INITTERM only)
PAGSKIP   T  +
LFDLY     T  +
BSPCH     C  +

CCDEL     B  +
SMCONV    B  +
STKFLG    B  +
PAGINATE  B  +
NOBLNK    B  +
LPBIT     B  + =0
TPAGNUM   T  +
TLINCTR   T  +
PPAGNUM   T  +
PLINCTR   T  +
PAGNUM    T  +
LINCTR    T  +

PAGHEAD   S    Contains zero in the frame field

OB        R    =OBBEG
OBSIZE    T    =TOBSIZE

R14       R  + =OBBEG+OBSIZE
OBEND     S  +
```

The area from the address pointed to by OBBEG to that pointed to by OBEND is filled with blanks

3.3.18 IROVF

These routines can be used to handle end-of-linked-frames conditions when using register IR with MCI, MII, or MIID instructions. By setting tally XMODE to the mode-id of one of these routines before executing the instruction, the routine will be entered automatically if an end-of-linked-frames (forward link zero) condition occurs. A warning message will be printed and control will pass to the instruction following the MCI, MII, or MIID instruction. Additionally, bit DAF9 may be set to truncate group data so that the condition does not arise again. The only difference between the two IROVF entry points is that the one in SYSTEM-SUBS-II initializes register R14 to be compatible with routines such as GNSEQI, and then branches to the code in WSPACES-II.

Input Interface

IR	R	Points into the frame whose forward link is zero
DAF9	B	If set, group data is terminated at the address specified by R14 (UPDITM, for instance, uses this feature); otherwise the warning message is printed but the data is unchanged
R14	R	Points to the address at which group data is to be truncated if DAF9 is set, typically the end of the last good item in the group; an AM is stored in the byte addressed by R14, marking the end of an item, and another AM is stored in the following byte, marking the end of a group
OBBEG	S	Points one prior to an output buffer for printing an error message (required by WRTLIN)
NXTITM	S	Contains the value to be used in R14 for group data truncation (SYSTEM-SUBS-I entry only)

Output Interface

IR	R		Points to the last byte of the frame
R14	R	+	=IR-1
SR4	S	+	
RMBIT	B	+	
LISTFLAG	B	+	=0
SIZE	T	+	
XMODE	T	+	

The message "*GROUP FORMAT ERROR xxxx" is printed, where "xxxx" is the number of the frame pointed to by IR

Element Usage

```
R15        R

T4         T  +
T5         T  + Used by MBDSUB
D1         D  +
D2         D  +
```

Subroutine Usage

MBDSUB; WRTLIN

BMSOVF used with XMODE if DAF9=1

Five additional levels of subroutine linkage required if LPBIT is set (for WRTLIN); Four levels required if DAF9 is set and BMSOVF is entered to obtain another overflow frame (using ATTOVF) - this would occur if R14 were also pointing at the end of a set of linked frames when IROVF was entered; one level always required for MBDSUB

3.3.19 ISINIT

ISINIT simply invokes WSINIT and HSISOS to initialize all the process work space pointers.

Input and Output Interfaces

See WSINIT and HSISOS documentation.

Element Usage
D0

Subroutine Usage
WSINIT, HSISOS
Three additional levels of subroutine linkage required

3.3.20 LINESUB

This routine returns the line number of the calling process in the accumulator

Input Interface
None

Output Interface

```
D0        D    Contains the line number associated with
               the process
```

Element Usage

```
D1        D
```

Subroutine Usage
GPCB0
One additional level of subroutine linkage required

3.3.21 MD415

This routine is used to pick up numeric parameters from a string addressed by register IB. Parameters may be either a single string of numeric characters, or two such strings separated by a hyphen.

Input Interface

IB	R	Points at least one before the first non-blank character of the parameter string, or to a SM indicating no parameters
SC2	C	Contains a blank

Output Interface

C3	T	Contains the value of the first numeric parameter if one is converted, otherwise set to zero
C4	T	Contains the value of the second numeric parameter except under the following conditions: if zero or one parameters are present, C4 is set to X'7FFF'; if the second parameter is less than the first, C4 is set equal to C3
IB	R	Points to the first non-blank character after the converted parameter string, but unchanged if originally pointing to a SM

3.3.22 NEWPAGE

This routine is used to skip to a new page on the terminal or line printer and print a heading. No action is performed, however, if bit PAGINATE or tally PAGSIZE is zero.

Input Interface

As for WRTLIN, except OB is first set equal to OBBEG by this routine

Output Interface

Same as for WRTLIN

Element Usage

Same as for WRTLIN

Subroutine Usage

WRTLIN and routines called by it, if PAGINATE is set and PAGSIZE is greater than zero

Additional subroutine linkage required only if WRTLIN is called; see WRTLIN documentation for the number of additional levels of linkage required, and add 1

3.3.23 NEXTIR - NEXTOVF

NEXTIR obtains the forward linked frame of the frame to which register IR (R6) currently points; if the forward link is zero, the routine attempts to obtain an available frame from the system overflow space pool and link it up appropriately (see ATTOVF documentation). In addition, if a frame is obtained, the IR register triad is set up before return, using routine RDREC.

NEXTOVF may be used in a special way to handle end-of-linked-frame conditions automatically when using register IR with single- or multiple-byte move or scan instructions (MIID, MII, or MCI). Tally XMODE should be set to the mode-id of NEXTOVF before the instruction is executed; if the instruction causes IR to reach an end-of-linked-frame condition (forward link zero), the system will generate a subroutine call to NEXTOVF, which will attempt to obtain and link up an available frame, and then resume execution of the interrupted instruction (assuming a frame was gotten). If there are no more frames in the overflow space pool, NOSPACE is called. Note that the "increment register by tally" instruction cannot be handled in this manner.

NEXTOVF is also used by UPDITM with register TS (R13). If NEXTOVF is entered with TS at an end-of-linked-frames condition, a branch is taken to a point inside UPDITM. Under any other condition (other than IR or TS end-of-linked-frame), NEXTOVF immediately enters the DEBUGGER.

Input Interface

IR	R		Points into the frame whose forward-linked frame is to be obtained (displacement unimportant)
ACF	H		For NEXTOVF only, must contain X'06' for IR end-of-linked-frame handling (set automatically by MIID, MII, and MCI instructions)

Output Interface

IR	R	+	Point to the first data byte of the
IRBEG	S	+	forward linked frame
IREND	S		Points to the last byte of the forward linked frame
RECORD	D		Contains the FID of the frame to which IR points
R15	R	+	
NNCF	H	+	
FRMN	D	+	As set by RDLINK for the FID in RECORD
FRMP	D	+	
NPCF	H	+	
OVRFLW	D		=RECORD if ATTOVF called, otherwise unchanged

Element Usage

R14	R	Used by RDLINK

Elements used by ATTOVF if a frame is obtained from the overflow space pool

Subroutine Usage

RDLINK; ATTOVF if a frame must be obtained from the overflow space pool; NOSPACE if ATTOVF cannot find any more frames

Three additional levels of subroutine linkage required

Exits

Normally returns via RDREC; possibly to NSPCQ if NOSPACE used (see NOSPACE documentation); to 5,DB1 if ACF not X'06' or X'0D' (NEXTOVF only)

3.3.24 OPENPFILE

This routine retrieves the base, modulo, and separation parameters of the system file POINTER-FILE, and bypasses the normal lock-code tests in doing so.

Input Interface

BMSBEG	S	Points to an area where the POINTER-FILE file-name may be copied, for RETIX

Output Interface

BASE	D	+	Contain the POINTER-FILE base, modulo,
MODULO	T	+	and separation
SEPAR	T	+	

Element Usage

R15	R	+	Utility
BMS	R	+	
CTR1	T		Used to save the value of tally USER
RECORD	D	+	
SIZE	T	+	
NNCF	H	+	
FRMN	D	+	
FRMP	D	+	
NPCF	H	+	Used by RETIX
IR	R	+	
R14	R	+	
BMSEND	S	+	
SR4	S	+	
XMODE	T	+	
DAF9	B	+	

```
SYSR0     S  + Used by GBMS if the POINTER-FILE item in
SYSR1     S  + the SYSTEM dictionary is a "Q" code item
SYSR2     S  +
```

Subroutine Usage

GMMBMS; RETIX; GBMS unless the POINTER-FILE entry in the SYSTEM dictionary is missing

Six additional levels of subroutine linkage required if the POINTER-FILE entry in the SYSTEM dictionary is a "Q" code item, otherwise four levels required

Exits

To MD994 with message 201 (value in C1) if the POINTER-FILE entry in the SYSTEM dictionary is missing or in improper format

3.3.25 PCBFID

This routine returns the FID of the PCB for the process as a string of four hexadecimal digits in the TS work space.

Input Interface

```
TSBEG     S     Points one before the area where the
                returned value is to be stored
```

Output Interface

```
TS        R  + Point to the last character of the
TSEND     S  + returned value, at TSBEG+1

R15       R     Points to a SM placed at TS+1
```

Element Usage

D0

3.3.26 PCRLF

FFDLY

PCRLF prints a carriage return and line feed on the terminal and enters FFDLY, which prints a specified number of delay characters (X'00').

Input Interface

LFDLY	H	Contains the delay count (for PCRLF only)
T0	T	Contains the delay count (for FFDLY only)

Output Interface

None

Element Usage

R14	R

3.3.27 PINIT

PINIT is used for process initialization. Pointers are set up to all work spaces; links are set up in frames of linked work spaces (HS, IS, OS, and PROC). All elements in the primary, secondary, and tertiary (DEBUG) control blocks are zeroed, except as noted below.

Input Interface

R0	R	Points to the PCB of the process to be initialized

Output Interface

R2	R	Points to the process's SCB (PCB+1)
HS	R	+ he beginning of the HS work
HSBEG	S	+ space (PCB+10)
HSEND	S	+
IS	R	+ POINT TO THE BEGINNING OF THE IS work
ISBEG	S	+ space (PCB+16)
ISEND	S	+
OS	R	+ Point to the beginning of the OS work
OSBEG	S	+ space (PCB+22)
OSEND	S	+
IBSIZE	T	=140
OBSIZE	T	=100
TTLY	T	=0 (For DEBUG use)
INHIBIT	B	=1

other elements as initialized by wsinit.

Address registers, and the PCB elements PRMPC, SC0, SC1, and SC2 (all characters) are not zeroed. In addition, the tertiary control block is initialized for the debugger by setting the corresponding INDEBUG bit to 1, and setting the corresponding R1 and return stack elements to execute debugger code.

Element Usage

(Functional elements initialized as described)

Subroutine Usage

WSINIT (local), LINK

Three additional levels of subroutine linkage required

3.3.28 PONOFF

PONOFF is used to reverse the setting of bit LISTFLAG before entering the WRAPUP processor. When LISTFLAG is set, all output to the terminal is suppressed by the standard output routines (see WRTLIN documentation). After reversing this bit, PONOFF exits to MD99.

3.3.29 PPUT (1,SPOOLADD)*

PPUT is used to output a line of data to the spooler process, which will then print it on the line printer or take other action depending on the process's entry in the spool assignment table (see spooler documentation).

Input Interface

OBBEG	S	Points one before the first character of the output data
OB	R	Points to the last character of the output data
NOBLNK	B	if set, the output buffer is not filled with blanks after the data is output

Output Interface

OB	R	=OBBEG
RMODE	T	=0 if processing is aborted due to no more overflow space available

The output buffer is filled with blanks (through the address originally pointed to by OB) unless NOBLNK is set

Element Usage

```
R8          R  +
R14         R  +
R15         R  +
INHIBITSV1  B  + Utility
CH0         C  +
D1          D  +
RECORD      D  +

OVRFLW      D    Used if ATTOVF is called
```

Subroutine Usage

ASG.TBL; two local subroutines; ATTOVF if more overflow space is needed to store data; 2,SPOOLINIT and CHANCE2 if ATTOVF cannot find any more space

Three additional levels of subroutine linkage required

Exits

To LINE if line-at-a-time spooler output is specified in the assignment table entry; to MD999 if processing aborted due to no more overflow space available

3.3.30 PRIVTST1 - PRIVTST2 - PRIVTST3

These routines check to see if the calling process has appropriate system privilege levels. If not, bits PQFLG and LISTFLAG and tally RMODE are set to zero, the history string is set null (HSEND=HSBEG), tally REJCTR is set to 82 (an error message number), and an exit is taken to MD99. Otherwise the routines return normally.

Entry	Bit tested (error if not set)
PRIVTST1	SYSPRIV1
PRIVTST2	SYSPRIV2
PRIVTST3	R0;B245

3.3.31 PRNTHDR

NPAGE

These are entry points into the system routine for pagination and heading control of output (also used by WRTLIN, WT2, and WRITOB when pagination is specified). PRNTHDR is used to initialize bit PAGINATE to 1, and tallies LINCTR and PAGNUM to zero and one, respectively. PRNTHDR then falls immediately into NPAGE, which outputs a header message.

A page heading, if present, must be stored in a buffer defined by register PAGHEAD. The header message is a string of data terminated by a SM; system delimiters in the message invoke special processing as follows:

SM (X'FF')	Terminates the header line with a CR/LF
AM (X'FE')	Inserts the current page number into the heading
VM (X'FD')	Prints one line of the heading and starts a new line
SVM (X'FC')	Singly, inserts the current time and date into the heading, but two SVM's in succession insert the date only
SB (X'FB')	Inserts data from one of various buffers into the heading; if the character following the SB is 'I', data is copied

from the area beginning one byte past the address specified by register BMSBEG; if the character is 'A', register AFBEG is used; for any other character, data is copied from the area beginning three bytes past the address specified by register ISBEG; data to be copied can be terminated by any system delimiter

Carriage returns, line feeds, and form feeds should not be included in header messages, or the automatic pagination will not work properly.

Input Interface

PAGINATE	B	=1 (NPAGE only; set automatically by PRNTHDR)
LINCTR	T	Contains the number of the line to be printed on the current page (NPAGE only; set to zero automatically by PRNTHDR)
PAGNUM	T	Contains the current page number (NPAGE only; set to one automatically by PRNTHDR)

Other parameters as for WT2 (see WRTLIN dcumentation),xcept for PAGINATE and PAGNUM (see above) and OB (initialized to OBBEG by NPAGE); note that the buffer where the translated heading message is built (specified by register OBBEG) must be at least two bytes greater than the longest line output in the translated heading (not necessarily the total heading size, if the original heading string contains any VMs), in order to accomodate a trailing crlf.

Output Interface

Same as for WT2

Element Usage

Same as for WT2

Subroutine Usage

Same as for WT2

Exits

To WT2

3.3.32 PROC User Exits PROC User Exits

Summary

A user-written program can gain control during execution of a PROC by using the Uxxxx or Pxxxx command in the PROC, where "xxxx" is the hexadecimal mode-id of the user routine. The routine can perfor special procsing, and then return control to the PROC processor. Necessarily, certain elements used by the PROC processor are maintained by the user program; these elements are marked with an asterisk in the table below.

Input Interface

*BASE	D	+	Contain the base, modulo, and separation
*MODULO	T	+	of the master dictionary
*SEPAR	T	+	
*PQBEG	S		Points one prior to the first PROC statement
*PQEND	S		Points to the terminal AM of the PROC
PQCUR	S	+	Point to the AM following the Uxxxx or
IR	R	+	Pxxxx statement
*PBUFBEG	S		Points to the buffer containing the primary and secondary (if any) input buffers; buffer format is SB ... Primary input ... SM SB ... Secondary input ... SM
*ISBEG	S		Points to the buffer containing the primary output line
*STKBEG	S		Points to the buffer containing "stacked input" (secondary output)
IB	R		Is the current input buffer pointer (may point within either the primary or secondary input buffers)
*SR35	S		Points to the beginning of the current input buffer
*SBIT	B		Set if a ST ON command is in effect
*ZBIT	B		Reset to identify the PROC processor in certain system subroutines
*SC2	C		Contains a blank

		SBIT on	SBIT off
IS	R	Points to the last byte moved into the secondary output buffer	Points to the last byte moved into the primary output buffer
UPD	R	Points to the last byte moved into the primary output buffer	Points to the last byte moved into the secondary output buffer

Output Interface

IR	R	Points to the AM preceding the next PROC statement to be executed; may be altered to change PROC execution

IS	R	+ May be altered as needed to alter data
UPD	R	+ within the input and output buffers, but
IB	R	+ the formats described above must be maintained

Exit Convention

The normal method of returning control to the PROC processor is to execute an external branch instruction (ENT) to 2,PROC-I. To return control and also reset the buffers to an empty condition, entry 1,PROC-I may be used. If it is necessary to abort PROC control and exit to WRAPUP, bit PQFLG should be reset before branching to any of the WRAPUP entry points (see WRAPUP documentation).

Note that when a PROC eventually transfers control to TCL (via the "P" operator), certain elements are expected to be in an initial condition. Therefore, if a user routine uses these elements, they should be reset before returning to the PROC, unless the elements are deliberately set up as a means of passing parameters to other processors. Specifically, the bits ABIT through ZBIT are expected to be zero be the TCL-II and ENGLISH processors. It is best to avoid usage of these bits in PROC user exits. Also, the scan character registers SC0, SC1, and SC2 must contain a SB, a blank, and a blank, respectively.

3.3.33 PRTERR

PRTERR is used to retrieve and print a message from the system file ERRMSG. A parameter string may be passed to the routine, in which case the parameters are formatted and inserted according to the codes in the message item.

Items in the ERRMSG file consist of an arbitrary number of lines (where a line is delimited by an AM), with each line containing a code letter in column one, possibly followed by a string or numeric parameter (numeric parameters enclosed in parentheses). The possible codes and their meanings are listed below. (Brackets indicate optional parameters.)

A [(dec. #)]	Parameter insertion code; the next parameter from the parameter string, if any, is placed into the ouput buffer; if "dec. #" Is specified, the parameter is left-justified in a blank field of that length
R [(dec. #)]	Like A, only the parameter is right-justified, in a field of "dec. #" Blanks if "dec. #" Is specified
H string	The character string is placed in the output buffer (no blank is necessary between the code letter and the beginning of the string)
E [string]	The message item-id, surrounded by brackets, is placed into the outpur-t bu
L [(dec. #)]	The output buffer is printed, and the specified number of line feeds is output (one if "dec. #" Is not specified)

S [(dec. #)]		The pointer to the current position in the output buffer is repositioned to the specified column (column one if "dec. #" Is not present)
X (dec. #)		The pointer to the current position in the output buffer is incremented by the specified number of spaces; if the end of a line is reached (see below), the buffer is printed and a new line is started
T		The system time in HH:MM:SS is added to the output buffer
D		The system date in DD MMM YYYY format is added to the output buffer

Input Interface

TS	R	Points one prior to the message item-id, which must be terminated by an AM; parameters optionally follow, being delimited by AM's; the parameter string must end with a SM
EBASE EMOD ESEP	D + T + T +	Used as the base, modulo, and separation for the message file if EBASE is non-zero; if EBASE is zero, PRTERR attempts to set EBASE, EMOD, and ESEP to the parameters for the system file ERRMSG, and exits abnormally if unable to do so
MBASE MMOD MSEP	D + T + T +	Used as the parameters for the master dictionary if necessary to set up EBASE, EMOD, and ESEP, but PRTERR exits abnormally if MBASE is zero
OBSIZE	T	Contains the maximum number of characters to be output on a line (normally set at logon time)
OBBEG OBEND	S + S +	Point to the beginning and end of the output buffer (normally set at logon time)

Other elements as required by WRTLIN (see WRTLIN documentation)

Output Interface

TS	R	Points to the AM after the message item-id if no parameters are processed, otherwise to the AM or SM after the last parameter processed
EBASE EMOD ESEP	D + T + T +	Contain the base, modulo, and separation parameters for the system file ERRMSG if EBASE was originally zero (and the file was successfully retrieved)

```
LINCTR    T  + Updated if bit PAGINATE is set
PAGNUM    T  +
```

Element Usage

```
SB60      B  +
SB61      B  +
CTR0      T  +
T6        T  +
BASE      D  +
MODULO    T  +
SEPAR     T  + Utility
AF        R  +
IR        R  +
BMS       R  +
BMSBEG    S  +
OB        R  +
R14       R  +
SR4       S  +

CTR1      T    Used with "R" code messages

SYSR1     S    Used with "S" code messages

INHIBIT   B    Set during retrieval  of file ERRMSG, if
               EBASE  is  originally  zero,  and  reset
               afterwords to the value on entry
```

All elements used by WRTLIN (unless PRTERR exits abnormally), and elements used by GBMS if PRTERR attempts retrieval of the system file ERRMSG

Subroutine Usage

RETIX, WRTLIN, TILD, DATE (for "D" code messages), TIME (for "T" code messages), GBMS (for retrieving ERRMSG)

Six additional levels of subroutine linkage required if GBMS attempts retrieval of an ERRMSG file which is a "Q" code item, otherwise four levels required

Exits

To 2,ABSL if EBASE and MBASE are both zero

3.3.34 RELBLK - RELCHN - RELOVF

These routines are used to release frames to the overflow space pool. RELOVF is used to release a single frame, RELBLK is used to release a block of contiguous frames, and RELCHN is used to release a chain of linked frames (which may or may not be contiguous). A call to RELCHN specifies the first FID of a linked set of frames; the routine will release all frames in the chain until a zero forward link is encountered.

Input Interface

```
OVRFLW    D    Contains  the FID of  the  frame  to  be
               released (for RELOVF),  or the first FID
               of the  block or  chain  to  be released
               (for RELBLK and RELCHN, respectively)
```


D0	D	Contains the number of frames (block size) to be released, for RELBLK only

Output Interface

None

Element Usage

```
OVRFLW    D  +
R14       R  + Utility
R15       R  +

D0        D  +
D1        D  + Used by SYSREL
D2        D  +
```

Subroutine Usage

SYSREL; two internal subroutines

Two additional levels of subroutine linkage required

3.3.35 RETI RETIX RETIXU

These are the entry points to the standard system routine for retrieving an item from a file. The item-id is explicitly specified to the routine, as are the file parameters base, modulo, and separation. Additionally, the number of the first frame in the group in which the item may be stored must be specified if the entry RETIXX is used. The other entries perform a "hashing" algorithm to determine the group (see HASH documentation). The group is searched sequentially for a matching item-id. If the routine finds a match, it returns pointers to the beginning and end of the item, and the item size (from the item count field). If entry RETIXU is used, the group is locked during processing, preventing other programs from accessing (and possibly changing) the data.

The item-id is specified in a buffer defined by register BMSBEG; if entry RETI is used, register BMS must point to the last byte of the item-id, and an AM will be appended to it by the routine. For all other entry points, the item-id must already be terminated by an AM.

Input Interface

```
BMSBEG    S     Points one byte before the item-id

BMS       R     Points to the last character of the
                item-id, for RETI, RETIXX, and UPRETIX
                only
BASE      D  +  Contain the base, modulo, and separation

MODULO    T  +  of the file to be searched
SEPAR     T  +

RECORD    D     Contains the beginning FID of the group
                to be searched, for RETIXX only
```

Output Interface

BMS	R	+ Point to the last character of the
BMSEND	S	+ item-id
RECORD	D	Contains the beginning FID of the group to which the item-id hashes (set if HASH is called)
NNCF	H	+
FRMN	D	+ Contain the link fields of the frame
FRMP	D	+ specified in RECORD; set by RDREC
NPCF	H	+
XMODE	T	=0

		Item Found:	Item Not Found:
RMBIT	B	=1	=0
SIZE	T	=value of item count field	=0
R14	R	Points one prior to the item count field	Points to the last AM of the last item in the group
IR	R	Points to the first AM of the item	Points to the AM indicating end of group data (=R14+1)
SR4	S	Points to the last AM of the item	=R14

Element Usage

None (except D0, D1, and R15)

Subroutine Usage

RDREC (local), HASH (except for RETIXX; local), GLOCK (RETIXU only), IROVF (for IR overflow space handling and error conditions)

Three additional levels of subroutine linkage required (for IROVF and GLOCK; RDREC and HASH require one level)

Exits

If the data in the group is bad - premature end of linked frames, or non-hexadecimal character encountered in the count field - the message

GROUP FORMAT ERROR xxxxxx

is returned (where xxxxxx is the FID indicating where the error was found), and the routine returns with an "item not found" condition. Data is not destroyed, and the group format error will remain.

3.3.36 SETLPTR - SETTERM

These routines are used to set output characteristics such as line width, page depth, etc., to the previously-specified values for either the terminal or the line printer. In addition, the current line number and page number are saved so that when switching from terminal to line printer output, say, and then switching back, pagination will continue automatically from the previous values.

Input Interface

LPBIT	B	Reset by SETTERM; set by SETLPTR
LINCTR	T	Contains the current line number
PAGNUM	T	Contains the current page number
OBSIZE	T	Contains the size of the OB buffer
TPAGSIZE or PPAGSIZE	T T	Contains the number of printable lines per page for the terminal or line printer
TOBSIZE or POBSIZE	T T	Contains the size of the output (OB) buffer for the terminal or line printer
TLINCTR or PLINCTR	T T	Contains the current line number for the terminal or lineprinter
TPAGNUM or PPAGNUM	T T	Contains the current page number for the terminal or line printer

Note: TPAGSIZE, TOBSIZE, TLINCTR, and TPAGNUM are required only by SETTERM; PPAGSIZE, POBSIZE, PLINCTR, and PPAGNUM are required only by SETLPTR

Output Interface

```
PAGSIZE    T  +
OBSIZE     T  + set  to the appropriate  characteristics
LINCTR     T  + for terminal or line printer output
PAGNUM     T  +

TLINCTR    T    =LINCTR; TLINCTR set by SETLPTR, PLINCTR
or              set by SETTERM
PLINCTR    T

OBSIZE     T    =79 if originally zero

R14        R  + =OBBEG+OBSIZE
OBEND      S  +
```

The area from the address pointed to by OBBEG to that pointed to by Obend is filled with blanks

3.3.37 SETUPTERM

This routine sets the default values for terminal and line printer characteristics (as used by INITTERM).

Input Interface

BSPCH	C	Contains the character to be echoed for a backspace
LFDLY	T	Contains the number of "fill" characters to be output after a CR/LF in the lower byte; if the upper byte is greater than one, a form feed is output before each page of paginated output, and that number of "fill" characters is output
TOBSIZE	T	Countains the terminal line width
TPAGSIZE	T	Contains the terminal page depth
POBSIZE	T	Contains the printer line width
PPAGSIZE	T	Contains the printer page depth
PAGSKIP	T	Contains the number of lines to be skipped at the bottom of each page

Output Interface

Default values initialized as described

3.3.38 SLEEP - SLEEPSUB SLEEP - SLEEPSUB

These routines cause the calling process to go into an inactive state for a specified amount of time. If SLEEPSUB is used, either the amount of time to sleep or the time at which to wake up may be specified.

Input Interface

D0	D	Contains the number of seconds to sleep, up to 86400 (one day), or, for SLEEPSUB, the time to wake up (number of seconds past midnight) if RMBIT is reset
RMBIT	B	For SLEEPSUB only, set if D0 contains the number of seconds to sleep, and reset if it contains the time to wake up

Output Interface

None

Element Usage

T2	T	+ Used by SLEEPSUB only, on a monitor call
D2	D	+ to get system time

Subroutine Usage

SLEEP used by SLEEPSUB

One additional level of subroutine linkage required by SLEEPSUB, none by SLEEP

3.3.39 SORT SORT

This routine sorts an arbitrarily long string of keys in ascending sequence only; the calling program must complement the keys if a descending sort is required. The keys are separated by SM's when presented to SORT; they are returned separated by SB's. Any character, including system delimiters other than the SM and SB may be present within the keys.

An n-way polyphase sort-merge sorting algorithm is used. The original unsorted key string may "grow" by a factor of 10%, and a separate buffer is required for the sorted key string, which is about the same length as the unsorted key string. The "growth" space is contiguous to the end of the original key string; the second buffer may be specified anywhere. SORT automatically obtains and links overflow space whenever needed. Due to this, one can follow standard system convention and build the entire unsorted string in an overflow table with OVRFLCTR containing the beginning FID; the setup is then:

```
start of        end of           "growth"   start of
unsorted keys   unsorted keys      space    second buffer
<----------/-   -/---------->  <------------><----------/-
```

The second buffer pointer then is merely set at the end of the "growth" space, and SORT is allowed to obtain additional space as required.

Alternately, the entire set of buffers may be in the IS or OS workspace if they are large enough.

Input Interface

SR1	S	Points to the SM preceding the first key
SR2	S	Points to the SM terminating the last key
SR3	S	Points to the beginning of the second buffer

Output Interface

SR1	S	Points before the SB preceding the first sorted key (the exact offset varies from case to case); the end of the sorted keys (separated by SB's) is marked by a SM

Element Usage

HBIT	B	+
LBIT	B	+
SB1	B	+
SC2	C	+
XMODE	T	+
D0	D	+
IS	R	+

```
OS        R  +
BMS       R  +
TS        R  + Utility
CS        R  +
R14       R  +
R15       R  +
S1        S  +
S2        S  +
S3        S  +
S5        S  +
S7        S  +
S8        S  +
S9        S  +
```

Subroutine Usage

COMP

GWS used with XMODE

Four additional levels of subroutine linkage required

3.3.40 TCL-II MD200 MD201 TCL-II MD200 MD201

These are the entry points (not subroutines) into the TCL-II processor, used whenever a verb requires access to a file, or to all or explicitly specified items within a file. MD200 is entered from the TCL-I processor after decoding the verb (primary mode-id = 2). MD201 is used by TCL-II itself to regain control from WRAPUP under certain conditions (see below). TCL-II exits to the processor whose mode-id is specified in MODEID2; typically processors such as the EDITOR, ASSEMBLER, LOADER, etc. Use TCL-II to feed them the set of items which was specified in the input data.

On entry, TCL-II checks the verb difinition for a set of option characters in attribute 5; verb options are single characters in any sequence and combination, and are listed below (all other characters are ignored).

Option	Meaning
C	Copy - items retrieved are copied to the IS workspace
E	Expand - items retrieved are expanded and copied to the IS work space (see EXPAND documentation); ignored if the "C" option is not present
F	File access only - file parameters are set up but any item-list is ignored by TCL-II; if this option is present, any others are ignored
N	New npm acceptable - if the item specified is not on gile, the secondary processor still gets control (the EDITOR, for example, can process a new item)

P		Print - on a full file retrieval (all items), the item-id of each item is printed as it is retrieved
U		Updating sequence flagged - if items are to updated as retrieved, this option is mandatory
Z		Final entry required - the secondary processor will be entered once more after all items have been retrieved (the COPY processor, for instance, uses this option to print a message)

The input data string to TCL-II consists of the file-name (optionally preceded by the modifier "DICT", which specifies access to the dictionary of the file), followed y a list of items, or an asterisk ("*") specifying retrieval of all items in the file. The item-list may be followed by an option list (options for the secondary processor), which must be enclosed in parentheses; see GETOPT documentation for further information about options.

Input Interface

IR	R	Points to the AM before attribute 5 of the verb
SR4	S	Points to the AM at the end of the verb
MODEID2	T	Contains the mode-id of the processor to which TCL-II transfers control (assuming no error conditions are encountered)
BMSBEG	S	Points one prior to an area where the file name is to be copied, if the "F" option is present, otherwise one prior to an area where item-ids are to be copied
ISBEG	S	Points one prior to an area where items are to be copied, if the "C" option is present

Elements as required by GETFILE

Output Interface

DAF1	B	Set if the "U" option is specified
DAF2	B	Set if the "C" option is specified
DAF3	B	Set if the "P" option is specified
DAF4	B	Set if the "N" option is specified
DAF5	B	Set if the "Z" option is specified
DAF6	B	Set if the "F" option is specified, or if a full file retrieval is specified (no "F" option)

DAF10	B		Set if more than one item is specified in the input data, but not a full file retrieval ("*")
DAF11	B		Set if the "E" option is specified

Note: the above bits are not initialized to zero

DAF8	B		Set if a file dictionary is being accessed, otherwise reset (from GETFILE)
DAF9	B		=0
IS	R		Points one past the end of the file name in the input string if the "F" option is present; points to the last AM in the copied item if the "C" option is present, otherwise to the end of the input string
ISBEG	S	+	Unchanged
BMSBEG	S	+	
RMBIT	B		Set if the file is successfully retrieved if the "F" option is present
SBASE	D	+	Contain the base, modulo, and separation
SMOD	T	+	of the file being accessed
SSEP	T	+	
BASE	D	+	=SBASE, SMOD, SSEP on the first exit
MODULO	T	+	only (from MD200)
SEPAR	T	+	
DBASE	D	+	Contain the base, modulo, and separation
DMOD	T	+	of the dictionary of the file being
DSEP	T	+	accessed if the "F" option is present
SC0	C		Contains a SB if the last item-id in the input string is enclosed in quote marks, otherwise contains a blank

The following specifications are meaningful only when the "F" option is not present:

SR0	S	Points one prior to the count field of the retrieved item
SIZE	T	Contains the value of the count field of the retrieved item
SR4	S	Points to the last AM of the retrieved item
ISEND	S	=IS if the "C" option is present

IR	R	Points to the last AM of the retrieved item to be copied, if the "C" option is present, otherwise points to the AM following the item-id
RMODE	T	=MD201 if items are left to be processed, otherwise=0
XMODE	T	=0

Elements as set up by GETOPT if the input data contains an option string

Element Usage

C1	T	Used for error messages

Elements used by the various subroutines below

Subroutine Usage

GETFILE; if no "F" option: GETOPT if the input data contains an option string, GETITM for full file retrieval, RETIX and one internal subroutine if not full file retrieval, GETSPC if more than one item (but not "*") specified, EXPAND if the "E" option is present, WRTLIN if the "P" option is present

MD201 only: WSINIT; GNTBLI if more than one item (but not "*") specified

MD995 and BMSOVF used with XMODE

Seven additional levels of subroutine linkage required by MD200; five additional levels required by MD201 for full file retrieval, otherwise three levels required

Error Conditions

The following conditions cause an exit to the WRAPUP processor with the error number indicated:

Error	Condition
13	DL/ID item not found, or in bad format
199	IS work space not big enough when the "C" option is specified
200	No file name specified
201	File name illegal or incorrectly defined in the M/DICT
202	Item not on file; all messages of this type are stored until all items have been processed; items which are on file are still processed
203	No item list specified
209	The format of the option list is bad

3.3.41 TIME - DATE - TIMDATE

These routines return the system time and/or the system date, and store it in the buffer area specified by register R15. The time is returned as on a 24-hour clock.

Entry	Buffer size required (bytes)	Format
TIME	9	HH:MM:SS
DATE	12	DD MMM YYYY
TIMDATE	22	HH:MM:SS DD MMM YYYY

Input Interface

R15	R	Points one prior to the buffer area

Output Interface

R15	R	Points to the last byte of the data stored; the byte immediately following contains a blank
R14FID	D	=0 (DATE and TIMDATE only)

Element Usage

```
D0     D  +
D1     D  + Used by TIME and TIMDATE only
D2     D  +
D3     D  +
```

Subroutine Usage

TIME used by TIMDATE; MBDSUB used by TIME

Two additional levels of subroutine linkage required by TIMDATE, one level required by TIME, none by DATE

3.3.42 TPREAD TPWRITE

TPREAD reads a specified number of bytes from the tape into a buffer pointed to by R15 at entry to the routine.

TPWRITE writes a specified number of bytes from the buffer pointed to by R15 to the tape.

Both TPREAD and TPWRITE are using a virtual tapedrive with common routines. The initial execution of either entry point causes initialization of two buffers of a size sufficient to contain TPRECL, which is assigned during execution of the T-ATT verb, or is obtained by execution of the RDLBL verb from the tape record size included i the standard R77 tape label. These buffers are released during WRAP-UP processing after RMODE and WMODE processing are completed. The process then returns to TCL or the CHAIN or PROC analogs to TCL.

At all times after initialization R7 points into the current ad or write location in the tape buffers and must be saved and restored if R7 is to be used for other purposes between reads or writes. In both cases the contents of the accumulator, D0, is the number of characters to transfer to or from the tape buffer. The alignment of R7 in the buffer and the relative size of TPRECL and D0 do not need to be considered.

If D0 is zero on a read, then TPREAD will return to the calling routine with R7 pointing one before the next string to be read, XMODE will be set to the tape handler routine, and the old XMODE, if any, will be in YMODE. This allows transparant tape reading using MIID or MIIT R7,XX. A forward link zero fault on R7 will cause the next tape record to be read into the last buffer, R7 to be reset to the beginning of the current buffer; and execution then continues in the MII instruction. The user is responsible for handling an end-of-file condition when reading the tape. When this occurs, the EOFBIT will be set.

If D0 is zero on a write, then TPWRITE will fill the rest of the tape buffer with the character pointed to by R15, which will cause the buffer to be written to tape. This is recommended in order to send the last partial tape record to the tape, after which WEOF should be executed.

Input Interface

ATTACH	B	Must be set. Use T-ATT verb.
TPRECL	T	As above.
R15	R	Points to one byte before the source or destination buffer start location.
R7	R	Must be the same at the beginning of the next tape operation as it was at the end of the last tape operation. Initialized by TPREAD TPWRITE on first-time call.
D0	D	Co�2Yns the number of bytes to be transferred to or from the tape buffers.

Output Interface.

R15	R	Points at the end of the source or destination buffer if D0 was non-zero; unchanged if D0 was zero.
D0	D	Is zero.
EOFBIT	B	Indicates end-of-file on read if set.
EOTBIT	B	Indicates end-of-tape if set; the tape handler will rewind the tape and tell the operator to mount the next tape, however. This may be executed in the middle of an MII instruction, as above, which will then continue to execute when the new reel in mounted and the label handled.

Element Usage.

The tape handler will stack and restore most of the elements which it uses. The following elements are modified, however.

T5	T	"
T6	T	"
T7	T	"
YMODE	T	For any current XMODE
D2	D	Temporary strage

R7	R	As the tape buffer pointer
R2;H0	H	For a flag
R4	R	Is used as a pointer to the text block in the write-label routine.
R14	R	Globally
R15	R	As noted above.

Subroutine usage.

TPREAD and TPWRITE use an extensive set of internal subroutines in such a way that element usage is transparant outside of the above set. Both may go to seven levels of subroutine usage if either encounters a parity error while handling a label on the second and following reels in a set of tapes.

Error conditions are sent to the terminal by the tape handler by means of the PRINT, CRLFPRINT and PCRLF routines for attention by the operator in a manner transparant to the calling routine. They include no write ring, parity error after ten retries, tape not ready, and block transfer incomplete messages and recovery alternatives.

3.3.43 TSINIT

This routine initializes the register triad associated with the TS work space.

Input Interface

None

Output Interface

```
TS         R  + Point  to the beginning  of the  TS work
TSBEG      S  + space (PCB+5)
(R14       R) +

TSEND      S  + Point  to the last  byte of the  TS work
(R15       R) + space (511 bytes past TSBEG);  note this
                is an unlinked work space
```

the first byte of the work space is set to x'00'.

Element Usage

D0

Subroutine Usage

One internal subroutine

One additional level of subroutine linkage required

3.3.44 UPDITM - UPDITMX

UPDITM and UPDITMX perform updates to a disc file defined by its base FID, modulo, and separation. If the item is to be deleted, the routines compress the remainder of the data in the group in which the item resides; if the item is to be added, it is added at the end of the current data in the group; if the item is to be replaced, it is replaced in place, sliding the remaining items in the group to the left or right as necessary.

If the update causes the data in the group to reach the end of the linked frames, NEXTOVF is entered to obtain another frame from the overflow space pool and link it to the previous linked set; as many frames as required are added. If the deletion or replacement of an item causes an empty frame at the end of the linked frame set, and that frame is not in the "primary" area of the group, it is released to the overflow space pool.

Entry UPDITM uses PRETIXU to retrieve the item to be updatedlocking the group.

Once item is retrieved, processing cannot be interrupted until completed.

Input Interface

BMSBEG	S		Points one prior to the item-id of the item to be updated; the item-id must be terminated by an AM
TS	R		Points one prior to the item body to be added or replaced (no item-id or count field); not needed for deletions; the item body must be terminated by a SM
CH8	C		Contains the character 'D' for item deletion; 'U' for item addition or replacement; 'G' if no group unlock.
BASE	D	+	Contain the base, modulo, and separation
MODULO	T	+	of the file being updated
SEPAR	T	+	

The following specifications are meaningful only for UPDITMX:

RMBIT	B	Set if the item to be updated exists in the file, otherwise reset
R14	R	Points one prior to the item count field if the item exists, otherwise points to the last AM of the last item in the group
RECORD	D	Contains the beginning FID of the group containing the item

Output Interface

Remainder of the last frame in the group filled with blanks

Element Usage

D3	D	+	
D4	D	+	
NNCF	H	+	Utility
FRMN	D	+	
FRMP	D	+	
NPCF	H	+	

Elements used by the various subroutines below

Subroutine Usage

RDREC; HASH, GLOCK, and RETIXU RELCHN if overflow frames returned; WTLINK if data ends in the last frame of "prime" space, or in overflow space; COPYALL if the item is on file; BKUPD; GUNLOCK

NEXTOVF, BMSOVF, and IROVF used with XMODE

Four additional levels of subroutine linkage required by UPDITM, three by UPDITMX

Error Conditions

1. If the group data is bad (premature end of linked frames, or non-hexadecimal character found in an item count field), IROVF is entered to print a warning message, and the group data is terminated at the end of the last good item before processing continues

3.3.45 WHOSUB

This routine returns the line number and current account name associated with the process as a string in the TS work space.

Input Interface

TSBEG	S	Points one before the area where the returned string is to be stored
BMSBEG	S	Points one before an area which RETIX can use in retrieving an item from the system file ACC

Output Interface

TSBEG	S	Points one before the returned string, which consists of the line number (in decimal digits), a space, and the account name as found in the system file ACC for the associated PCB; if the ACC entry is not found, "UNKNOWN" is returned
TS	R	Points to the last character in the returned string
TSEND	S	Points to a SM placed at TS+1
D3	D	Contains the line number associated with the process
BMSBEG	S	Points one before the item-id used in accessing the ACC file, if the file is present; the item-id consists of four characters representing the PCB in hexadecimal digits

BMS	R	Points to the last character of the above item-id if the ACC file is present; set by RETI
RMBIT	B	Set if the ACC file is present and the appropriate item is found, otherwise reset

Element Usage

R15	R	+ Utility
S4	S	+
T4	T	+
T5	T	+
D0	D	+ Used by MBDSUB
D1	D	+
D2	D	+
R14	R	+
BASE	D	+
MODULO	T	+
SEPAR	T	+ Used by GETACBMS
T6	T	+
BMS	R	+
SR1	S	+
RECORD	D	+
NNCF	H	+
FRMN	D	+
FRMP	D	+
NPCF	H	+ Used by RETI (and GETACBMS if the ACC
XMODE	T	+ file is a "Q" item)
DAF9	B	+
SIZE	T	+
IR	R	+
SR4	S	+

Subroutine Usage

LINESUB; MBDSUB; GETACBMS; GPCB0 if the ACC file is found; RETI if the ACC entry for the process is found

Five aditional levels of subroutine linkage required

3.3.46 WRAPUP PROCESSOR WRAPUP PROCESSOR

MD99 MD993 MD994 MD995 MD999

These are the entry points into the system routine which "wraps up" the processing initiated by a TCL statement, performs disk updates and prints messages as required, and reinitializes functional elements for processing another TCL statement. WRAPUP may also be treated as a subroutine (except when entered at TCLXIT or NSPCQ) by setting tally RMODE to the mode-id of the routine to which WRAPUP should return control after it is done. Note, however, that WRAPUP always set the return stack to a null or empty condition before exiting.

The various entry points are provided to simplify the interface requirements when WRAPUP is used to store or print messages from the ERRMSG file; the features of each can be seen in the following table:

MD993	C1 contains a message number; C2 contains a numeric parameter; the value in C1, converted to an ASCII string, is used as the item-id of an item to be retrieved from the message file (normally ERRMSG); the message is set up in the history string (see below), and control passes to MD99
MD994	C1 contains a message number; IS points one before the beginning of a string parameter, which is terminated by an AM or SM; the message is set up in the history string and control passes to MD99
MD995	Like MD994, except the string parameter is stored at BMSBEG+1 through an AM or SM
MD99	Message numbers (without any parameters) may be stored in REJCTR, REJ0, and REJ1 (no action is taken if zero); if RMODE is zero, messages are printed regardless of the value of VOBIT (see below); the messages are set up in the history string and control passes to MD999
MD999	The history string is processed, and process work spaces are reinitialized; control passes to TCL if RMODE is zero, otherwise to the routine specified by RMODE
TCLXIT	The history string is set null, PROC control is unconditionally reset, and control passes to TCL (this entry point is used by the DEBUG "END" command)
NSPCQ	In addition to the functions performed at TCLXIT, all disk group locks associated with the process are unlocked, and the overflow management routine in mode OF1 is unlocked if currently locked by the process

Input Interface

HSBEG	S	+	Point one before the beginning and to
HSEND	S	+	the end, respectively, of the history string; if HSBEG=HSEND, the string is null

Three types of history string elements are recognized by WRAPUP; all others are ignored. The type of processing done for each element depends on the second, and possibly third character of the element string. (The quote marks in the following examples are not part of the strings.)

1. Output message

   SM "O" AM message-id AM (parameter AM...) SM

   where "message-id" is the item-id (normally a decimal numeric) of an item in the message file

   The parameter string is passed to PRTERR for message formatting (see PRTERR documentation)

2. Disk Update/Delete

   SM "DU" AM base VM modulo VM separation AM item-id
   AM item-body AM SM

   SM "DD" AM base VM modulo VM separation AM item-id
   AM SM

   where "DU" causes the item in the file specified by "base", "modulo", and "separation" to be replace, and "DD" deletes it

3. (End of history string)

   SM "Z"

Conventionally, a process wishing to add data to the history string begins at HSEND+1; after the additional elements have been added, the string is terminated (once again) by a SM and "Z", and HSEND is set pointing to this SM.

WMODE	T	If non-zero, the value is used as the mode-id for an indirect subroutine call (BSLI *) executed immediately after the history string has been processed, and before work space and printer characteristics are reset; this allows special processing to be done on any entry into WRAPUP
RMODE	T	If non-zero, WRAPUP exits to the specified mode-id instead of to TCL
VOBIT	B	If set, and RMODE is non-zero, messages are stored in the history string, for output on a later entry into WRAPUP with RMODE zero
REJCTR REJ0 REJ1	T + T + T +	May contain message numbers which do not require parameters; REJCTR is always tested first, then REJ0, and then REJ1; no action is taken on a zero value; a value of 9999 is used internally by

WRAPUP to identify which messages have been processed, and should not normally be used as an input value for REJ0 or REJ1

```
C1         T  + (See MD993, MD994, and MD995 above)
C2         T  +
```

LPBIT B — If set, all open spool files are closed

OVRFLCTR D — If non-zero, used as the starting FID of a linked set of overflow frames which is released to the system overflow space pool; used by SORT, for instance, to store the beginning FID of a sorted table, in which case the overflow space used by SORT is always released, even if processing is aborted by an "END" command from DEBUG

USER T — Used to control the final exit from WRAPUP when RMODE=0; see "exits"

Output Interface

HSEND S — =HSBEG except when messages are stored instead of printed

```
VOBIT      B  +
LPBIT      B  +
WMODE      T  + =0
REJCTR     T  +
REJ0       T  +
REJ1       T  +
```

Return stack — Null: RSEND=X'01B0', RSCWA=X'0184', and the rest of the return stack is filled with X'FF'

RMODE T — Set to zero by TCLXIT and NSPCQ

INHIBIT B — Set to zero by NSPCQ

Elements as initialized by WSINIT (and ISINIT if RMODE=0)

The following elements are set up only if RMODE=0:

```
XMODE      T  + =0
OVRFLCTR   T  +

IBSIZE     T    =140
```


Element Usage

```
UPD        R

BASE       D  +
MODULO     T  + Used in disk updates
SEPAR      T  +
CH8        C  +

R15        R    Used by NSPCQ
```

Elements used by the subroutines below

Subroutine Usage

WSINIT; MBDSUB for message numbers; PRTERR to print messages; CVTNIS and UPDITM to do disk updates; CRLFPRINT if a format error is found in a "DD" or "DU" history string element; PCLOSEALL if LPBIT=1; if RMODE=0: ISINIT, RESETTERM, RELSP (if USER=2), RELCHN (if OVRFLCTR is non-zero); UNLOCK.GLOCK, GUNLOCK.LINE, and TILD by NSPCQ

Maximum of seven additional levels of subroutine linkage required if RELCHN must print an error message; maximum of six levels required for PRTERR; four levels required for UPDITM; three levels required for ISINIT; two levels always needed for WSINIT

Exits

To the entry point specified in RMODE if non-zero; to LOGOFF if USER=3 (set, for instance, by the DEBUG "OFF" command); to MD0 if USER=2 (set by the LOGOFF processor); otherwise to MD1

Error Conditions

If a format error is found in a "DD" or "DU" history string element, the message

DISK-UPD STRING ERR

is displayed, and processing continues with the next element

3.3.47 WRTLIN WRITOB WT2

These are the star-2d routines for outputting data to the terminal or line printer. Entry WRTLIN deletes trailing blanks from the data and then enters WT2. WT2 adds a trailing carriage return and line feed, increments LINCTR, and enters WRITOB, which outputs the data.

The data to be output is pointed to by OBBEG, and continues through the address pointed to by OB. Output is routed to the terminal if bit LPBIT is off, otherwise it is stored in the printer spooling area. Pagination and page-heading routines are invoked automatically if bit PAGINATE is set. If it is set, then when the number of lines output in the current page (in LINCTR) exceeds the page size (in PAGSIZE), the following actions take place: 1) The number of lines specified in PAGSKIP are skipped, 2) The page number in PAGNUM is incremented, and 3) A new heading is printed (see PRNTHDR documentation). A value of zero in PAGSIZE suppresses pagination, however, regardless of the setting of PAGINATE.

OBBEG	S	Points one byte prior to the output data buffer
OB	R	Points to the last character in the buffer; the buffer must extend at least one character beyond this location
LPBIT	B	If set, output is routed to the spooler (Note: routine SETLPTR should be used to set this bit so printer characteristics are set up correctly)
LISTFLAG	B	If set, all output to the terminal is suppressed
NOBLNK	B	If set, blanking of the output buffer is suppressed
LFDLY	T	Lower byte contains the number of "fill" characters to be output after a CR/LF
PAGINATE	B	If set, pagination and page-headings are invoked
PFILE	T	Contains the print file number for PPUT; meaningful only if LPBIT is set

The following specifications are meaningful only if PAGINATE is set:

PAGHEAD	S	Points one byte before the beginning of the page-heading message; if the frame field of this register is zero, no heading is printed
PAGHEAD	S	Points to the location of the page-heading message
PAGSIZE	T	Contains the number of printable lines per page
PAGSKIP	T	Contains the number of lines to be skipped at the bottom of each page
PAGNUM	T	Contains the current page number
PAGFRMT	B	If set, the process pauses at the end of each page of output until some terminal input (even just a carriage return) is entered
LFDLY	T	If the upper byte is greater than one, and output is to the terminal, a form-feed (X'0C') is output at the top each page, and the number in the upper byte is used as the number of "fill" characters output after the form-feed

Output Interface

```
    OB          R     =OBBEG
```

The following specifications are meaningful only if PAGINATE is set:

```
    LINCTR      T  + Reset appropriately
    PAGNUM      T  +

    T7          T    Contains the original value of PAGNUM
```

Element Usage

```
    R14         R  +
    R15         R  + Scratch
    SYSR1       S  +

    R8          R  +
    RECORD      T  + Used by PPUT (when LPBIT is set)
    OVRFLW      T  +

    SYSR2       S    Used if PAGINATE is set  and the  header
                     message contains a VM

    T4          T  +
    T5          T  + Used if  PAGINATE is  set and the header
    D2          D  + message contains a SVM
    D3          D  +
```

All elements used by ATTOVF (called by PPUT if more disk space needed)

SUBROUTINE USAGE

FFDLY, PPUT (if LPBIT set), WT2 (if PAGINATE set and the header message contains a VM), TIMDATE (if PAGINATE set and the header message contains a SVM), DATE (if PAGINATE set and the header message contains two SVMs in succession)

Four additional levels of subroutine linkage required if LPBIT is set; three levels required for TIMDATE; one level always required for LFDLY

3.3.48 WSINIT WSINIT

This routine initializes the following process work space pointer triads: BMS, BMSBEG, BMSEND; CS, CSBEG, CSEND; AF, AFBEG, AFEND; TS, TSBEG, TSEND; IB, IBBEG, IBEND; OB, OBBEG, OBEND; also PBUFBEG and PBUFEND. In each case, the "beginning" storage register (and associated address register, if present) is set pointing to the first byte of the work space, and the "ending" storage register is set pointing to the last data byte. All work spaces except the last (PROC) are contained in one frame; PBUFBEG and PBUFEND define a 4-frame linked work space.

WORK SPACE	SIZE (BYTES)
BMSBEG-BMSEND	50

AFBEG-AFEND	50
CSBEG-CSEND	100
IBBEG-IBEND	Contents of IBSIZE; max. 140
OBBEG-OBEND	Contents of OBSIZE; max. 140
TSBEG-TSEND	511
PBUFBEG-PBUFEND	20000 (4 linked frames)

Input Interface

IBSIZE	T	Size of IB buffer
OBSIZE	T	Size of OB buffer

Output Interface

Registers are set up as described above. The first byte of each work space, except the OB, is set to x'00'. The OB work space is filled with blanks (x'20'). IBSIZE and OBSIZE are set to 140 if initially greater.

Element Usage

R14	R
R15	R

Subroutine Usage

TSININIT (local), and one internal subroutine

Two additional levels of subroutine linkage required

3.3.49 WTBMS

This routine converts base, modulo, and separation file parameters to an ASCII string.

Input Interface

```
BASE      D  +
MODULO    T  + Contain values to be converted
SEPAR     T  +

TS        R    Points one before the output area
```

Output Interface

```
TS        R  + Point to an AM at the  end of the output
R15       R  + string; the  form of the string is  BASE
               VM MODULO VM SEPAR AM (no  spaces around
               delimiters)
```

Element Usage

```
    D0        D  +
    D1        D  +
    D2        D  +
    T4        T  + Used by MBDSUB
    T5        T  +
    R14       R  +
    R15       R  +
```

Subroutine Usage

MBDSUB; one internal subroutine

Two additional levels of subroutine linkage required

3.3.50 XISOS

XISOS simply exchanges the contents of the IS/ISBEG/ISEND and OS/OSBEG/OSEND register triads.

Chapter 4

SYSTEM DEBUGGER

THE PICK SYSTEM

USER'S ASSEMBLY MANUAL

4.1 OPERATION COMMANDS

NOTE: The form <data specification> is used to indicate a pattern discussed in the section on data specification.

4.1.1 A -- address of element

FORMAT:

A

will display the current instruction location of the virtual code in the form

I ff.dd

where ff is the frame number in decimal and dd is the displacement in hex.

A<data specification>

will display the address of the data specified in the form

f.dd

immediately following the command. The leading format specification part of the data specification is meaningless and will generate the response

ILLGL SYM

immediately after the command.

4.1.2 B -- break

FORMAT:

Bff.dd

will cause a break-point to be set at ff.dd. The command

Bff or Bff.0

will cause every instruction in the frame ff to be a break-point.

The command line for B may contain one or two numeric fields only. They may be in hex or decimal. A + will be emitted on successful completion of the instruction, or the message "TBL FULL" will be emitted.

4.1.3 C -- character display

FORMAT:

C<data specification>

will cause the display to be in character. Any window is allowable. The command is invalid with the A and L commands. The command is part of the data specification section.

4.1.4 D -- display current commands

FORMAT:

D

will cause the break-points, traces and data break-point specifications currently in effect to be displayed.

4.1.5 DB -- toggle debugger availablity

FORMAT:

DB

will toggle the debugger availablity flag. It must be executed from SYSPROG. This is a system wide security feature that can be used to disable most degugger commands.

4.1.6 E -- single-step control

FORMAT: En

where n (1-250), will cause a break and entry to the debug command processor on every nth instruction in the virtual code.

FORMAT:

E

will turn off the single-step function.

4.1.7 END -- back to TCL

FORMAT:

END or end

will cause the process to cleanup and return to TCL.

4.1.8 G -- the go command

FORMAT:

G

will cause the process to continue execution at its current address, if entry to the debugger was not caused by a system abort.

FORMAT:

Gff.dd

will cause the process to commence execution at address dd in frame ff, where dd and ff are in either hex or decimal. No other variations in the syntax are allowed. If the debugger considers the address specified invalid, either because a G has been issued after an error occured, or because of an error in the syntax of the statement, the message,

ADDR

will occur.

4.1.9 H -- toggle echo bit

FORMAT:

H

will toggle the echo bit of the virtual process.

4.1.10 I -- integer display

FORMAT:

I<data specification>

will cause the format of the display to be in integer. This form will be generated by any reference to a symbol of types H, T, D, or F. Any window specification greater than 6 bytes will default to 1 byte. The command is invalid with the A and L commands. This command is part of the data specification section.

FORMAT:

I

will cause further output to be in integer form.

4.1.11 K -- kill break-points

FORMAT:

K

will cause all break-points set by a B command to be terminated. It will emit a -.

FORMAT:

Kff.dd

will kill the break-point ff.dd and emit a hyphen, if it is in the table; or it will emit the message "NOT IN TBL" if the break-point is not in the table.

FORMAT:

Kff or Kff.0

is used in the case that a break was set for all instructions in frame ff. No other variations on the syntax are allowed.

4.1.12 L -- frame links

FORMAT:

L<data specification>

will emit the link fields of the frame implied by the data specification. Format specifications C, I, or X in the data specification are meaningless and will cause an error message.

There is no device for modification of the link fields other than the traditional display-and-modifiy.

4.1.13 M -- modal trace

FORMAT:

M

will toggle the modal trace condition.

4.1.14 N -- number of breaks

FORMAT:

Nn

where n is a tally, will cause the debugger to print the instuction address and other characteristics of n breaks of any kind before it enters the debug command state. If a real error is encountered, the debug command state will be entered immediately.

FORMAT:

N

cancels this such that all breaks will enter the debug command state.

4.1.15 OFF -- back to logon

FORMAT:

OFF

will clean up and log the process off.

4.1.16 P -- toggle LISTFLG

FORMAT:

P

will toggle the bit that controls whether output is output or whether it is tossed into the bit bucket.

4.1.17 R -- register

FORMAT:

Rn

where n (0-15), if it is encounterd in the primary parse, specifies indirect addressing off Rn. It is part of the data specification section.

4.1.18 T -- Trace

FORMAT:

T<data specification>

caused the data element specified to be emitted, along with its address on each break, whether the command state is entered or not. T must be the first character in the command string. A + will be emitted if the command is successful, or the message "TBL FULL" will be emitted.

4.1.19 U -- Untrace

FORMAT:

U

will cause all traces set by a T command to be canceled. It will emit a hyphen.

FORMAT:

U<data specification>

will cause the trace of the specified element to be canceled if it is in the table, and a hyphen will be emitted. If it is not in the table, then the message "NOT IN TBL" will be emitted.

4.1.20 X -- heXidecimal format

FORMAT:

X<data specification>

will cause the data to be displayed in hex. Any window is allowable. The command is invalid with the A and L commands.

4.1.21 Y -- data breaks

FORMAT:

Y<data specification>

will cause the process to break each time the data specified changes. Y must be the first letter in the command. This makes things run very slowly. Note that the current value of the data is kept with the address data, so that the table element size will change with varying sizes of data. Note that the current data is stored in aligned words. Successful completion will terminate with a +; or the message "TBL FULL" will be emitted.

4.1.22 Z -- data unbreak

FORMAT:

Z

will cancel all data-data break commands. A hyphen will be emitted.

FORMAT:

Z<data specification>

will cancel the data break specified. It will emit a hyphen or the message "NOT IN TBL".

4.2 OPERATION COMMANDS : ARITHMETIC UTILITIES

4.2.1 ARITHMETIC CALCULATING FEATURES

FORMATS:

```
ADDD n n

SUBD n n

MULD n n

DIVD n n

ADDX n n

SUBX n n

MULX n n

DIVX n n

XTD n

XTD n n

DTX n

DTX n n
```

do the same things as the related verbs, where XTD <=> RTD and DTX <=> DTR. The numeric arguments, n, are strings without punctuation.

4.3 DATA SPECIFICATION

Data may be referenced directly or indirectly. It may be referenced numerically or symbolically. Window or offset may be specified. Display type, C, I, X, or B may be specified.

4.3.1 DIRECT REFERENCE

FORMAT:

```
ff.dd
```

will reference the data field at dd in frame ff.

FORMAT:

```
dd
```

will reference the data field at dd in the PCB. The frame will be taken to be unlinked.

FORMAT:

```
/ff.dd
```

will take ff to be a linked frame.

4.3.2 INDIRECT REFERENCE

Indirect reference includes all cases wherein a live register is specified, including all symbolic references, or where an *SR form is specified.

4.3.2.1 IMPLICIT indirect reference.

FORMAT:

```
Rn
```

where n (0-15) will reference the data to which Rn points.

FORMAT:

```
/symbol-name
```

where symbol-name is in the PSYM or TSYM, and the PSYM and TSYM are "set", will generate the regsiter number, displacement, format type and window of the symbol. It will be referenced through the implicitly-specified register and displacement.

4.3.2.2 EXPLICIT indirect reference.

FORMAT:

```
*symbol-name
```

will reference the data which the register Rn, if the symbol name is Rn, or the storage register at symbol-name, points.

FORMAT:

```
Rn.dd
```

will apply the displacement, dd, to the location pointed to by Rn in order to obtain a storage register, with which to address the desired data.

FORMAT:

```
*ff.dd        or        *dd
```

will take the location specified to be a storage register, and behave as above. The displacement, dd, will be applied to the frame address in order to find the address of the storage register.

FORMAT:

```
**symbol-name    or    **ff.dd    or    *dd
```

will do the same in the second order. They reference the storage at which the storage register at which the referenced storage register points, with the one condition: That if the first byte of the medial storage register is X'82', then the element is taken to be a BASIC indirect string element and the storage register is taken from two bytes beyond this location. If any of the data fields are invalid as storage registers, then the message "ERR!" will be emitted.

4.4 FORMAT SPECIFICATION

If any of the above forms are preceeded by the character C, I, or X, then that will control the format of the display.

C - CHARACTER display format
I - INTEGER display format
X - HEXADECIMAL display format

4.5 WINDOW SPECIFICATION

If the above location specifications are succeded by a semi-colon, then a window is to be set by the form

;n

where n is a tally for display or a half-tally for the Trace and Y-trace.

4.6 OFFSET SPECIFICATION

The offset specification occurs in conjunction with the window. It has an explicit form and an data-field form.

4.6.1 Explicit offsets.

FORMAT:

```
;o,w
```

where o is a positive or negative tally, and w is a positive number, as above, then o will be an offset from the location specified in the data reference section of data specification. W will be the window used. This form works for traces, except in the case that the location is an indirect reference from a storage register whose location is specified by the form ff.dd.

4.6.2 Implicit offsets.

FORMAT:

```
;Co          or          ;Co,w
```

where o and w are as above, and C e[B,H,C,T,D,F,S,R], will cause the offset to be taken as the number of fields. The field width is 1 bit in the case of B, 1 byte in the case of H and C, 2 in the case of T, 4 in the case of D, 6 in the case of F and S, and 8 in the case of R. O may be positive or negative. If the window is not inluded, then the implicit window deriving from the field type is used, else the specified window is used.

There are further side-effects to this form. The case of

```
;C
```

where C is as above, will take an offset of zero, the implied window and the display type. Note that symbolic reference to data fields has the same effect.

The display-type may be superceeded by a leading format specification of the set C, I, or X.

In the specific case of bits, the form

```
;Bo,w
```

will cause the display to be in bits, starting at bit o, the offset from the addressing base, for a width of w bits. Bits and bit fields may be traced with either trace. There is a further asymmetry here. The displacement specified for a symbollically-addressed bit is in bits. Therefore, the form ff.dd will treat dd as a bit-count in the direct-reference form.

4.7 DISPLAY MODIFIERS

In general, the display modifiers which follow the semi-colon may exibit some excentric behavior because of various logical and functional colisions.

4.8 DISPLAY FORM

The character @ is used to indicate null. The general forms work for the display form, and, mostly, with the trace forms.

```
T  U  Y  Z  @       Trace commands

X  C  I  @          Format specifiers

/  *  **            symbolic, indirect references

dd  .dd             PCB direct reference  | E, N, ME commands

ff.dd   ff,dd       direct reference       | D, G, L, A
.ff.dd  .ff,dd      (frame in hex)         | commands

ff  .ff             D command only
symbol-name         with / or * or **      | L, A commands
                    window, offset and type specifiers.
                    window must be positive, offset may be negative.
                    the format specifier at the beginning of the
                    string will superceed the type specifier.

;n  ;.n             window specification: n bytes
;o,n  ;o.n          offset, o bytes, window, n bytes, decimal or hex
;.o,n  ;.o.n
;-o,n  ;-o.n
;B                  bit display, offset 0, window 1 bit
;Bo                 ibid, offset o
;Bo,n               offset o, window n, in bits
;B,n                offset 0, window n, in bits
;C                  character type, window 1, offset 0
;H                  integer type, window 1, offset 0
;Co,n               window n, offset o bytes, et cetera.
;T                  integer type, window 2, offset 0
;To                 window 2, offset o tallys = 2*o bytes.
;To,n               window n, offset o tallys.
;T,n                window n, offset 0
;D                  integer, window 4
;Do                 window 4, offset o dtlys = 4*o bytes
;Do,n               window 4, offset o dtlys = 4*o bytes
;S                  type X, length 6
;F                  integer type, length 6
;So                 window 6, offset o ftlys = 6*o bytes
;R                  hex type, length 8
;Ro                 window 8, offset o = 8*o bytes
```

FORMAT: of the suffix is the same in all cases. A number of permutations are left out due to redundancy.

4.9 DISPLAY PROMPTS

The value of data fields are changed after they have been displayed using the devices in the previous section. This section considers the actions avaliable at the '=' prompt given by the display prosessor.

4.9.1 <CR> -- back to the command processor

FORMAT:

<CR>

carriage-return will return to the command processor.

4.9.2 <LF> -- the next window

FORMAT:

<LF>

line-feed, will display the next window of data, on the same line.

4.9.3 <control-N> -- the address and the next window.

FORMAT:

<control-N>

will display the address of the next window and the next window on the next line.

4.9.4 <control-P> -- the address and the previous window.

FORMAT:

<control-P>

will display the address of the previous window and the previous window on the next line.

4.9.5 '<string> -- character data

FORMAT:

'<string>

will cause the characters in the <string> to be placed in the data area starting at the beginning of the displayed window for the length of <string>, which will not exceed 40 bytes. The string must terminate with CR, LF, control-N, or control-P. The string terminators noted hereinafter have the same effect as the same character used as the only response to the display prompt.

4.9.6 INTEGER INSERTION

FORMAT:

<decimal number>

will cause the value of <decimal number> to be placed in the window displayed, filling from the right, if the window is 1, 2, 4, or 6 bytes in length, and does not cross a frame boundary, else an error message will occur. The string must terminate with CR, LF, control-N, or control-P.

4.9.7 HEXIDECIMAL STRING INSERTION

FORMAT:

.<hex string>

will cause the value of the data area beginning at the left of the window displayed to be replaced by the hex string. The string must contain an even number of characters, and must contian only hex characters. The string will not have more than 38 hex characters in it. The string must terminate with CR, LF, control-N or control-P.

4.9.8 BIT STRING INSERTION

If the display type is bit,

FORMAT:

<binary string>

where <binary string> is a sequence of 1's and 0' less than 40 characters long, will cause the bits starting from the first bit in the displayed window to be replaced by the bits in the string. The string must terminate with CR, LF, control-N or control-P.

4.9.9 CLEARING WINDOWS

FORMAT:

0

will have the effect of clearing the window to null, if the type is not bit. It must be followed by CR, LF, control-N or control-P.

4.9.10 ADDRESS DISPLAY

FORMAT:

A

will display the address of the last window, and redisplay the last window.

4.9.11 DISPLAY TYPE, WINDOW, AND OFFSET MODIFICATION

FORMAT:

C or Cn or Co,n

will change the display type, window and offset, if specified, and redisplay either the original field with the new type or window specification, or the resultant field if the offset is modified. The string must be followed by a CR or LF, both of which leave one in the display mode, and on the next line.

The legal display types are C, character, I, integer, X, hexidecimal, and B, bit. Transfers to and from bit have the effect of byte-alignment in either direction, and retaining the numerical size of the window, which is then interpreted either in bits or bytes.

The window specification sets the window at the new size.

The offset specification is in bytes or bits, depending on the type specified, may be positive or negative, in hex or decimal, and simply redirects the data specification pointer to a new location.

The intent of this is to mainpulate type and window in display mode quickly and simply.

Chapter 5

THE PC SYSTEM ASSEMBLER

CAUTION *** CAUTION *** CAUTION

USE EXTREME CAUTION IN CREATING AND LOADING ASSEMBLY CODE !!!

Improper user written assembly code can cause severe problems on your system including loss of data, group format errors, and system crashes. PICK SYSTEMS cancels ALL warranties on any computer system that is running user written assembly code.

CAUTION *** CAUTION *** CAUTION

PROPRIETARY INFORMATION

5.1 LOADING THE ASSEMBLY ACCOUNT FLOPPY

The PICK PC SYSTEM ASSEMBLER floppy contains an account which has all the files necessary to :

- CREATE
- ASSEMBLE

and

- LOAD

PICK Assembler Code for the PC SYSTEM.

Follow these instructions to install the assembler account :

1. Logto 'SYSPROG'.
2. The Assembler account requires a minimum 700 disk frames. Ensure 700 frames of available disk space, by keying in at TCL 'POVF' <CR>.
3. Mount floppy in diskette drive "A".
4. Type 'T-ATT' <CR> and note that your terminal did indeed attach the tape drive.
5. Type 'T-REW' <CR>.
6. Type 'ACCOUNT-RESTORE ASSEMBLER' <CR>.
7. The System prompts with Account Name on Tape? Type 'ASSEMBLER' <CR>.
8. Assembler files display as loaded.
9. System returns to TCL.

5.2 ASSEMBLING CODE ON PC SYSTEMS

The PICK Assembler for PC SYSTEMS uses the same PICK source code as all other PICK systems, but the physical procedures are a little different (simplified).

Follow these instructions to assemble and load your assembly code :

1. Put your assembly source code into the file "APSM" .

2. Use the PROC "AS" to either -

   a. 'AS item.name' assemble one program

   or

   b. 'GET-LIST list.name' and then assemble a list of programs. AS

3. Assembled object code is stored in the file "ASM" .

4. Since PC SYSTEMS are software machines you must ensure that your code had no errors, and that the total frame size is less than 2K.

   Both the APSM and the ASM file should be checked by typing:

   ```
   MLIST APSM item.name (E
   MLIST ASM item.name (E
   ```

   Also check that the total resultant object frame size is less than 2K. To determine the size type:

   ```
   LIST ASM 'item.name' SIZE
   ```

5. Load the assembled code by using the standard command - 'MLOAD ASM item.name'

Printed in the U.S.A.

171-010-002 A1