ASPLM010-1
3.0/B
March, 1974

ASSIST SYSTEM
PROGRAM LOGIC MANUAL

Program&Documentation: John R. Mashey
Project Supervision : Graham Campbell
PSU Computer Science Department

This manual contains various information describing the internal structure and techniques used in the ASSIST system. It includes details required for maintenance and extension of the system, describes the purposes and methods of all the modules in the system, and in general, is required reading for any person performing modification of the system Depending on the intentions of the reader, certain sections of this manual should be consulted. All readers should begin by reading the following section GUIDES TO EFFECTIVE USE and also the section INTRODUCTION.

GUIDES TO EFFECTIVE USE

This section essentially describes how to use this manual properly, and should be consulted before reading any other section or examining the ASSIST source program.

A. WHO is to use this manual?

This manual is definitely required reading for any person desiring modifications to the ASSIST source program for any reason. This category includes maintenance of ASSIST, extensions and additional features to be added to ASSIST, and modifications for handling programs for a given installation in a different manner than provided for already. It also includes improvements in the code used internally, and any modifications needed to run ASSIST in an operating system environment not currently allowed.

Some sections of this manual may be useful to the system administrator, as they can aid him in determining the system resource requirements needed to make effective use of ASSIST, and in deciding which of the available options should be included in a given generation of ASSIST for his installation.

The more general sections of this manual should be useful to the instructor of computer science who wishes to provide his students with an example of a system for discussion, or with possible programming projects for them to do.

B. WHAT does this manual contain?

Each major part of this manual describes one major division of the ASSIST program. Section A. of each part gives a general summary of the input, output, and overall control logic for the modules of that part. Entry point hierarchy tables included here show the structure and calling relationships among the various program components. Section B. of each part then gives more detailed information about each module in the given part of system. In general, the information is presented in each section in order from most general to most specific, to facilitate easy location of any given module or any code performing any particular function.

The Appendices present detailed information which either concerns many of the program modules in ASSIST, or whose nature is such to special consideration or treatment independent of the rest of the manual. The information in the Appendices should be especially useful to any person performing modifcation to the ASSIST source program.

C. WHEN should this manual be used?

This manual may be consulted before generation of an ASSIST system is performed, in order to insure that the desired options are present. This manual definitely should be examined before reading the source program of ASSIST. While the source program contains extensive internal documentation, overall structure is detailed in this manual, and provides the easiest method of locating anything in the source program.

D. WHERE does this manual fit into the literature?

The documentation for the ASSIST system consists of the following:

ASSIST INTRODUCTORY ASSEMBLER USER'S MANUAL
This manual gives all necessary information required to run jobs under ASSIST, describes the subset of S/360 Assembler Language accepted by ASSIST, and describes the output produced by ASSIST. THE USER'S MANUAL SHOULD BE EXAMINED BEFORE ANY OTHER DOCUMENTATION.

ASSIST ASSEMBLER REPLACEMENT USER'S GUIDE
This manual gives the information required to run a replacement program for a module of ASSIST. It shows deck setup, debugging aids, register conventions, and error messages. It should be examined by any instructor making use of ASSIST for replacement assignments.

ASSIST SYSTEM DISTRIBUTION WRITEUP
This manual supplies the procedures required for generating an ASSIST system for given options and requirements. Concerning options available for generation, it briefly notes the most likely combinations which may be used. For unusual circumstances, the reader is referrred to the ASSIST PROGRAM LOGIC MANUAL for further details on options.

E. HOW should this manual be used by various groups of people?

The reader intending to examine the ASSIST source program should consult the following:

INTRODUCTION
APPENDIX I. GENERAL CONVENTIONS AND INFORMATION
PART I - MAIN CONTROL AND SERVICE PROGRAMS - Section A.
Depending on area of interest, any of PARTs I-IV and any of the Appendices.
ASSIST source listing - should include at least an assembly listing, and possibly a utility listing of the complete source

The reader desiring more detailed information about generation options than supplied by the ASSIST SYSTEM GENERATION MANUAL should examine the following sections:

INTRODUCTION
APPENDIX II. SET VARIABLES AND CONDITIONAL ASSEMBLY
APPENDIX VII. SYSTEM RESOURCE REQUIREMENTS, JOB CONTROL LANGUAGE
APPENDIX VIII. TIME AND RECORDS PROCESSING

The reader desiring only a general idea of the internal workings of ASSIST should read these sections:

INTRODUCTION
Section A. of PARTs I. II, III, and possibly IV. (These sections describe the overall control logic of each major division of the system.)

INTRODUCTION

ASSIST (Assembler System for Student Instruction and Systems Teaching) is a small, high-speed, low-overhead assembler/interpreter system especially suitable for student use. It consists of five main components, which are as follows:

I. Main control and service programs
This includes a job monitor, input/output routines, scanning and conversion modules, and a debugging/dumping program. It may include object deck punch and loader routines.

II. Assembler for a subset of S/360 Assembler Langugage
This component of the system assembles user source program and creates resulting object program in memory.

III. Interpreter for S/360 object code
This module interpretively executes user object programs, permitting complete control of the user program, in addition to execution of special debugging instructions.

IV. Replace monitor

V. Macro Processor
Supports inline macros, open code conditional assembly.

TABLE OF CONTENTS

NOTE certain portions of this manual are not currently available (such as 110, 210, and 500). In general, these deal with program logic inside individual modules. The reader should consult the comments in Appendix V, or the source program itself, which contains fairly heavy documentation (28 per cent comment cards).

PART I - MAIN CONTROL AND SERVICE PROGRAMS

This part of the ASSIST Program Logic Manual describes both the overall flow of control and structure of the ASSIST system, and details of various service programs which may be used during execution. The service subprograms noted include both those which are normally only called from the main program, and those which are available for use by any other module in the ASSIST system.

The following sections detail the input to and output from the entire ASSIST system, the overall logic of the main program ASSIST, and a hierarchy chart showing calling relationships between entry points in the system. Following these sections are given more detailed descriptions of each module in the control and service program group.

A.OVERALL CONTROL LOGIC

1. INPUT

a. USER SOURCE PROGRAM/OBJECT PROGRAM

The user source program, written in assembler language, is read from a card reader, using a DDNAME of SYSIN. This is the only required input to ASSIST. Some versions of ASSIST may contain a module used to load object decks (AOBJIN), in which case an object deck takes the place of a source deck.

b. USER DATA (OPTIONAL)

(Optional) data is read during user program execution, from a card reader, using a DDNAME of FT05F001, if possible. For jobs using the BATCH option, or if two readers are not available (as in an OS-PCP system), any data cards provided must follow the user source program, prefixed by a $ENTRY card. Any number of user jobs may be batched.

c. USER PARM FIELD (OPTIONAL)

ASSIST expects the operating system to provide to it the contents of the PARM field from the user EXEC card, following standard OS/360 conventions for accessing PARM fields. The user may request various options through the use of this field. The reader is directed to the following writeup for a listing of the available options:

ASSIST INTRODUCTORY USER'S MANUAL - PART III.

If ASSIST is generated to provide the replacement feature, i.e., &$REPL>0, the following should be consulted for additional options:

ASSIST ASSEMBLER REPLACEMENT USER'S GUIDE

2. OUTPUT

a. PRINTED OUTPUT

ASSIST prints all output on a line printer, using a DDNAME of FT06F001. The printed output may require up to 133 characters, including carriage control, although most output requires no more than 121 characters. The following output may be printed:

1) USER PROGRAM ASSEMBLY LISTING
2) USER PROGRAM PRINTED OUTPUT
3) USER PROGRAM COMPLETION DUMP
4) VARIOUS HEADINGS AND STATISTICAL INFORMATION

b. PUNCHED OUTPUT

The user program may request that cards be punched during its execution. If possible, cards are actually punched, using the DDNAME FT07F001. If a DCB cannot be opened to this DDNAME, or if the NOPUNCH PARM option is used, any card images produced will not be punched, but will be printed instead. It is also possible to create an ASSIST which deletes all punching code, thus saving space. A version of ASSIST can be generated possessing the ability to punch object decks also.

The reader should refer to the following writeup for a detailed description of the output produced by ASSIST, including the effects of various PARM options:

ASSIST INTRODUCTORY USER'S MANUAL - PART III, PART IV.

The reader should also refer to APPENDIX VII of this manual, which details the system resources and job control language required to use ASSIST. Briefly, ASSIST can function effectively with one card reader, one line printer, less than 30K bytes of memory for program, and a variable amount of storage for workareas and user program, approximately 50-90 bytes per source statement. No intermediate storage is required. if all possible options are allowed (as of version 2.1/A), the assist program requires approximately 62K bytes. This is larger thean almost any version which would actually be run, since few installations would use all options.

3. OVERALL CONTROL LOGIC SUMMARY

The following summarizes the overall flow of control in the entire ASSIST system. Since overall control resides entirely in the control section ASSIST, this section is essentially a brief summary of the later section B.1 of this part. This section can be used to get a quick overview, while the later section describes the ASSIST control section in more detail.

a. ENTIRE RUN INITIALIZATION

After receiving control from the operating system, ASSIST first sets initial values for any flags which may be required to be set at this time.

Certain operations occur only 1 time for an entire ASSIST run, and are thus described here, although they occur after the first case of the processing described in b. below. These include calling INSUB ASMSINIT, which obtains the largest single block of free storage available, then returns some to the operating system for buffers, modules, etc. The value returned can be controlled by the PARM field FREE= option.

ASSIST then calls entry XXXXINIT of csect XXXXIOCO. This entry OPENs the DCB's for the source reader (SYSIN) and line printer (FT06F001), and sets flags in AJOBCON denoting the success or failure of OPENing the two DCB's. If either fails, ASSIST writes a message to that effect on the system log, then terminates with a condition code of 16.

b. ASSEMBLY/EXECUTION/DUMP CYCLE (1 $JOB)

ASSIST now makes 3 or 4 calls to routine APARMS, which is used to scan optional parameters (see PART III of USER'S MANUAL). The first call supplies default values and absolute limits on numerical options, and the second call processes the invoking PARM field, if any exists. If this is the first time through this process, the actions described above in a. are performed to initialize memory and i/o modules. (This is done in the order given in order to allow optional parameters to influence this initialization).

At this point, ASSIST determines whether a BATCH run is in progress. If so, it reads cards until a $JOB card is found (INSUB ASFLUSH), and calls APARMS with the parameters on the first such card encountered, thus allowing user's in BATCH mode to supply options.

APARMS is called a third time (NOBATCH) or fourth time (BATCH) to supply default values for any numerical options.

As described in APPENDIX VIII, time, records, and pages limits are calculated, the ASSIST header line printed on a new page, and either the invoking PARM field (NOBATCH) or $JOB card (BATCH) added.

c. ASSEMBLY OF USER PROGRAM (OR LOAD OF OBJECT DECK)

ASSIST then sets up the assembler control table (VWXTABL csect, AVWXTABL dsect) for use by the assembler component of the system. This includes setting limits on the memory available to the assembler, placing the address of the table control section into the appropriate register (R12), and initializing any flag values which may have been provided by the user PARM field. In addition, an STIMER macro may be set to interrupt if the user overruns his time limit. The assembler main program (MPCON0) is then called to assemble the user program, and load the resulting object code into the beginning of the single large workarea used by the system.

If the user specified the OBJIN option (thus showing that an object deck is supplied in place of the source deck), ASSIST skips the actions described above and calls AOBJIN to load the object deck (assuming that ASSIST has been generated to allow this option.)

After the user program has been assembled, ASSIST tests to see if the user did not desire execution of his program, or if the program has become unexecutable because it contained too many errors. In this case, execution and completion dumping are skipped, and section f. is done.

d. USER PROGRAM EXECUTION

If a BATCH run is in progress, ASSIST requires that a $ENTRY card be supplied after the user program to initiate execution, so it reads cards until one is found. If a $JOB card is found first, it returns to section b and begins processing of next run.

Using various values from the assembler control table AVWXTABL, ASSIST creates an execution control block (ECONTROL), which contains all necessary information for describing the user program and how it should be treated. This table is actually located in a part of the assembler control table, to save space. After initializing such values as the pseudo registers, Program Status Word, etc, in ECONTROL, ASSIST calls entry XXXXSNIN of csect XXXXSNAP, which performs any needed initialization for this csect, which is used to perform execution-time register and storage dumping. ASSIST tests to make sure that neither the time limit nor the record limit have already been exceeded during assembly. If so, user program execution is skipped, and section f. is done.

ASSIST flags the user program in execution, and calls EXECUT, which is an interpreter for S/360 object code. The user program is then interpretively executed. The interpreter may call entries XXXXREAD, XXXXPNCH, and XXXXPRNT of csect XXXXIOCO to read cards, punch cards, and print lines for the executing user program. EXECUT may also call csects XXXXDECI and XXXXDECO to perform decimal conversions for the user, and may call XXXXSNAP csect to supply execution-time dumps of registers and storage.

Interpretation of the user object program continues until some error occurs, or until the program branches to the location originally given to it as a return address. Errors include program interrupts, overrrun of instruction count limit, overrun of record limit, and possibly overrun of time limit (See section H. for this). After setting appropriate condition flags, EXECUT returns control to ASSIST.

e. USER PROGRAM COMPLETION DUMP

If user execution terminated properly, a message is printed to this effect. Otherwise, XXXXSNAP csect is called using a special type of call to the XSNAP macro, which produces a user completion dump instead of normal XSNAP output. Depending on options supplied by the user, XXXXSNAP prints either a full dump (Program Status Word, Completion Completion Code, Instruction Trace, User Registers, and User Storage), or a short dump (all parts of a full dump except User Storage).

f. BATCH RUN TESTING

ASSIST prints a message if the user has exceeded either a time or record limit. If the run is not a BATCH run, ASSIST terminates (see section g.)

ASSIST now returns to section b to search for the next $JOB card. If ASSIST ever encounters either a real end-file or a $STOP card while searching for a $JOB or $ENTRY card, the run is terminated at that point.

g. TERMINATION

ASSIST calls XXXXFINI entry of XXXXIOCO, which closes all DCB's which have been OPENed. The return code is set to 0 to indicate a successful run, and all acquired storage is returned to the operating system. ASSIST then returns control to the calling program.

h. TIMER RUNOUT PROCESSING

If ASSIST is generated with one of the options providing timing for a user program, an STIMER exit routine may be executed. If this occurs, the exit routine tests a flag to determine if the system is currently executing a user program. If so, a flag is set in the execution control block ECONTROL. The interpreter EXECUT tests this flag after every successful branch by a user program, and terminates user program execution if the flag has been set.

If the user program was still being assembled, ASSIST sets the 'STOP' flag in the assembler control table, which terminates assembly when the next statement is encountered during either assembly passes.

ASSIST also sets the 'RECORDS EXCEEDED' flag, which is tested by the printer/punch i/o modules. This is sufficient to terminate the user dump if it is being produced.

See APPENDIX VIII for a complete description of timer/records/pages control, as a number of different effects can be obtained by using various combinations of generation options and PARM values.

4. ENTRY POINT HIERARCHY TABLE

The following lists all entry names which may be called duirng the course of an entire ASSIST run, excluding internal modules of the assembler section of ASSIST, which has its own table (see PART II, section A.4.) The entries are listed in order by level, where the level is the maximum number of modules in a calling chain above the given module, i.e., ASSIST has a level of 0 because it is the main program. The entries are listed first by level, then alphabetically within each level. Under each entry are listed the entries it may call, first by level, then in alphabetical order. Certain entries may only be called by use of certain macro instructions, in which case the macro names are also given. Any entry which calls no others is flagged with an asterisk, for ease of use in tracing calling chains.

LEVEL 0 ENTRIES

```
     ASSIST - ASSIST system main control program
       1  AOBJIN, AODECK, APARMS, EXECUT, MPCON0, XXXXFINI*, XXXXINIT*,
          XXXXSNIN*
          $SORC(XXXXSORC)*, XSNAP(XXXXSNAP)
       3  $PRNT(XXXXPRNT)*
```

LEVEL 1 ENTRIES

```
     AOBJIN - Object Deck Input (only if &$OBJIN=1)
       2  $SORC(XXXXSORC)*, XSNAP(XXXXSNAP)
       3  $PRNT(XXXXPRNT)*

     AODECK - Object Deck Punch (only if &$DECK=1)
       2  $PNCH(XXXPNCH)*
       2  $READ(XXXXREAD)*, XSNAP(XXXXSNAP)
     APARMS - ASSIST PARM field analysis routine
       2  XDECI(XXXXDECI)*

     EXECUT - ASSIST S/360 object code interpreter
       2  XDECI(XXXXDECI)*, XDECO(XXXXDECO)*, XHEXI(XXXXHEXI)*
       2  XHEXO(XXXXHEXO)*, $PNCH(XXXXPNCH)*, XSNAP(XXXXSNAP)
       2  XSNAP(XXXXSNAP)
       3  $PRNT(XXXXPRNT)*

     MPCON0 - ASSIST assembler main control program
          (**Note, the following list includes any entries called by any
          sections of the ASSIST assembler.  See assembler hierarchy
          table in Part II, section A.4. for details).

       1  REFAKE (only if &$REPL>0. Also note that this call only
          occurs during a replace run, and does not appear in the
          actual call sequence, due to adcon modification.)
       2  RESYMB*(&$REPL=2), $SORC(XXXXSORC)*
       3  $PRNT(XXXXPRNT)*
```

REENDA - Post Assembly Replace Monitor (only if &$REPL>0)
  3 $PRNT(XXXXPRNT)*
  7 SYFIND* (entry point inside csect SYMOPS of the assembler)

REFAKE - Replace Monitor Call Interception (only if &$REPL>0)
  2 XDECO(XXXXDECO)*, XSNAP(XXXXSNAP)
  3 $PRNT(XXXXPRNT)*

REINTA - Pre-Assembly Replace Monitor (only if &$REPL>0)

XXXXFINI* - Finish up input/output control - CLOSE DCB's

XXXXINIT* - Initialize input/output control - OPEN DCB's

XXXXSNIN* - Initialize XXXXSNAP module before user execution.

LEVEL 2 ENTRIES

RESYMB* - Replace Monitor Call Allowed Lookup (only if &$REPL=2)

XXXXDECI* - Decimal Input Service Program (XDECI macro)

XXXXDECO* - Decimal Output Service Program (XDECO macro)

XXXXHEXI* - Hexadecimal Input Service Program (XHEXI macro)

XXXXHEXO* - Hexadecimal Output Service Program (XHEXO macro)

XXXXPNCH* - Punch card service program ($PNCH macro)

XXXXREAD* - Read data card service program ($READ macro)

XXXXSNAP - Create debugging output, completion dump (XSNAP macro)
  3 $PRNT(XXXXPRNT)*

XXXXSORC* - Read source card service program ($SORC macro)

LEVEL 3 ENTRIES:

XXXXPRNT* - Print a line service program ($PRNT macro)

PART II - THE ASSEMBLER

The assembler section of ASSIST is a high-speed two-pass assembler which produces an object program directly in memory, ready to be interpretively executed. For the assembler language accepted by this assembler, see the ASSIST INTRODUCTORY USER'S MANUAL. This section of the manual gives first an overview of the internal workings of the assembler, then descriptions of the logic for each separate control section in the assembler.

A. OVERALL CONTROL LOGIC

1. INPUT

a. Address of primary assembler table (AVWXTABL DSECT).

The calling program passes to the main program of the assembler the address of a table which contains all communications areas, address constants, useful constants, and some workareas. The calling program fills in some values before calling the assembler. These values include the following:

Two words are given the values of low and high limits of a single large workarea which may be used by the assembler.

Two bytes are given values of various bit flags which determine exactly what running mode the assembler will use, and what options will be in effect.

A halfword is given the value of the maximum number of errors which can occur and still permit execution.

b. Deck of assembly language source cards.

The assembler source deck is read using the $SORC macro. This deck is terminated by an END card, end-of-file indicator, or ASSIST internal control card ($JOB, $ENTRY, or $STOP card used in BATCH run).

2. OUTPUT

a. Source listing.

The ASSIST source listing resembles the standard assembler listing very closely, but may be omitted if the NOLIST option is specified. All statements which are flagged are always printed, regardless of the status of print control.

The assembler prints various statistics at the end of the listing which note the numbers of errors and warnings, and describe the amount of core storage required to perform the assembly.

b. Object program.

The object program is produced at the beginning of the dynamic workarea, and is ready for execution at the end of the assembly. The object program is not created if NOLOAD is specified in the user PARM field. The assembler also stops producing code if the number of errors exceeds the limit at any time.

c. Values in main assembler table.

The assembler sets various flags and values in the main assembler table, whose address was passed to it originally. These values are then used by the calling program, and include the following:

Two bytes of flag bits (same as those passed in, but with more bits possibly set). One bit notes whether the assembled program should be permitted to be interpreted.

Two words give the real low and high limits of the assembled program in memory.

Two words give the low and high limits of the program as assembled, i.e. the addresses appearing in the user assembly listing.

One word gives the relocation factor applied to the user program addresses to obtain the corresponding real addresses in memory.

One word gives the entry point address in the user program, which is either the first byte of the user program or an address given on a user END card.

A halfword gives the number of statements in the assembly, for possible use in calculating assembly rates.

The assembler does not return a return code in register 15; all information to be returned is placed in the main assembler table.

3. OVERALL CONTROL LOGIC SUMMARY

a. MPCON0 - ASSEMBLER MAIN PROGRAM

The main program initializes some areas of the main assembler table AVWXTABL, sets a SPIE to trap interrupts, initializes the program mask, and then calls the remaining subroutines. After printing the storage usage, it returns control to the calling program.

MPCON0 calls all entry points in the assembler which are required to be called in a fixed order. The entry points then fall into the following groups:

1) PASS I initialization entries.
2) PASS I main control (MOCON1 csect).
3) PASS I ending, PASS II initialization entries.
4) PASS II main control (MTCON2 csect).
5) PASS II ending entries.

b. MOCON1 - PASS I MAIN CONTROL PROGRAM

MOCON1 receives control after all initialization has been done, both for the overall assembly, and for each subroutine requiring it. It then controls the first pass of the assembly, causing source cards to be read and scanned until an END card is found, after which it returns control to the main program MPCON0.

MOCON1 begins the cycle for each source statement by calling the entry INCARD, which reads a source statement, and builds the required record blocks for the statement. The record blocks include a block for the source image and required flags (RSBLOCK), an optional block for continuations and sequence numbers (RSCBLK), an optional block for error code/scan pointer pairs (REBLK), and an optional block for information depending on the type of operation code (RCODBLK). The RSBLOCK for a statement includes bit flags which note the existence of any other blocks for a statement. INCARD builds an RSBLOCK and possibly an RSCBLK if required. If it encounters an end-of-file indication while reading, it creates a dummy END card, since MOCON1 executes until it finds an END card.

MOCON1 now checks the current statement for being a comment card, and scans for a label on the statement. If a legal label is found, it calls SYENT1 to enter the label into the symbol table (but does not define the symbol at this time). The address of a label in the symbol table is saved in the main assembler table, where it can be accessed by other modules.

MOCON1 now scans for an operation code in the statement. If one is found, it calls OPFIND to determine whether the opcode is legal, and if so, which one it is. OPFIND returns the address of an opcode control table entry (OPCODTB), which contains various flag values (3 bytes). Among other things, the first two bits of one of the bytes determines what type of instruction has been found.

MOCON1 now scans for an operand field, setting the scan pointer to the address of the first blank after the opcode if there is no operand.

The type of instruction is determined, and the proper second-level subroutine is called to process the statement (IAMOP1 for machine operations, IBASM1 for assembler statements). Each second-level program performs any location counter alignment required for the statement, performs some scanning of the statement, depending on its type, and determines the total length of the statement to be added to the location counter. Each creates a record code block for variable data (RCODBLK), and returns the address of it to MOCON1.

MOCON1 completes the RCODBLK by placing the beginning location counter value for the statement in the block, and also places this value in the symbol table entry for the statement label, if there was one. The location counter is incremented, and UTPUT1 is ccalled to save all o the record blocks which exist for the statement. If the statement was an END statement, MOCON1 returns control to MPCON0; otherwise it repeats the above process for the next statement.

See part V. for MACRO Processing description in MOCON1.

c. MTCON2 - PASS II MAIN CONTROL PROGRAM

MTCON2 is called after all of the PASS I ending routines and PASS II initialization routines have been called. After brief initialization of its own, it enters a loop, one time for each statement.

MTCON2 first calls UTGET2, which returns either an indication that no statements are left to assembler, or the addresses of all existing record blocks for the next statement. UTGET2 also restores the record error block (REBLK) to its place in AVWXTABL (AVREBLK), if the REBLK already exists.

MTCON2 tests for the existence of a record code block (RCODBLK). If none exists, the statement was either a comment or had an unknown opcode, so MTCON2 just calls OUTPT2 to format and print the statement, since no further processing can be done for it. If an RCDOBLK exists, MTCON2 sets the location counter form the value saved in it, sets the scan pointer to the scan pointer saved in the RSBLOCK, and then calls the appropriate second-level routine to process the statement.

The second-level routine (ICMOP2 for machine instructions, IDASM2 for assembler instructions), performs any required operand processing, then calls UTPUT2 to load any assembled object code, and OUTPT2 to format and print the statement, with any error messages required.

After all statements have been processed, MTCON2 aligns the highest location counter value to a doubleword boundary, then returns control to MPCON0.

4. ENTRY POINT HIERARCHY TABLE

The following table lists all entry points called during the course of an assembly. The entries are listed in order by level, where the level of a module is defined to be the maximum number of modules in a calling chain above the given module, i.e. the main program has a level of 0. The entries are listed first by level, then in alphabetical order within each level. Under each module are listed the entries that it may call, by level, then by alphabetical order. Names preceded by 'M' instead of a level number are names of macros which call intrinsic modules, whose names are also given. For ease of use, any entry which calls no others in the assembler is flagged with an asterisk.

LEVEL 0 ENTRIES

```
MPCON0  - assembler main control program
  1  BRINIT*, ESINT1*, LTINT1*, LTEND1*, MOCON1, MTCON2, OPINIT*,
  1  OUEND*,OUINT*,SYEND2*,SYINT1*,UTEND2,UTINT1,UTEND1
  6  ERRTAG*
  M  $PRNT(XXXXPRNT)
```

LEVEL 1 ENTRIES

```
ESINT1* - pre-pass I external symbol module initialization

LTEND1* - end of pass I for literal table processor

LTINT1* - pre-pass I literal table processor initialization

MOCON1  - Pass I main control program
  2  IAMOP1, IBASM1, INCARD, OPFIND*, UTPUT1*
  4  SYENT1*
  5  ERRLAB
  6  ERRTAG*
  M  $PRNT(XXXXPRNT)

MTCON2  - Pass II main control program
  2  ICMOP2, IDASM2, UTGET2*
  5  OUTPT2*

OPINIT* - initialize opcode table processor

OUEND2* - end of assembly for output formatting processor
  M  $PRNT(XXXXPRNT)

OUINT1* - pre-pass I output processor initialization

SYEND2* - end of assembly for symbol table processor

SYINT1* - pre-pass I symbol table initialization

UTEND2  - end of assembly for utilities module (code production)
  5  UTPUT2*

UTEND1* - terminate pass I, initiate pass II for storage utilities
  3  XXXXDKE1
```

```
UTINT1* - pass I initialize utilities routines
  3  XXXXDKOP
```

LEVEL 2 ENTRIES

```
IAMOP1  - Pass I machine instruction scanning
  3  LTENT1, SCANEQ
  6  ERRTAG*

IBASM1  - Pass II assembler instruction scanning and processing
  3  ESCSEC*, ESENX1, LTDMP1*
  4  CODTL1
  5  CCCON1*, ERRLAB
  6  ERRTAG*, EVALUT
  7  SDBCDX, SYFIND*
  8  SDDTRM*

ICMOP2  - Pass II machine instructions - operand scanning, assembly
  3  BRDISP*, LTGET2*
  5  OUTPT2*, UTPUT2*
  6  ERRTAG*, EVALUT
  7  SDBCDX, SYFIND*
  8  SDDTRM

IDASM2  - Pass II assembler instructions - scanning and assembly
  3  BRDROP*, BRUSIN*, ESENX2, LTDMP2
  4  CNDTL2
  5  CCCON2*, OUTPT2*, UTPUT2*
  6  ERRTAG*, EVALUT

INCARD  - input ofsource cards, construction of record blocks
  6  ERRTAG*
  M  $SORC(XXXXSORC)

OPFIND* - look up opcode in opcode table

UTGET2*  - Pass II retrieval of record block addresses
  3  XXXXDKRD

UTPUT1* - Pass I saving of record blocks in dynamic workarea
  3  XXXXDKWT
```

LEVEL 3 ENTRIES

```
BRDISP* - decode address into base-displacement form

BRDROP* - drop a register from base register availability

BRUSIN* - allow a given register to be used as a base register

ESCSEC* - external symbol manipulation - START, CSECT, DSECT

ESENX1  - ENTRY, EXTRN processing during Pass I
  4  SYENT1*

ESENX2  - ENTRY, EXTRN processing during Pass II
  4  SYENT1*
  6  ERRTAG*

LTDMP1* - calculate literal pool length during Pass I

LTDMP2  - have a literal pool assembled and printed during Pass II
```

4 CNDTL2

```
     LTGET2* - retrieve program address of a given literal, Pass II

     LTENT1  - scan literal during Pass I, enter into literal table
       4  CODTL1

     LTGET2* - retrieve program address of a given literal, Pass II

     SCANEQ  - scan expression until = or blank found
       7  SDBCDX

     XXXXDKE1-terminate write phase, initiate read phase during
              disk utility run

     XXXXDKOP-initialize disk utility during disk utility run

     XXXXDKRD-read a buffer from disk during disk utility run

     XXXXDKWT-write a buffer to disk during disk utility run


LEVEL 4 ENTRIES

     CNDTL2  - assemble code for DC or literal constant, have it printed
       5  CACON2, CBCON2*, CCCON2*, CFHCN2*, CPCON2*,  CVCON2,  CXCON2*,
       5  CZCON2*, OUTPT2*, UTPUT2*
       6  CDECN2, ERRTAG*

     CODTL1  - scan DS, DC, literal  constant,  build  CNCBLOCK  for  it
       5  CACON1, CBCON1*, CCCON1*, CDECN1*, CFHCN1*, CPCON1*,  CVCON1*,
       5  CXCON1*, CZCON1*
       6  ERRTAG*, EVALUT
       8  SDDTRM*

     SYENT1* - enter symbol in symbol table  return @ table entry for it


LEVEL 5 ENTRIES

     CACON1  - Pass I A-type constant processor
       6  SCANCO

     CACON2  - pass II A-type constant processor
       6  EVALUT

     CBCON1* - pass I B-type constant processor

     CBCON2* - pass II B-type constant processor

     CCCON1* - pass I C-type constant processor

     CCCON2* - pass II C-type constant processor

     CDECN1  - pass I D and E-type constant processor
       6  CDECN2

     CFHCN1* - pass I F- and H-type constant processor

     CFHCN2* - pass II F- and H-type constant processor
```

CPCON1* - pass I P-type constant processor

CPCON2* - pass II P-type constant processor

CVCON1* - pass I V-type constant processor

CVCON2 - pass II V-type constant processor
7 SYFIND*

CXCON1* - pass I X-type constant processor

CXCON2* - pass I X-type constant processor

CZCON1* - pass I Z-type constant processor

CZCON2* - pass II Z-type constant processor

ERRLAB - flag a label error, saving scan pointer
6 ERRTAG*

OUTPT2* - format and print a statement, with error messages
M $PRNT(XXXXPRNT)

UTPUT2* - load and duplicate object code, filling unused space

LEVEL 6 ENTRIES

CDECN2 - pass II D- and E-type constant processor
8 SDDTRM*

ERRTAG* - create and save a scan pointer/error code pair

EVALUT - general expression evaluator routine
7 SDBCDX*, SYFIND*

SCANCO - scan expression until comma or blank found
7 SDBCDX*

LEVEL 7 ENTRIES

SDBCDX - determine type of self-defining term, call right processor
8 SDBTRM*, SDCTRM*, SDDTRM*, SDXTRM*

SYFIND* - find symbol in symbol table, return @ table entry, if any

LEVEL 8 ENTRIES

SDBTRM* - evaluate binary self-defining term

SDCTRM* - evaluate character self-defining term

SDDTRM* - evaluate decimal self-defining term

SDXTRM* - evaluate hexadecimal self-defining term

PART III - THE INTERPRETER

The interpreter section of ASSIST is a program which interpretively executes S/360 object code. It can perform all of the standard instruction set, and may permit decimal and floating point operations if these are desired. Although privileged operations and SVC calls are not performed at the current time, conditional code exists in the program for decoding them, and branching to individual sections of code to do each individual instruction. The interpreter also allows a number of simple I/O, debugging, and conversion pseudo instructions, which can be handled as macro instructions by regular S/360 assemblers, thus maintaining compatibility.

A. OVERALL CONTROL LOGIC

1. INPUT

a. ADDRESS OF EXECUTION CONTROL BLOCK (ECONTROL)

All execution parameters and workareas are contained in one block, which is passed to the interpreter by the calling program. This design makes for flexible use of the interpreter for several differing purposes, and keeps the interface between the interpreter and the rest of the system at a minimum. This table contains the following items of special importance, in addition to other things:

A block of four simulated floating point registers. The values here are used to initialize the real floating point registers, and the registers are stored back here when execution is completed.

A block of 16 simulated general purpose registers. These contain the initial values of the user registers on input, contain the contents of the user registers during execution, and are used to produce the completion dump after execution terminates.

A simulated Program Status Word.

Various byte flags for execution mode, special error codes, debug control, and completion dump condtrol.

Two instruction count limits, the first giving the maximum number of instructions to be executed, and the second for decrementing and testing.

Two addresses giving the lower and upper limits for a storage to be printed in a completion dump. These addresses may be changed during user program execution by execution of an XLIMD pseudo instruction.

A block of six addresses originally created by the ASSIST assembler, which describe the storage limits (both real and as given in the assembly listing), entry point address, and execution-time relocation factor for the user program.

An instruction stack (actually a circular linked list), in which is kept the last ten instructions done, with their addresses. This is used by the completion dump routine to produce an instruction trace.

Various other work words and execution-time values are also kept in ECONTROL.

b. DECK OF DATA CARDS

An (optional) deck of data cards may be read by the interpreter for the user program. An XREAD pseudo instruction in a user program causes execution of the code generated by a $READ macro in the interpreter, which obtains the next card, or gives an end-of-file indication. Under no circumstances is it possible for the user program to read beyond the end-of-file.

2. OUtPUT

a. PRINTED OUTPUT

The interpreter may have output printed, either using a $PRNT macro in response to an XPRNT insruction in the user program, or using an XSNAP macro to perform services requested by an XDUMP instruction.

b. PUNCHED OUTPUT

The interpreter may have cards punched, using an $PNCH macro when the user program contains an XPNCH pseudo instruction. Note that in some circumstances, the cards to be punched may be listed on the printer instead. See the INTRODUCTORY ASSEMBLER USER's MANUAL, part III, regarding the NOPUNCH option in the user PARM field.

c. VALUES IN THE EXECUTION CONTROL BLOCK (ECONTROL)

The interpreter sets various flags and values in the execution control block. These values may then be tested by the calling program to determine the reason for completion, and are used by the completion dump program to produce its output. Most of the variables were noted above in section A.1.a of this part. Others include the following:

Special error flag byte, which notes either a normal termination by a return to the address originally supplied to the user program as a return address, or else a code indicating one of several special completion codes (such as exceeding time or records, branch out of range, and others of type COMPLETION CODE ASSIST =).

The address of a completion code/error message block, which may be used by the completion dump routine.

3. OVERALL CONTROL LOGIC SUMMARY

The interpreter begins by initializing various values in the execution control block ECONTROL, and setting registers for its own use. After all initialization is complete, the actual interpretation begins.

The next instruction to be executed is fetched from user storage, and placed in the next instruction stack entry (ECSTACKD), along with the address of the instruction, condition code, and program mask. Some preliminary decoding is done, which increments the location counter by the length of the instruction, and sets up registers with several codes indicating actions to be performed for the instruction. A four-way branch is taken to separate the instructions into the following types: RR, RX, SI-RS, SS, each of which has a primary decoding section.

At the primary decoding section for each type, common processing for all instructions of that type is performed. This includes decoding register addresses, some fetching of simulated register values, decoding operand addresses, and checking of storage addresses for legality. After this has been done, each primary decoding section branches to one of a number of secondary decoding sections belonging to it. Each of these sections completes the interpretation for a single instruction, or for a group of instructions which can be handled in the same way. After this has been done, control passes back to code for checking legality of a successful branch, or to code for checking instruction count limit excession. The next instruction is fetched, and the cycle repeats.

Certain instructions can cause a call to an external routine. These include all the X-macro pseudo instructions, and any SVC calls.

4. ENTRY POINT HIERARCHY TABLE

As of 11/30/70, the interpeter consists of only one control section, and has no other internal modules. External module calls are shown by the overall hierarchy chart (Part I.A.4.). This section is included only for possible future use with SVC routines or special I/O routines which may be added.

C. OPTIONAL EXTENDED INTERPRETER

1. OVERVIEW.

The ASSIST Optional Extended Interpreter (EXECUT) was designed with two important ideas in mind:

> This interpreter would be more table driven in nature than the original interpreter. The flow of program logic would center around one large decoding table. This table although enlarging and slowing down the interpreter slightly, would make program logic easier to follow and update.
>
> The Optional Interpreter would support more s/360 -S/370 instructions as well as a new pseudo instruction, XOPC.

Overall program speed and program size were considered secondary in this design.

MACRO USAGE:

$SAVE	-	Used to save registers.
$RETURN	-	Used to restore registers.
$SPIE	-	Catch execution time interrupts.
EIXTAB	-	Create a secondary displacement table entry.
EITAB	-	Create a main table entry.

DSECT USAGE:

ECONTROL	-	Main interpreter interface with ASSIST system.
ECSTACKD	-	Structure of instructions executed stack.
ECBRSTAK	-	Structure of stack of branch instructions.

2. DECODING TABLES.

The logic of EXECUT (new extended) centers around two tables EIOPCDTB and EICONTAB. The first table, EIOPCDTB, is a 256 byte secondary control table ordered according to opcodes. Each byte of the table contains an index into the main control table EICONTAB. Bytes of the secondary control table corresponding to invalid opcodes contain entries pointing to special EICONTAB entries in the top of the main table. This allows for easy checking of operation exceptions.

The main contol table, EICONTAB, contains entries for every instruction (grouped or singularly) which are used in the decoding of each individual interpreted instruction. Each main table entry is 8 bytes in length. The first byte contains miscellaneous decoding flags; which machines the instruction is allowed to run on, does the instruction have an extended opcode, is the instruction a privileged instruction? The second byte of each table entry contains the length of the given instruction and is used in updating the ASSIST PSW. The third byte of each table entry contains flags telling about the kind of storage checking to be performed on the first and second operands of the instruction. The fourth byte of each table entry describes the type of storage alignment needed by the given instruction. The fifth byte of each table entry describes the type of register specification needed by the instruction (even or odd). The sixth table byte is a flag telling where in the instruction the length of storage being modified is located. Bytes 7 and 8 of the table represent a halfword displacement past program label EISPEJMP. This displacement is used in branching to the special routine which actually interprets each instruction.

These two tables are located at the bottom of the interpreter. The secondary control table (EIOPCDTB) is created using repeated calls to the EIXTAB macro. The main control table (EICONTAB) is created using repeated calls to the EITAB macro.

3. OVERALL PROGRAM LOGIC FLOW

Execut is called from the main control program (ASSIST). At entry to Execut, register 10 has the address of the ECONTROL block. Two base registers are used, R13 first then R12. Initialization of the instruction stacks is performed along with other initialization of the fake user registers and some work areas.

After initialization, decoding and interpretation of the first instruction begins. Each instruction is interpreted in the same manner, as follows: The instruction is inserted into the instruction stack and the ASSIST instruction counter is decremented checking for a timer run out. The opcode of the instruction is used as an index into the secondary control table (EIOPCDTB). The one byte value in this table is multiplied by 8 and is used as an index into the main table (EICONTAB). The main table entry for the instruction being interpreted is moved into an 8 byte work area (EICTNTRY). The decoding of the instruction entails checking which flag bits of the table entry are active. Several internal subroutines are executed to do checking not performed in the main decoding loop. The names of these and a short description follow:

EICHKST - Used when a privileged operation is found to check if the interpreter is in the Supervisor State

EITRIC - Used to provide the instruction trace (if enabled) and count instructions for the Instruction Execution Count Facility (if enabled)

EIBASDSP - Used to calculate an address from the base-displacement found in the instruction

EIMSFCHK - Used to check fetch and store addresses to see if they are in the program area.

Following the common decoding process bytes 7 and 8 of the main table entry for a given instruction are loaded into a register (R1). A branch on this displacement passed program label EISPEJMP is made to execute the special routine associated with each instruction.

The instructions were grouped where ever possible with regard to the special routines (i.e. 1 special routine may be used to interpret many instructions). Register usage in EXECUT remains the same no matter what the instruction. The opcode of the instruction being interpreted is moved into an instruction of the same format, where ever possible, and is actually executed. A spie is used to catch the user interrupts possible which were not already checked for.

After each instruction is interpreted, control is passed back to the top of the main decoding loop and this entire process is repeated.

ASSIST MACRO PROCESSOR INTERFACE SPECIFICATIONS

This section specifies the interfaces between the macro processsor modules and the remainder of the ASSIST assembler, also noting variables which may be of use to the macro processor, or are needed for various communications. The following items are given: MODULES, SET VARIABLES, VARIABLES, PARAMETERS.

**************************** INDEX TO THIS SECTION *********************

MODULES

MODULES

MACINT

MACINT is called 1 time by MPCON0 before Pass 1 of the assembly is begun. Any required initialization can be performed at this time. Internal initialization should be omitted if ASSIST is not in MACRO mode, and such code should be omitted, particularly any which obtains working storage. The following test can be used:

```
          TM   AVTAGSM,AJOMACRO
          BZ   NOINIT              skip initialiation
```

Even if no macro processing is done in a given run, the following initialization is REQUIRED, if the macro processor is generated at all:

1. The fullwords AVGEN1CD and AVGEN2CD should be set = AVADDHIH. This is required for checking purposes in INCARD and UTPUT1. During processing of macro expansions, AVGEN2CD contains the address of the first byte of the last (temporally) generated statement blocks from MEXPND, while AVGEN1CD contains the address of the byte beyond the first one created, so that INCARD works backwards until AVGEN1CD <= AVGEN2CD.

2. Any global translate tables modified by any macro code must be set to their correct values (with exception of 64 bytes of AWZEROS beginning at AWZEROS+C' '). This allows for possiblity of interrupt for time or space during a BATCH run.

MACRO1

MACRO1 can be called under any of the following circumstances:

1. When a MACRO command is encountered in the input source program, MACRO1 is called to read and process the entire macro definition which follows.

2. If ASSIST is generated with the macro library facility, MACRO1 may be called to scan a macro definition which is obtained from a macro library. MACRO1's processing is essentially the same as for 1. This type of processing is caused by finding an *SYSLIB card.

3. When a macro-type command (GBLx, LCLx, ACTR, SETx, AIF, AGO, etc) is discovered in open code, MACRO1 may be called to scan it, and perform desired action, which may include having a number of cards read for a forward branch of an AGO or AIF.

4. If a statement is discovered containing a SET symbol, MACRO1 can be called to have it scanned and appropriately expanded.

ENTRY CONDITIONS

REGISTERS

RA = scan pointer to first byte of OPERAND field, if any. If no operand is present, = @ 2nd blank beyond the opcode.

RC = @ OPCODTB entry for the statement, if any exists (i.e., for cases 1, 2, 3, but not 4).

AVWXTABL VARIABLES

AVREBPT = @ REBLK for stmt, if errors exist in it already.

AVRSBPT = @ RSBLOCK for the statement, which includes various flags and the source statement itself.

AVRSCPT = @ RSCBLK for the statement, if it is continued or has sequence numbers.

AVSOLAST = @ last blank before the afterquote.

AVOULNCN = 3-byte, packed decimal number of CURRENT statement, not next statement, can be used for diagnostics.

AVPRINT1 gives current print conditions, maybe tested to check on allowability of MACRO definition:

```
          TM   AVPRINT1,AVPRSAVE
          BO   NOTALLOWED
```

AVTAGSM   contains various flag bits of interest to macro proc.
Flag AJOMACRO is definitely on (else MACRO1 will never be called).
Flag AJOMACRG is on if Assembler G options allowed (FUTURE USE******).
Flag AJOMACRH is on if Assembler H options allowed (FUTURE USE******).
Flag AJOMACSL is on if user desired to obtain macros from library.

EXIT CONDITIONS

REGISTERS

RB = return code showing action to be taken by MOCON1.

= 0 ==> MOCON1 should call INCARD to obtain the next source cardimage fromthe card reader. This requires that MACRO1 has totally disposed of the current source card residing in RSBLOCK, etc. This is normal return for 1, 2, and 3(except AIF, AGO branches).

= 4 ==> the current statement exists in the usual record block area, is probably a generated statement, or the statement found when doing a forward read for open code AIF/AGO. It should essentially be processed as though it had just been read via INCARD, altough the processing may differ slightly for the two cases given, i.e., the stmt may or may not be a generated statement. This return would be a normal return for case 3 (AIF, AGO) and case 4.

= 8 ==> a statement exists in the usual record block position, but UTPUT1 should be called immediately to store it away, i.e., it probably has appropriate errors attached. This could occur for any macro-type statement found out of order (MACRO, GBL, LCL, etc), or perhaps for error messages caused by errors in AIF/AGOing in open code. This would be an abnormal return from cases 1,2,3.

SUMMARY OF POSSIBLE RETURN CODES FOR THE CASES

CASE	0	4	8
1	NORMAL	NO	ORDER ERROR
2	NORMAL	NO	ORDER ERROR
3	NORMAL	NORMAL(AIF,AGO)	ORDER ERROR, OTHER ERROR
4	NO	NORMAL	NO

GENERAL NOTES ON MACRO1

MACRO1 should allocate space when needed from the low end of the dynamic area, using $ALLOCL, so that this space may eventually be overlaid with object code.

When scanning a macro definition, opcodes may be discovered which are not defined. These should be added to the list of macro names, but marked as not yet defined. It is expected that they will either turn up later in the user program, or else (if SYSLIB option exists), be probably found in the library.

When a macro prototype is scanned, the name of the macro might already be in the macro table, so that a new block should not be gotten for the macro, but he existing one used instead. Double defn of a macro occurs only when a prototype is found, and it is already actually defined, not just present in the table.

INDIVIDUAL CASES FOR CALLING MACRO1

1. MACRO STATEMENT FOUND IN INPUT.

MACRO1 identifies the opcode as MACRO, by checking the OPCTYPE flag in the OPCODTB passed to it. It should check to make sure that MACRO is allowed at that point. If not, it should return immediately noting that the statement is out of order.

Assuming the MACRO statement is accepted, it should be printed via OUTPT2, and the prototype statement obtained via INCARD, and scanned. Three cases exist for the name of the macro:

a. It may not have been encountered before at all. In this case, it should be added to the list of macros, marked as DEFINED, and the appropriate control block filled in as scanning the definition continues.

b. It may have been previously defined. An error message should be issued, and rest of the macro scanned.

c. It may be present in the macro list, but NOT defined. This would occur when an undefined opcode is encountered during a previous macro definition, and so is added to the list, but marked undefined. The control block for the macro should be then marked defined, and processing continued as though the name had not yet been seen.

After scanning the prototype and printing it via OUTPT2, MACRO1 should call INCARD repeatedly to obtai1the rest of the definition, allocating storage for control blocks as needed from dynamic-low, using $ALLOCL.

Note: when calling OUTPT2, the following setup is appropriate:

```
RB = $OUCOMM  :  notes that no location counter, etc is needed.

RSBFLAG:  actual statements: do not modify.
       :  special error messages: $RSBNPNN+$RSBMERR .
```

2. MACRO FROM SYSLIB

This situation is handled exactly as is a macro from the input stream. The only difference is that the PRINT is turned OFF by the macro library processor, so that the macro definitions do not get printed by OUTPT2.

3. MACRO-TYPE COMMAND IN OPEN CODE

MACRO1 is called when anything identified as a conditional assembly statement is found. The actions taken are as follows: (NOTE: RSBFLAG should be marked $RSBNP## for all stmts)

a. GBLx, LCLx, ACTR

Check for proper order, flag if incorrect, using ERRTAG, and set RSBFLAG with $RSBNP##, so that the statement will be numbered, but will not be processed except to be printed during Pass 2. Return with RB =8 so that MOCON1 will call UTPUT1 and then get next statement.

If statement order is acceptable, perform required actions. Storage maybe allocated using $ALLOCL from dynamic-low, or if desired, using $ALLOCH from dynamic-high, with low being much preferred.

When the statement is processed, it should be passed to OUTPT2 and printed (RB=$OUCOMM).

b. SETx

The statement is scanned, and the indicated action performed. If any errors exist, they may be flagged using ERRTAG, and return should be made with RB =8, and also, the RSBFLAG should be flagged $RSBNP##.

c. AIF, AGO

The statement is scanned, and indicated action performed. AIF's which fail are treated exactly as are SETx.

AGO's and successful AIF's cause the following actions to occur: First, UTPUT1 is called to save the current AIF/AGO stmt (whose RSBFLAG should be marked $RSBNP##). Next, INCARD is called until the desired sequence symbol is discovered, at which point return should be made with RB = 4, thus allowing MOCON1 to treat the current statement as the one just read.

NOTE: if an end-of-file is encountered by INCARD, it will generate an END card automatically (and will do this every time that it is called after an end-file occurs). Thus, MACRO1 may just flag the END card with an appropriate message (ERRTAG), and return normally (RB = 4). This message may be omitted, since it might be obvious what has occurred.

4. STATEMENT WITH POSSIBLE SUBSTITUTION OF SET VARIABLE

In this case, RC (OPCODTB ptr) = 0. The statement should be scanned, and substitution performed if needed. If any is actually done the original statement should have its RSBFLAG marked $RSBNPNN, and should be passed to UTPUT1 to save it. Then it can be expanded, thus creating a new RSBLOCK overlaying the previous one, which should be marked $RSBGENR, not only to show that it is generated, but also to stop macro-type statements from being generated. Return is made with RB = 4, so that the generated statement is scanned and processed appropriately.

MEXPND

This module is called by MOCON1 to expand macro calls, and is only called under the following circumstances:

1. An unknown opcode is encountered.

2. It is not already a generated statement (i.e., $RSBGENR is not already set on in RSBFLAG).

3. AVTAGSM is flagged with AJOMACRO.

When called, MEXPND may check for the opcode being a defined macro immediately, and return with RB = $ERIVOPC if it is not. If it is set, MOCON1 flags it using ERRTAG and continues. Possible expansion is shown by RB = 0 on return. If desired, MEXPND may set RB = 0 always and always show the outer level macro as a generated statement.

If the call is for a defined macro, MEXPND expands it as follows:

1. Workareas and stack space are allocated as desired, from the dynamic-low area, with no restrictions on boundary alignment of the pointer AVADDLOW.

2. Space is reserved for an error message statement immediately below the current address in AVADDHIH. This may be used if macro expansion causes an overflow.

3. Initialization is done as desired for the entire macro nest expansion. This includes initializing a variable to 0, to be used as a counter for macro nest level, to be compared to AVMMNEST as a limit. This counter is incremented by 1 every time a macro is called, and decremented for each MEND or MEXIT.

4. Processing of the outer macro and any inner macros begins.

MEXPND places generated statements in the dynamic-high area, with the first generated statement at the highest actual memory address, working lower as statements are generated. When exgenerating statements MEXPND does the following:

1. Sets AVGEN1CD = AVGEN2CD = AVADDHIH (@ lowest used byte so far, always on fullword boundary). AVGEN1CD will then remain unchanged. Macro call is scanned

2. For each statement generated, appropriate information is placed starting at the address (AVGEN2CD) - 1, and working backwards, as described below. At the end of processing for each statement, AVGEN2CD should contain the address of the (temporally) last byte of information saved, i.e., the lowest address of usable information. During this process, AVADDHIH should NOT be modified. It may also be desirable to create a dummy first block, in order to allow for symbol table/literal table expansion from AVADDHIH downward, without running over generated statements.

3. ERROR HANDLING: if the MACTR, MNEST, or MNEST counters are overrrun during expansion, an appropriate message should be generated and placed as a generated statement, withe flags $RSBNPNN+$RSBMERR, or at least $RSBNPNN, with latter case used if it is not to be counted as an actual error.

If storage overflow occurs, MEXPND should cancel the entire nest of genrated code, and place an appropriate message as a generated stmt, so that the user will be informed. It should also set the AVOVERFL bit on in AVTAGS3, s that a message will be printed at the end of the assembly.

ENTRY CONDITIONS

REGISTERS

RA = scan pointer to first character of the opocde.

VARIABLES

AVSOLAST is set appropriately, as limit to scanning required.

EXIT CONDITIONS

RB = 0 ==> 0 or more generated statements exist in dynamic-high and MOCON1 should call INCARD continue calling INCARD to obtain them.

RB = nonzero value ==> error in macro call statement which prevents it from being expanded, such as being an undefined or invalid opocde, or any other reason which prevents expansion. In this case, RB = error code to be supplied to ERRTAG, and RA = scan pointer to the error. MOCON1 will flag the statement immediately, and call UTPUT1 to save it. Also, the RSBFLAG should be set to show RSBNP##.

FORMAT OF GENERATED STATEMENT BLOCKS

The following describes the layout of generated statements. AVGEN1CD contains the address of the first byte following the last byte generated for the first statement generated, while AVGEN2CD has the address of the first byte of the last statement generated, so that AVGEN2CD <= AVGEN1CD .

Essentially, an information block for a generated statement consists (in descending order of addresses) of the fixed part of an RSBLOCK (slightly modified), the variable part of one (source stmt), and optionally, the error code/scan pointer sections of a REBLK, if there are any such errors in the statement.

The following gives the layout of the block. The ADDRESS field is given relative to the ORIGINAL value of AVGEN2CD, before the code was generated.

ADDRESS	NAME(if any)	description
-1	RSBSCAN	reserved for future use
-2	RSBNUM	= length-1 of REBLK, if exists, i.e., will become REBLN.
-3	RSBFLAG	flag byte for RSBLOCK (see notes below)
-4	RSBLENG	length-1 of generated statement. This value + RSB$L will become the actual RSBLENG for the generated statement.
-5		last byte of generated statement
-5-(RSBLENG)		first byte of generated statement
-5-(RSBLENG)-1		last byte of REBLK, if exists
-5-(RSBLENG)-1-(RSBNUM)		first byte of error code/scan ptr part of REBLK. RSBNUM will become REBLN.

NOTE: this setup assumes RSB$L = 4 (length of RSBLOCK fixed section).

NOTES ON RSBLFAG: the following are possible combinations of flags in RSBFLAG, with what they are used for:

$RSBGENR	normal generated statements, with no local (ERRTAG) errors attached already. No REBLK exists inthis case.
$RSBNP##	a macro call, will not be further processed, but will be numbered. Also COMMENTS cards.
$RSBNPNN+$RSBMERR	a special error message, will not be further processed expcet for printing, but is printed specially, as error message.
$RSBGENR+$REBX	like $RSBGEBR, except some normal errors are are already attached.
$RSBNP##+$REBX	for any illegal statment such as illega opcode, so that MOCON1 doesn't waste time looking it up again.

APPENDIX I: GENERAL CONVENTIONS AND INFORMATION

A. PROGRAM DOCUMENTATION

1. PHILOSOPHY AND GENERAL DESCRIPTION

In general, the documentation philosphy followed inside ASSIST is to put as much documentation as possible inside the source program to keep it from being separated from the program, and to keep it in machine-readable form. Commments cards are set up in such a way that comments of a global nature (e.g. subroutine entry/exit conventions, dummy section descriptions, etc.) can easily be extracted from the source program and printed in summary form.

ASSIST documentation is reasonably heavy. Approximately 20 % of all source cards in the system are comments cards. At least 95 % of all machine instructions and macro calls have comments with them. Many assembler instructions and conditional assembly instructions also have comments.

In addition to comments, program readability is aided by liberal use of SPACE, EJECT, and TITLE cards to block off logical parts of the program. Every control section, and most macros and dummy sections are titled.

2. INTERNAL DOCUMENTATION FOR SUMMARY USE

Certain sections of the system have comments cards which are not only useful for understanding the sections to which they belong, but which may be required as part of a summary. These sections include control sections, dummy sections, entry points, internal subroutines, and macro instruction definitions. In general, the most important comments for a section immediately precede it, and are completely blocked off by special characters, in order to make them stand out.

The general form of summary documentation is as follows:

```
**--> atype: objectname  brief statement of purpose                  *
*    descriptive information                                         *
*    delimiter line  (*'s, .'s. or +'s).                             *
```

atype gives the type of object described, and is one of the following: CSECT, DSECT, ENTRY, INSUB, or MACRO .

objectname gives the name of the section being described.

Each of the different types uses certain character combinations to flag the information in the block and make it easy to pick out the comments cards containing important information of a given sort. The differences are as follows:

atype	first 2 characters	delimiter cards, margins
CSECT	*.	.
DSECT	*.	.
ENTRY	*.	.
INSUB	*+	+
MACRO	.*	*

```
**--> CSECT: XREFA   CROSS REFERENCE CONTROL SECTION...................
*.         WRITTEN BY ALICE FELTE,ALAN ARTZ, AND RICH LONG            .
*.                                           ---SPRING/SUMMER 1973    .
*.                                                                    .
*.  THIS CSECT IS THE MAIN CONTROL SECTION FOR THE CROSS REFERENCE    .
*. FOR ASSIST. IT HAS THREE ENTRY POINTS WHICH WILL BE DESCRIBED LATER.
*. THIS ROUTINE CONTROLS ALL THE CROSS-REFERENCE FACILITY IF IT IS TO .
*. BE GENERATED.  THE FIRST PASS THE FLAGS AND LOCATION COUNTER ARE   .
*. INITIALIZED--XRINT1.  SPACE IS ALLOCATED FOR THE CROSS-REFERENCE   .
*. ENTRIES AND NECESSARY FLAGS ARE SET FOR THE SECOND PASS--XRINT2.   .
*. THE *XREF CARD WILL BE SCANNED BY XRSCAN.                          .
*.                                                                    .
*.  XRINT1: PASS ONE INITIALIZATION                                   .
*.         CALLED FROM MPCON0.                                        .
*.         1)  INITIALIZE THE ADDITIONAL LOCATION COUNTER,            .
*.             AVXRLNCN, TO 1.                                        .
*.         2)  INITIALIZE THE COUNTER, AVXRCNT, FOR THE NUMBER OF     .
*.             REFERENCES TO 0.                                       .
*.                                                                    .
*.  XRINT2: PASS TWO INITIALIZATION                                   .
*.         CALLED FROM MTCON2.                                        .
*.         1)  ALLOCATE SPACE USING THE MACRO $ALLOCH TO THE          .
*.             DSECT, XREFTAB, SIZE * THE NUMBER OF REFERENCES        .
*.             TO BE COLLECTED AND INITIALIZE ALL SPACE TO 0.         .
*.         2)  SET AVXRLAVS TO FIRST FREE NODE.                       .
*.         3)  SET HEADER NODE FOR THE TREE STRUCTRUE EQUAL TO 0.     .
*.                                                                    .
*.  XRSCAN:  CARD SCANNING ROUTINE.                                   .
*.             A FLAG IS PASSED IN A REGISTER TO DETERMINE WHICH      .
*.         PASS IS BEING PROCESSED.  FOR THE FIRST PASS, SCAN THE     .
*.         CARD AND SET THE SD FLAG ACCORDINGLY.  FOR THE SECOND      .
*.         PASS, SCAN THE CARD AND SET THE SR FLAG ACCORDINGLY.       .
*.                                                                    .
*......................................................................

**--> ENTRY: XRINT1     PASS ONE INITIALIZATION........................
*.       THIS IS CALLED FROM MPCON0 ONLY ONCE                         .
*.       MODULE DESCRIPTION--                                         .
*.            INITIALIZES AVXRLNCT, THE ADDITIONAL LINE COUNTER, TO 1 .
*.       AND AVXRCNT, COUNTER FOR THE NUMBER OF REFERENCES FOUND, TO 0.
*.                                                                    .
*......................................................................
```

```
**--> ENTRY: XRINT2     PASS TWO INITIALIZATION.........................
*.       THIS IS CALLED FROM MPCON0 ONLY ONCE.                         .
*.       MODULE DESCRIPTION--                                          .
*.            ALLOCATES A BLOCK OF SPACE USING $ALLOCH WHERE THE SIZE .
*.       IS AVXRCNT * XRSIZE.  IT SET AVXRLAVS TO THE ADDRESS OF THE  .
*.       BEGINNING OF THE BLOCK OR FIRST FREE NODE AS RETURNED BY     .
*.       $ALLOCH.  IT ALSO SETS AVXRHEAD, THE HEADER POINTING TO THE  .
*.       FIRST ENTRY IN THE TREE, EQUAL TO 0.                          .
*.                                                                     .
*.......................................................................

**--> ENTRY: XRSCAN     CARD SCANNING ROUTINE...........................
*.       THIS IS CALLED FROM MOCON1 AND MTCON2 TO SCANN THE *XREF CARD.
*.                                                                     .
*.       ENTRY CONDITIONS--  RA  @ TO BEGIN *XREF PARM SCAN            .
*.                           RD  IDX TO SET FLAGS(0=PASS 1,8=PASS 2)   .
*.                                                                     .
*.       MODULE DESCRIPTION--                                          .
*.            CHECK TO SEE WHICH PASS IT IS IN.  DEPENDING ON WHICH    .
*.       PASS IT IS, THE *XREF CARD IS SCANNED AND THE FLAGS SET.      .
*.            IF IT IS PASS ONE, THE CARD IS SCANNED FOR SD=.  IF IT  .
*.       NOT THERE, AVXRFLAG IS NOT CHANGED.  IF IT IS, CHECK FOR     .
*.       LEGAL VALUES OF *, 0, OR 1.  IF IT IS NONE OF THESE THREE,   .
*.       THE STATEMENT IS FLAGGED WITH A SYNTAX ERROR.  IF IT IS A    .
*.       LEGAL VALUE, THE AVXRFLAG IS SET ACCORDINGLY.                 .
*.            IF IT IS PASS TWO, THE CARD IS SCANNED SR= AND IS        .
*.       PROCESSED SIMILARLY TO SD= ABOVE.                             .
*.                                                                     .
*.......................................................................
```

```
**--> CSECT: XRCOLL     COLLECTION ROUTINE.............................
*.       THIS IS CALLED BY SYFIND AFTER IT IS FOUND THAT THE SYMBOL   .
*.       IS DEFINED AND THE REFERENCE IS TO BE COLLECTED.             .
*.                                                                    .
*.       ENTRY CONDITIONS--  RA  HAS THE ADDRESS OF THE SYMBOL IN THE .
*.                                SYMBOL TABLE.                       .
*.                                                                    .
*.       MODULE DESCRIPTION--                                         .
*.          AVXRHEAD HAS THE ADDRESS OF THE FIRST NODE IN THE TREE.   .
*.          AVXRLAVS HAS THE ADDRESS OF THE FIRST AVAILABEL FREE NODE .
*.                                                                    .
*.            THE FOLLOWING ALGORITHM IS FROM "THE ART OF COMPUTER    .
*.       PROGRAMMING" VOL. 1  'FUNDAMENTAL ALGORITHMS' BY DONALD KNUTH.
*.            CHECK HEADER 'AVXRHEAD' FOR EMPTY TREE(= 0).  IF EMPTY, .
*.       EXECUTE INSUB 'XRCLAVS' TO GET FREE NODE FOR PROCESSING.     .
*.       'XRCLAVS' INSERTS SYMBOL AND INITIALIZES LINKS IN NODES---   .
*.       LEFT LINK=0,RIGHT KINK=-1 (ODD DISPLACEMENT IMPOSSIBLE, NEGA-.
*.       TIVE TO SIMPLIFY CHECKS IN XRPRNT ROUTINE).  IF NOT EMPTY,   .
*.       DETERMINE WHETHER OR NOT A NODE HAS ALREADY BEEN CREATED FOR .
*.       THE PRESENT SYMBOL BY COMPARING THE ADDRESS OF THE SYMBOL    .
*.       IN REG RA TO THE ADDRESSES OF SYMBOLS ALREADY IN THE TREE    .
*.       NODES.  IF EQUAL, PROCESS THE REFERENCE (DESCRIBED LATER).   .
*.       OTHERWISE, COMPARE ACTUAL SYMBOLS TO DETERMINE WHERE IN THE  .
*.       TREE THE NEWLY CREATED NODE SHOULD BE INSERTED.  IF THE NEW  .
*.       SYMBOL IS SMALLER IN VALUE THAN THAT OF A NODE IN TREE, THE  .
*.       COMPARISON CONTINUES WITH IT'S LEFT SUBTREE.  IF LARGER, COM-.
*.       PARISON CONTINUES WITH RIGHT SUBTREE.  WHEN A ZERO LEFT LINK .
*.       IS FOUND, OR NEGATIVE RIGHT LINK, THE LINK IS CHANGED TO     .
*.       POINT TO THE NODE WHICH WILL CONTAIN THE INFO FOR THE NEW    .
*.       SYMBOL(NODE FETCHED AND INITIALIZED BY 'XRCLAVS'.            .
*.                                                                    .
*.       PROCESSING THE REFERENCES:                                   .
*.            ONCE THE SYMBOL IS PLACED IN THE TREE, THE REFERENCE    .
*.       MUST BE ENTERED IN A BLOCK OF REFERENCES.  THIS IS DONE IN   .
*.       THE FOLLOWING MANNER:                                        .
*.            1)  IF THE PTR TO THE BLOCK OF REFERENCES IS NULL       .
*.                (I.E. FIRST REFERENCE), A BLOCK MUST BE             .
*.                ALLOCATED AND THE ADDRESS PLACED IN THE POINTER     .
*.                OF THE XREFTAB.                                     .
*.            2)  IF IT IS NOT NULL, THE POINTER IS AN ADDRESS AND    .
*.                THE BLOCK CAN BE LOCATED.                           .
*.                                                                    .
*.            3)  THE FIRST FULLWORD OF THE REFERENCE-BLOCK           .
*.                CONTAINS EITHER:                                    .
*.                A)  THE NUMBER OF SLOTS LEFT IN THE BLOCK.          .
*.                    THE REFERENCE MAY BE ENTERED IN THE BLOCK, THE  .
*.                    NUMBER OF SLOTS IS DECREMENTED BY 1.            .
*.                B)  NEGATIVE ADDRESS OF AN ADDITIONAL BLOCK         .
*.                C)  ZERO, MEANING A NEW BLOCK MUST BE ALLOCATED.    .
*.                    ALLOCATE A NEW BLOCK AND SET THE POINTER IN     .
*.                    PRECEDING BLOCK TO IT (NEGATIVE ADDRESS). THEN  .
*.                    A) MAY BE FOLLOWED.                             .
*......................................................................
```

```
* * * *  REGISTER USAGE: XRCOLL * * * * * * * * * * * * * * * * * * * *
*   R0=  X'0000FFFF'               USED TO INITIALIZE NODE LINKS      *
*   RW=  @ NODE IN XREF LIST BEING CHECKED  (@ XREFTAB)               *
*   RX=  @ SYMSECT OF SYMBOL ALREADY IN XREF TABLE                    *
*   RA=  @ SYMSECT OF SYMBOL TO BE CHECKED IN XREFTAB                 *
*   RB=  @  BEGIN OF XREF TABLE (FROM WHICH OFFSETS COMPUTED)         *
*   RC,RD,RE,RY,RZ   WORK REGISTERS                                   *
*   R14= INTERNAL LINKAGE                                             *
*   R15= BASE REGISTER                                                *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

*.-->  INSUB: XRCLAVS . . . . . . . . . . . . . . . . . . . . . . . . .
*.            GET THE FIRST FREE NODE FROM THE LIST OF AVAILABLE      .
*.       SPACE, AVXRLAVS.  SETS AVXRLAVS TO POINT TO THE NEW FIRST    .
*.       FREE NODE.  STORES THE ADDRESS OF THE SUMBOL'S SYMSECT ENTRY .
*.       IN THE NEW NODE.                                             .
*.       RW  HAS THE ADDRESS OF THE NEW NODE                          .
*.       RX  HAS ADDRESS OF OLD NODE                                  .
*.       LEFT LINK INITIALIZED TO ZERO; RIGHT LINK TO -1              .
*.       NOTE: IT IS POSSIBLE TO HAVE THREAD OF A NODE POINT BACK TO  .
*.       ROOT NODE WHICH HAS INDEX DISPLACEMENT OF ZERO.  SINCE -0 IS .
*.       NOT DISTINGUISHABLE FROM +0, THE END OF THE TREE IS DENOTED  .
*.       BY -1 VICE 0                                                 .
*. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  .

**--> CSECT: XRPRNT     PRINT ROUTINE.................................
*.       CALLED FROM MPCON0 TO PRINT OUT THE CROSS REFERENCE.         .
*.            THE COMPRESS BIT OF AVXRFLAG IS TESTED BY AVXRCOMP TO   .
*.       DETERMINE WHICH FORMAT TO USE FOR PRINTING.  IF IT IS OFF,   .
*.       EACH REFERENCE SYMBOL IS PRINTED ON A NEW LINE.  IF IT IS ON,.
*.       THE REFERENCED LABELS ARE PRINTED MORE THAN ONE PER LINE IF  .
*.       THERE IS ROOM.                                               .
*.            THE FOLLOWING ALGORITHM IS FROM "THE ART OF COMPUTER    .
*.       PROGRAMMING" VOL. 1  'FUNDAMENTAL ALGORITHMS' BY DONALD KNUTH.
*.            THE TREE IS THEN TRAVERSED IN POSTORDER.                .
*.                GET THE ADDRESS OF THE FIRST NODE IN THE TREE FROM  .
*.            AVXRHEAD.  IF IT IS 0, PRINT A MESSAGE THAT NO SYMBOLS  .
*.            HAVE BEEN REFERENCED.  IF IT IS NOT 0, FOLLOW THE LEFT  .
*.            LINKS UNTIL IT IS 0.  THEN PRINT THE SYMBOL FROM THE    .
*.            NODE AND ALL ITS REFERENCES.  NOTE: A NEGATIVE          .
*.            REFERENCE IS A MODIFY AND A POSITIVE REFERENCE IS A     .
*.            FETCH.  IT IS PRINTED ACCORDING TO THE FORMAT DESCRIBED .
*.            ABOVE.                                                  .
*.                THEN THE RIGHT LINK IS CHECKED.  IF IT IS -1,WE ARE .
*.            AT THE END OF THE TREE AND RETURN TO ASSIST.            .
*.            IF IT IS LESS THAN -1,IT IS A THREAD BACK TO A NODE.    .
*.            GET THE POSITIVE ADDRESS OF THE NODE, PRINT THE SYMBOL  .
*.            AND ITS REFERENCES.  CHECK THE RIGHT LINK AGAIN.        .
*.            IF IT IS POSITIVE, IT IS THE ADDRESS OF THE NEXT NODE.  .
*.            GO TO THAT NODE AND CHECK ITS LEFT LINK AS ABOVE.       .
*.                                                                    .
*......................................................................
```

```
* * * *  REGISTER USAGE: XRPRNT * * * * * * * * * * * * * * * * * * * *
*   RW=  @ CURRENT XREFTAB ENTRY PROCESSED                            *
*   R0=  LAST @ TO START STMT # (COMPRESSED OUTPUT)                   *
*   R2=  -1  DENOTES END OF TREE                                      *
*   RA=  LAST @ TO START A SYMBOL (COMPRESSED OUTPUT)                 *
*   RB= @ XREFBLK BEING PROCESSED                                     *
*   RC,RD,RZ   WORK REGISTERS                                         *
*   RE=  @ OF 1ST ELEMENT (BASE FROM WHICH OFFSETS GIVEN)             *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

*.--> INSUB: XRPRLINE       PRINTS A LINE OF REFERENCES . . . . . . . .
*.       SETS RZ TO POINT TO THE BEGINNING OF THE LINE. CLEAR         .
*.       OUTPUT LINE TO ALL BLANKS.                                   .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ..

*.--> INSUB: XRPNUMSL . . . . . . . . . . . . . . . . . . . . . . . . .
*.       GETS INDEX TO 1ST REFERENCE OF BLOCK, INDEX TO LAST REFERENCE.
*.       TO PRINT, AND VALUE FROM XRBLKNUM TO USE AS FLAG FOR TEST FOR.
*.       ADDITIONAL BLOCKS LATER IN MAIN SECTION OF CODE.             .
*.         RX=  INDEX TO 1ST REFERENCE TO BE PRINTED                  .
*.         RD=  INDEX TO LAST REFERENCE TO BE PRINTED                 .
*.         RY=  FLAG USED LATER(IF - THERE IS AN ADDITIONAL BLOCK)    .
*......................................................................
         SPACE 2
```

a. CSECT

Each of the different types of blocks may contain certain specific kinds of information, in addition to general descriptive text. These are also coded in specific ways to facilitate future production of lists and indices. The individual types are described as follows:

The information in a CSECT block generally describes overall properties of the control section. In some cases, the text may be very short, if the control section has a large number of entry points with a fair amount of comments. If all of the entry points of a csect call the same subroutines, use the same set of dummy sections, or use the same set of macros, these will be noted in the block for the csect. If the csect is itself an entry point, the block may contain any of the information described below under ENTRY.

b. DSECT

In addition to descriptive text, a DSECT block may contain any of the following types of comments cards:

*.   LOCATION: where in the program the data described by the   .
dsect resides, such as in a specific table of a csect.

*.   NAMES: notes the first 2-3 characters which begin all   .
names belonging to the dsect.

*.   GENERATION: if this dummy section describes a data block   .
which is generated (all or in part) by a specific macro, the macro name(s) used are noted here.

c. ENTRY

In addition to descriptive information giving the purpose of the subroutine and possibly when it is called, any or all of the following may appear if appropriate:

*.   ENTRY CONDITIONS   .

Following this statement is a list of the entry conditions for the entry point, usually consisting of a list of the parameter registers for the entry point and their usage.

*.   EXIT CONDITIONS   .

This statement precedes a list of the exit conditions for the entry, which is normally a list of parameter registers and their usage.

*.   CALLS list of entry point names, separated by commas.   .

A CALLS statement gives an alphabetical list of all entry points which may be called by this subroutine (when entered form the entry point associated with the block), with the exception of the special subroutines which can only be called by macro expansions and thus do not necessarily follow normal linkage conventions. These include the input/output and debugging modules (XXXXREAD, XXXXSNAP, etc). One or more CALLS statements may be needed to complete the list.

*.   USES DSECTS: list of dummy sections referenced inside the section of code, in alphabetical order.

*.   USES MACROS: list of highest-level macros used by the section of code, in alphabetical order.

*.   NAMES: description of any additonal restrictions on the labels used in this section of code.

d. INSUB (INternal SUBroutine)

Internal subroutine comments normally include a brief explanation of the section's purpose, followed by ENTRY CONDITIONS and EXIT CONDITIONS, specified similar to those of an ENTRY.

Note, as of 9/1/70, many internal subroutines do not completely follow the standard format, using '+' signs as delimiters, since this was only started recently.

e. MACRO

Macro documentation includes a brief statement of the purpose and usage of the macro, with a list of the macro parameters. If the macro uses other macros, the following is included:

```
.*        USES MACROS:   list of all macros directly called.          *
```

B. REGISTER AND SUBROUTINE LINKAGE CONVENTIONS

1. REGISTER EQUATE SYMBOLS

In order to facilitiate debugging and comprehensibility of the program, symbolic registers are always used inside ASSIST. A number of different sets of register equate symbols are provided, and they are as follows:

a. ABSOLUTE REGISTER EQUATES

The standard set of equates of R0-R15 for general purpose registers 0-15, and the set of equates F0-F6 for the floating point registers 0-6 are included, and these symbols are used whenever more symbolic equates are not approriate. For instance, these are always used for registers 0, 1, and 2, since they have special properties.

b. SYMBOLIC REGISTER EQUATES (MAINLY FOR THE ASSEMBLER)

The following set of equates is provided mainly for use inside the ASSIST assembler, but they may also be used by other parts of the system which communicate with the assembler:

```
RW        EQU   R3                  GENERAL WORK REGISTER 1
RX        EQU   R4                  GENERAL WORK REGISTER 2
RY        EQU   R6                  GENERAL WORK REGISTER 3
RZ        EQU   R6                  GENERAL WORK REGISTER 4

RA        EQU   R7                  PARAMETER REGISTER 1
          This register is commonly used as a scan pointer register
          inside the assembler.
RB        EQU   R8                  PARAMETER REGISTER 2
          This register is commonly used to pass a control value to
          a subroutine, and on return, almost always contains either
          an error code, or a zero to show no errors.
RC        EQU   R9                  PARAMETER REGISTER 3
          This register is most often used in the assembler for passing
          a 24-bit value (such as the result of an expression or a
          self-defining term).
RD        EQU   R10                 PARAMETER REGISTER 4
RE        EQU   R11                 PARAMETER REGISTER 5
          Registers RD and RE may be used for subroutines needing more
          than two or three arguments, but are more commonly used as
          work temporary work registers.
```

```
RAT       EQU   R12                ASSEMBLER TABLE POINTER-READ ONLY
          This register points the main assembler table (VWXTABL csect,
          AVWXTABL dsect) during an assembly.  No subroutine in the
          assembler may modify this register.
RSA       EQU   R13                SAVE AREA POINTER/BASE REG FOR SOME
          This register is used to point to an OS/360 save area, for
          any subroutine which may call another.  Almost all subroutines
          use this as a base register if they are not lowest-level
          routines.
RET       EQU   R14                RETURN ADDRESS USED IN CALLS
          This is used in subroutine linkage for the return address to
          a calling program.  This symbol is generally used whenever
          subroutine linkage is being set up, while R14 is used when the
          register is being used as a temporary work register.
REP       EQU   R15                ENTRY POINT ADDRESS/OFTEN USED BASE
          This register is used to hold the entry point address for all
          subroutines in the assembler.  Lowest-level routines usually
          use this as a base register.  In other routines, this may be
          used as a local work register, in which case the symbol R15
          is normally coded.
```

c. OTHER REGISTER EQUATE SYMBOLS

In addition to the two main sets of equates mentioned above, the ASSIST interpreter EXECUT has a local set of equates, and several routines of the assembler have a few register equates also. These sets are not currently used outside the control sections to which they belong.

2. LINKAGE CONVENTIONS - THE ASSEMBLER

The linkage conventions inside the ASSIST assembler consist of a few modifications to the standard OS/360 linkage conventions, which have been changed mainly to save time and space. The differences are as follows:

a. Registers R0-R6 (or R0-R2, RW-RZ) are protected across any calling sequence and must be restored if changed. R14 (RET) must also be restored if changed before returning.

b.Register R12(RAT) may not be changed by any routine.

c. Registers R7-R11 (RA-RE) are used for parameters and temporary work registers, and are not protected at all across calls. No routine ever requires more than five arguments, so these five registers are sufficient.

d. Except for the above, all normal OS/360 conventions are followed regarding save area linkage requirements and usage. In general, most routine only save as many registers as required. Lowest-level routines use R15 as a base, and do not perfrom save area linkage, other routines usually use R13 as a base and save area pointer. Many of the lowest-level routines save no more than one or two registers.

C. NAMING CONVENTIONS

1. CONTROL SECTIONS

All labels in a control section begin with the first two characters of the control section name. The single exception to this rule is the csect VWXTABL, which contains names beginning with V, W, and X.

All normal csect and entry point names are six characters long, except for those in the intrinsic routines which are only callable by macro expansions. These are all eight characters long, and begin with the characters 'XXXX' (such as XXXXSNAP, XXXXIOCO, etc).

In the assembler portion of ASSIST, entry points used only in the first or second assembly pass end either with a 1 or 2, respectively.

2. DUMMY SECTIONS

All labels in a dummy section begin with the first two-three characters of the dummy section name. Dummy sections used only inside one control section normally use the first two characters of its name, followed by a third character to distinguish the dummy section. The single exception to this rule is the dsect AVWXTABL, which contains symbols beginning with AV, AW, and AX.

3. MACROS

In general, macro names beginning with the character $ are global macros, and are liable to be used in more than one control section. A macro used only in one control section begins with the first two characters of the control section name. In addition, all the X-macros (XSNAP, XSAVE, XIDENT, etc) are also global macros, and do not follow the above conventions because they already existed before ASSIST was written.

Any macro which calls an intrinsic routine ends with the same four characters as the entry point which it calls (such as XSNAP-XXXXSNAP, $READ-XXXXREAD, XDECI-XXXXDECI, etc).

4. SET SYMBOLS

Set symbols used in ASSIST are of two kinds: the first are directly given values to indicate the options desired for an ASSIST generation. Their names all begin '&$'. The second kind are for internal usage, and may be indirectly set from values given by the first type. There are no restrictions on the names of these symbols.

5. REGISTERS

Register equate symbols normally begin with 'R', but a few routines have some local equates, which begin with the first two characters of the control section in which they are used.

6. MISCELLANEOUS INFORMATION

In general, symbols beginning with '$' are of a global nature. In addition, no symbols contain the characters '@' or '#', which are reserved for future expansions of ASSIST.

D. CODING CONVENTIONS AND TYPICAL TECHNIQUES

1. SYMBOLIC CODING

Symbolic programming is used heavily in ASSIST. Register equates are always used, and equate symbols are provided for most lengths. Very few nonsymbolic lengths are used anwhere in the code.

Since code modifcation is used in ASSIST to save time and space, the following rules are followed to make the code readable:

a. All instructions modified during execution contain either of the symbols $ or $CHN in the fields modified. If an opcode is modified, one of these symbols is added to the first operand field.

b. All modified instructions are labeled if they are modified by any instructions other than the ones immediately preceding or following the modified instruction.

2. TABLE MACROS

In order to facilitate changes in forms of large tables, almost all tables have entries generated macros instructions. This is especially useful for tables which often have new entries added (such as the PARM field table in csect APARMS, or the opcode table in csect OPCOD1).

One table macro which is used very heavily is the $AL2 macro, which generates a list of halfword offset values of labels from a given base label. Jumps are then taken by selecting one of the offset values, using codes which are multiples of two, then taking an indexed branch to the base label. This technique is used instead of the more common multiple-of-four codes and branch table method, because it is at worse only slightly slower, and uses only half the space.

3. NONOBVIOUS CODE

In general, the code is written in a reasonably straightforward fashion. However, in order to fulfill all the conflicting goals of ASSIST (high speed, small space, replacibility of assembler modules, linkage conventions close to OS/360 ones, and provision for future I/O simulators requiring large space), some sections of code are now optimized in a nonobvious way.

a. Some instructions are modified during execution. As mentioned above under SYMBOLIC CODING, all modified instructions use the symbols $ or $CHN to make this fact clear.

b. Most routines save and restore only those registers required, and some routines have every single register allocated at some points in their execution. The register allocation comments at the beginning of such routines describes this however, so that free registers can be found if needed.

c. Some routines use the knowledge of the order of variables to move values around with less instructions, particularly LM and STM. The order dependencies are always noted where the variables are defined.

d. In general, excessively optimized or tricky code is confined within a single control section, with its interface to remaining sections defined in a simple way.

APPENDIX II. SET VARIABLES AND CONDITIONAL ASSEMBLY

In order to to make possible the creation of ASSIST programs with differing sets of options at a minimum cost in memory, conditional assembly is heavily used throughout the ASSIST source program. The following lists the set variables which are used to control conditional assembly and includes their types and meanings of their possible values.

VARIABLE TYPE VALUES AND DESCRIPTIONS

&$ACCT GBLB =0 no accounting discrimination is possible
=1 accounting discrimination is possible.

This option is a future use option which can be used to include code to discriminate between different account numbers, thus allowing different classes differing program capabilities and options.
NOTE LOCALLY WRITTEN CODE MUST BE ADDED FOR THIS OPTION.

&$ALIGN GBLB =0 Model provides data alignment(360's)
=1 does not require alignment(360/85&370's)

&$ASMLVL GBLB =0 ASSIST is being assembled for a DOS system.
=1 ASSIST is being assembled for an OS system.

As of 08/20/71 (version 1.2/A1S1), the DOS modifications are being added, but are not yet available for distribution.

&$BLEN GBLA =# set to length of buffer(in bytes)

&$BUFNO GBLA =# set to number of buffers(>0)

&$BLEN and &$BUFNO are required for a disk utility version of ASSIST. Can never be set to 0. Must be present if &$DISKU > 0

&$CMPRS GBLB =0 the CMPRS listing option is not provided.
=1 code is provided for the CMPRS listing option, which allows printing two columns of statements per page, reducing the assembly listing by 50%.

If it =1, code is added to csects APARMS and OUTPUT.

&$COMNT GBLA =0 no COMNT option is available.
=# the COMNT option is available, and if coded, or if &$ACCT=1 and a given account number is used, will count the number of comments on machine instructions. User execution is not allowed if less than # per cent have comments on them.

If >0, code is added to APARMS, IAMOP1, and OUTPUT csects.

&$DATARD GBLB =0 only 1 card reader (SYSIN) exists.
=1 2 card readers exist, user program can read cards from different DDNAME than assembler source or object program.

This option adds code to XXXXIOCO.

&$DEBUG GBLB =0 debugging code is allowed to be generated in ASSIST modules.
=1 debugging code is not generated inside ASSIST.

This set variable interacts with other variables and macros to control generation of conditional debugging code. See set variables &DEBUG, &ID, and &TRACE, and macros $DBG and XSRTR. APPENDIX VI gives full details on ASSIST internal debugging aids.

&$DECK GBLB =0 no object decks can be produced.
=1 an object deck can be punched (entrypoint AODECK in AOBJDK exists).

Adds code in ASSIST, AOBJDK.

&$DECSA GBLB =0 assembler does not include code for decimal instruction set.
=1 assembler recognizes and assembles decimal instructions.

&$DECSM GBLB =0 the machine on which ASSIST is to run does not have the decimal instruction set option.
=1 the machine on which ASSIST is to run does have the decimal option.

As of 9/15/70, ASSIST requires the decimal instruction set. Some sections of code (particularly csects ASSIST and OUTPUT) use decimal instructions for convenience. This set variable is provided for future use in adding alternate code to let ASSIST run without this feature.

&$DISKU GBLA =0 no intermediate disk code generated.
=1 becomes user option, DISKU/NODISKU.
=2 always disk intermediate storage.

if >0, code is added to XXXXIOCO, and UTOPRS.

&$DSKDV GBLC =Device Number for DISKU option, only used when &$DISKU=0 and DOS system
default='2314'

&$ERNUM GBLA = value of highest-numbered error equate symbol, normally = 2 * number of different equate symbols.

This variable is used by macro $SERR to create equate values at first, and later to generate space for a pointer table in OUTPUT csect.

&$FLOTA GBLB =0 the ASSIST assembler does not include code and tables for the floating point instructions.
=1 the ASSIST assembler will recognize and assemble the floating point instructions.

&$FLOTAX GBLB =0 the ASSIST assembler does not include code and tables for the extended floating point instruction set.
=1 the ASSIST assembler will recognize and assemble extended floating point instructions.

&$FLOTE GBLB =0 ASSIST interpreter EXECUT will not execute floating point instructions.
=1 ASSIST interpreter will execute floating point instructions.

&$FLOTE is set to 0 if either &$FLOTA or &$FLOTM have that value, i.e., the interpreter will not contain code to perform floating point operations unless the assembler accepts the opcodes for them and the computer has the hardware to perform them.

&$FLOTEX GBLB =0 the ASSIST interpreter EXECUT will not execute extended floating point instructions.
=1 the ASSIST interpreter will execute extended floating point instructions.

&$FLOTEX is set to 0 if &$FLOTMX has that value, i.e. the interpreter will not contain code to perform extended floating point operations unless the machine has the hardware to perform them.

&$FLOTM GBLB =0 machine on which ASSIST is to run does not have floating point instructions set.
=1 machine does have floating point insructions.

The only part of the ASSIST assembler using floating point instructions is csect CDECNS (Floating point constant processor). This section could be rewritten using only fixed-point operations, depending on the value of &$FLOTM, for machines not having floating point, but with users wanting to use D and E constants.

&$FLOTMX GBLB =0 machine on which ASSIST is to run does not have extended floating point instruction set.
=1 machine does have extended floating point instructions.

&$FREE GBLA default value of FREE= parameter, should be set to the minimum value which releases enough space for operating system for buffers and modules to support whatever i/o devices are allowed.

&$GENDAT GBLC = date given version of ASSIST generated, in form mo/dy/yr. Becomes part of ASSIST header line.

&$HEXI GBLB =0 ASSIST interpreter EXECUT will not execute the XHEXI instruction.
=1 The XHEXI instruction will be executed.

&$HEXO GBLB =0 ASSIST interpreter EXECUT will not execute the XHEXO instruction.

&$IDF, GBLA = default and maximum possible values of I= option.
&$IMX If desired ASSIST has any kind of timer, these

values should be set very high so that the timing control variables are used to stop loops instead.

&$IOUNIT GBLC Specifies the ddname or file names used for standard I/O (not including XGET/XPUT).

#	USE	OS DEFAULT	DOS DEFAULT
1*	Primary Card Input	SYSIN(QSAM-GM)	SYSIPT(DTFCD)
2	Secondary Input	FT05f001(QSAM_GM)	SYSRDS(DTFCD)
3*	Printer	FT06F001(QSAM-PL)	SYSLST(DTFPR)
4	Punch	FT07F001(QSAM-PL)	SYSPCH(DTFCD)
5	Disk Intermediate	FT08F001(BSAM)	SYS001(DTFSD)
6	Macro Library	SYSLIB(BPAM)	N.A.
7	FOR FUTURE USE		
8	FOR FUTURE USE		

* Denotes Required Dataset (Access Method/DTF)

&$KP26 GBLB =0 only 029 keypunch decks can be accepted.
=1 ASSIST responds to the KP=26 option, and can translate incoming 026 keypunch decks correctly.

&$LDF,
&$LMX GBLA = default and maximum possible values of the L= parameter, only meaningful if &$PAGE=1. These are normally set to the maximum possible number of lines on an actual page (usually 63).

&$MACOPC GBLB =1 Open code conditional assembly allowed(if &$MACROS=1)
=0 No open code allowed

&$MACROS GBLB =0 the assembler part of ASSIST should not contain code for processing user-written macro instructions.
=1 the ASSIST assembler should contain code to process user-written macros.

&$MACROG,H,V GBLB NO asmg,h,v
FOR FUTURE USE

&$MACSIZ GBLA = address of byte 1 byte beyond maximum address of the computer. **ONLY REQUIRED IN NON-OS/360 systems since ASSIST obtains this information from OS/360 CVT.

&$MACSLB GBLB =1 Macro library search implemented
=0 No macro library search

&$MCHNE GBLC Type of machine(for header only)

&$MODEL GBLA = model number of S/360 for which the system is being assembled.

This variable is used to identify output, and is also provided for future use in generating code optimized for different models, and for modified versions for use with S/370 rather than S/360.

&$OBJIN GBLB =0 ASSIST cannot load object decks.
=1 object decks can be loaded.

This option adds code to ASSIST, and creates entry AOBJIN of csect AOBJDK.

```
&$OPTMS   GBLA      = value from 0 to 9 indicating the degreee of space
                      versus speed optimization in the code generated for
                      ASSIST.  Lower values generate smaller, but slower
                      versions of ASSIST.
```

This variable mainly affects sections of code in ICMOP2 and IDASM2. A small value causes the code in these sections to be generated to be general and small, rather than special case oriented and larger. If &$OPTMS is less than 3, error codes only are printed, saving about 1K of memory devoted to error messages in csect OUTPUT. &$OPTMS is also used to set the value of &$SYHASH (for symbol table). Also, if &$OPTMS<3, error messages are not printed, just the error numbers, and this change saves approximately 1K by itself.

&$PAGE GBLB =0 no page control code exists.
=1 page control code exists, and various options are added (CPAGE, L=, P=, PD=, PX=, SS, SSD, SSX).

This option adds code to ASSIST, XXXXIOCO.

&$PDF, &$PMX GBLA = default and maximum possible values of P= option (i.e., total number of pages allowed for 1 $JOB run in a BATCH or whole run otherwise). Only meaningful if &$PAGE=1

&$PDDF, &$PDMX GBLA = default and maximum possible values for the PD= option (number of pages saved for user completion dump). Only meaningful if &$PAGE=1.

&$PRIVOP GBLB =0 privileged operation codes are not accepted.
=1 privileged operation codes are assembled, and recognized by the interpreter EXECUT.

This variable affects the opcode tables in OPCOD1 and a few lines of code in ICMOP2 devoted to analysis of the operand forms needed only by privileged operations.It also affects code in the interpreter EXECUT.

&$PUNCH GBLB =0 no real card punch exists in ASSIST. Attempted punching will be simulated.
=1 real card punch exists.

This adds code to XXXXIOCO.

&$PXDF, &$PXMX GBLA = default and maximum possible values for the PX= option (number of pages allowed for user execution plus completion dump). Meaningful if &$PAGE=1.

&$P370 GBLB =0 the ASSIST interpreter EXECUT will not allow S/370 privileged operations.
=1 the ASSIST interpreter will recognize S/370 privileged operation codes.

This variable affects only code in the interpreter. Its value is set depending upon the value of &$PRIVOP and &$S370. If &$PRIVOP = 1 (i.e. the user wants privileged operations) and &$S370 ^= 0 (i.e. the user wants some type of S/370 interpretation) then &$P370 is set to 1. Else it is set to 0.

&$P370A GBLB =0 the ASSIST assembler will not permit S/370 privileged operation codes.
=1 the ASSIST assembler will recognize and assemble S/370 privileged operations.

This variable affects the opcode table in OPCOD1 and a few lines of code in ICMOP2 needed only for S/370 privileged operation codes. It is set to 1 if the values of &$PRIVOP and &$S370A are both 1 (i.e. the user wants privileged operation codes and also wants S/370 operation codes). Else it is set to 0.

&$RDF, GBLA = default and maximum values of R= option (total
&$RMX output lines and cards during 1 $JOB run).

&$RDDF, GBLA = default and maximum values of RD= option (number
&$RDMX of output records to be reserved to provide a user completion dump).

&$RECORD GBLA =0 or 1 record limit handling is done using the values of R=, RD=, and RX= options only.
=2 record limit processing may involve the use of the $TIRC macro (RECREM operand). This option should only be used if there is a way to obtain the actual number of records remaining for a job during execution. The $TIRC macro may have to be modified to accomplish this at a particular installation.

This variable affects only csect ASSIST.

&$RELOC GBLB =0 assembler will not contain code to relocate a program to actual location in memory.
=1 assembler will contain code to relocate user program to the area in memory where it actually is. The program can be run with store-only protection.

This option is required to =1 if &$REPL>0, since the Replace Monitor cannot handle programs unless they are relocated.
The only csect affected is UTOPRS. Code is added to test for the replace option being used, in which case flags are set to simulate the existence of a START card with the address in memory where the user program will be loaded, which permits relocation to be done with no other extra code in ASSIST.

&$REPL GBLB =0 ASSIST will not contain code for the replace process.
=1 ASSIST will recognize the REPL option, and will contain the code (csects REMONI, RFSYMS, and modifications to ASSIST csect) to perform dynamic replacement of control sections in the ASSIST assembler. See PART IV of this manual. This option allows module to be replaced as long as they need not call other modules.
=2 as for =1, except that the extra code is added to permit a replacement program to make calls to existing ASSIST assembler modules.

&$RXDF, GBLA = default and maximum values of RX= option (total
&$RXMX output records allowed for user execution plus dump together).

&$SPECIO GBLB =0 no special I/O operations are recognized.
=1 special I/O operations are recognized by the assembler (opcode type = $IS).

As of 9/15/70, this is a future use option for including code to simulate I/O operations in core, using QSAM type commands. At this time, the assembler part of this code does not exist, although some of the code for scanning JCL and building control tables does. See thesis paper of Harry McGuire, PSU.

&$SYHASH GBLA = number of fullwords in the initial pointer table used by symbol table handler SYMOPS csect, and allocated in the high end of the dynamic workarea.

This value varies from 8 to 64, and is set depending on the value of &$OPTMS, depending on space requirements. See beginning of csect SYMOPS for details on how this value is set.

&$SYSTEM GBLC = operating system being used. This value is noted on the printed output from ASSIST. It is also used to set the value of &$ASMLVL, depending on whether the first two characters of &$SYSTEM are 'OS' or not.

As of 9/15/70, this should have one of the values: OS-PCP, OS-MFT, or OS-MVT, or DOS.

&$S370 GBLA =0 the ASSIST interpreter EXECUT will not execute IBM S/370 instructions.
=1 the ASSIST interpreter will interpret S/370 instructions making free use of the S/370 instruction set.
=2 the ASSIST interpreter will interpret S/370 instructions using only standard S/360 instructions.

&$S370A GBLB =0 the ASSIST assembler will not include code for the S/370 instruction set.
=1 the ASSIST assembler will recognize and assemble standard S/370 instructions.

&$TDF, GBLC = default and maximum possible values which can
&$TMX be used as the T= option (i.e. total time for 1 $JOB .) Specified to millisec. if desired. Effective only if &$TIMER>0.

&$TDDF, GBLC = default and maximum possible values of TD=
&$TDMX option (time in seconds saved for execution dump). Can be specified to millisecond accuracy, and is meaningful only if &$TIMER>0.

&$TIMER GBLA =0 no timing is done at all.
=1 timing is done using only the IBM STIMER and TTIMER macros.
=2 timing is done using a local macro to find at execution time the remaining time left for a job.

This option affects code in APARMS, ASSIST, and EXECUT.
It also enables use of T=, TD=, and TX= option values.

&$TXDF, &$TXMX GBLC = default and maximum values of TX= option, (i.e. time for user program execution+dump together). Effective only if &$TIMER>0, and can be specified to millisecond accuracy if desired.

&$VERSLV GBLC = version and level of ASSIST being generated.

This value is printed by ASSIST to identify itself and aid in detection of errors...as of 2/1/73, this has the value '2.1/A'.

&$XIOS GBLB =0 the x-macro pseudo instructions are not assembled by the ASSIST assembler.
=1 the x-macro instructions are assembled and interpreted.

This value affects the following instructions: XDECI, XDECO, XDUMP, XHEXI, XHEXO, XLIMD, XPNCH, XPRNT, XREAD, which are all handled as instructions by ASSIST, and are macro instrcuctions under OS/360 assembler As of 9/15/70, this value should be 1, since there is no other way to perform input/output at this time. This value affects the opcode table in the csect OPCOD1, scanning code in ICMOP2, and execution code in EXECUT.

&$XXIOS GBLB =1 XGET/XPUT are not allowed
=0 XGET/XPUT allowed

&X$DDMOR GBLB =1 standard ddnames only
=0 allow user's own ddnames
only if &$XXIOS=0
certain names listed in XDDTABLE

&DEBUG GBLC = hexadecimal value used by XSRTR macro for flag testing. See macros $DBG and XSRTR, and APPENDIX VI for full explantion.

&ID GBLC = value to be used for identification at an entry point in a control section. Has value '*', in which case all entry points have identifications, or value 'NO', in which case none of them do. This is set depending on value of &$DEBUG. See macro $SAVE and APPENDIX VI for expanations.

&TRACE GBLC = control value for snaps generated by $SAVE and $RETURN macros at entry and exit points.

```
= NO   no trace code is generated at all
= *    a trace message is printed
= SNAP GP registers are printed, with message
```

See also Appendix VI for a full explanation of usage.

APPENDIX III. DUMMY SECTIONS AND TABLES   01/31/73 - 2.1/A

AJOBCON	03440100	2
AOBJCARD	05098030	2
APCBLK	04552200	2
AVWXTABL	02878020	2
CNCBLOCK	02721000	3
CONBLK	10366100	3
ECONTROL	03572100	3
ECSTACKD	03712200	3
ERCOMPCD	03420100	4
EVCTDSCT	11486100	4
ICBLOCK	02663100	4
IHADCB	07552100	4
LTBASETB	16288050	4
LTLENTRY	16310100	4
MACLIB	02876250	5
MCBOPRST	41335000	5
MCBSTRMS	41250000	5
MCBSU	41025000	5
MCGLBDCT	40785000	5
MCLCLDPV	40875000	5
MCOPQUAD	41160000	5
MCPARENT	40940000	5
MCPAROPR	41500000	5
MCPARSUB	41570000	5
MCSEQ	41095000	5
MSGBLOCK	02877300	5
MXPNTSAV	41385000	5
OPCODTB	02269100	5
OUCMPRSD	18748060	6
OUSTMTIM	18748900	6
RCODBLK	02776100	6
REBLK	02801000	6
RECORBLK	30004000	6
RFSYMBLK	30082000	7
RSBLOCK	02811500	7
RSCBLK	02848100	7
RSOURCE	02864100	7
SYMSECT	02689100	7
X$SLOTFO	02317244	7
XDECIB	07150600	7
XDECOB	07151080	8
XHEXIB	07151590	8
XHEXOB	07151845	8
XIOBLOCK	07540000	8
XSPIEBLK	08563175	8
XXSNAPC	07556200	8

```
**--> DSECT: AJOBCON    MAIN JOB CONTROL TABLE. . . . . . . . . . . . .
*.       THIS DSECT PROVIDES THE PRIMARY COMMUNICATION TABLE USED     .
*.       BY THE MAIN PROGRAM ASSIST, THE I/O ROUTINES(XXXXIOCO), THE  .
*.       PARM FIELD ANALYZER (APARMS), THE MAIN PROGRAM OF THE        .
*.       ASSEMBLER (MPCON0), AND THE REPLACE MONITOR (REMONI).  IT    .
*.       PROVIDES FOR GLOBAL FLAG VALUES DEALING WITH THE OVERALL     .
*.       JOB IN PROGRESS, PARM FIELD VALUES, USEFUL CONSTANTS, BLANKS,.
*.       ZEROES, WORKAREAS, AND DYNAMIC STORAGE AREA LIMITS.          .
*.       LOCATION: IN TABLE ASJOBCON OF CSECT ASSIST.                 .
*.       NAMES: AJ------                                              .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> DSECT: AOBJCARD   IMAGE OF OBJECT DECK CARD . . . . . . . . . . .
*.       THIS DSECT DESCRIBES 1 CARD OF AN ASSIST OBJECT DECK.  THE   .
*.  DECK FORMAT IS COMPATIBLE WITH NORMAL S/360 OBJECT DECKS, SO THAT .
*.  THEY CAN BE USED UNDER SOME CIRCUMSTANCES.    THEY ARE HOWEVER    .
*.  SIMPLER, IN ORDER TO ALLOW FOR PRODUCTION OF THEM FROM STUDENT-   .
*.  COMPILERS, I.E. XPL.  LATER VERSIONS OF THE LOADER MAY PERMIT     .
*.  MORE COMPLEX OBJECT DECKS, BUT AS OF 9/01/71, THE ONLY TYPES OF   .
*.  OBJECT DECK CARDS RECOGNIZED ARE TXT AND END CARDS.               .
*.       NAMES: AO------                                              .
*.       REFERENCE: ASSEMBLER(F) PROGRAMMER'S GUIDE - GC26-3756-4     .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> DSECT: APCBLK     APARMS PARM CODE BLOCK. . . . . . . . . . . . .
*.       THIS BLOCK DESCRIBES A PARM OPTION TABLE, GIVING THE NAME OF .
*.       THE PARM, A FLAG BYTE, AND AN OFFSET ADDRESS TO A PROCESSING .
*.       SECTION OF CODE IN CSECT APARMS.  IT IS USED ONLY IN APARMS. .
*.       LOCATION:  INSIDE TABLE APBPARMA IN CSECT APARMS.            .
*.       GENERATION: EACH APCBLK IS CREATED BY 1 CALL TO APCGN MACRO. .
*.       NAMES: APC-----                                              .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> DSECT: AVWXTABL   MAIN CONTROL TABLE FOR THE ASSEMBLER. . . . . .
*.       THIS DSECT IS USED BY ALMOST ALL SUBROUTINES OF THE ASSEMBLER.
*.       FOR COMMUNICATION, COMMON CONSTANTS, AND WORKAREAS, AND IS   .
*.       ALSO USED SOMEWHAT BY THE MAIN PROGRAM ASSIST AND THE        .
*.       REPLACE MONITOR REMONI.                                      .
*.       LOCATION: CSECT VWXTABL, WITH SAME NAMES PREFIXED WITH 'A'.  .
*.       NAMES: AX------,AW------,AV------  (DEPENDS ON SECTION)      .
*.       THIS DSECT CONTAINS THE FOLLOWING SECTIONS:                  .
*.                                                                    .
*.             1. ADDRESS CONSTANTS(NAMES: AX, FOLLOWED BY ENTRY NAME).
*.       THIS SECTION CONTAINS 1 ADDRESS CONSTANT FOR EVERY CALLABLE  .
*.       ENTRY POINT IN THE ASSIST ASSEMBLER.  THESE ARE READ-ONLY,   .
*.       EXCEPT DURING A REPLACE RUN, IN WHICH THE ADCONS FOR A       .
*.       SINGLE CSECT ARE TEMPORARILY MODIFIED.  THE LABEL AX$BASE IS .
*.       USED AS A BASE ADDRESS FOR THE CALCULATION OF OFFSETS TO     .
*.       INDIVIDUAL ADCONS, FOR THOSE ROUTINES REQUIRING TABLE-DRIVEN .
*.       CALLING SEQUENCES (CNDTL2,CODTL1,MPCON0,REMONI).  NOTE THAT  .
*.       ALL ENTRY POINTS HAVE 6-CHARACTER NAMES.  THE MACRO $CALL    .
*.       IS USED IN CONJUNCTION WITH THIS PART OF AVWXTABL.           .
*.                                                                    .
*.             2. CONSTANT VALUES (NAMES: AW------)                   .
*.       THIS SECTION CONTAINS USEFUL CONSTANT VALUES, SUCH AS        .
*.       ZEROES, BLANKS, MASK VALUES, TRANSLATE TABLES, EDIT PATTERNS..
*.       ALL VALUES ARE READ-ONLY, EXCEPT THAT ANY ROUTINE MAY        .
*.       MODIFY PART OF THIS SECTION IF IT RESTORES IT BEFORE        .
*.       ALLOWING ANOTHER SUBROUTINE TO GAIN CONTROL.  TRANSLATE     .
*.       TABLES INCLUDE ONES FOR SCANNING DECIMAL NUMBERS AND MACHINE .
*.       INPUT CONVERSION - HEX TO BINARY, SCANNING SYMBOLS AND      .
*.       INSTRUCTION OPERANDS, SCANNING HEXADECIMAL CONSTANTS, DOING  .
*.       GENERAL EXPRESSIONS, CONVERTING BINARY TO OUPUT HEXADECIMAL. .
*.       GENERATION: SECTION AWCONADS IS CREATED BY MACRO WCONG.     .
*.                                                                   .
*.             3. VARIABLES (NAMES: AV------)                        .
*.       THIS SECTION CONTAINS ALL VARIABLE AREAS USED FOR           .
*.       COMMUNICATION INSIDE THE ASSIST ASSEMBLER, IN ADDITION TO   .
*.       VARIOUS WORKAREAS, WHICH MAY BE OVERLAPPED TO SAVE SPACE.   .
*.       THE AREAS PROVIDED INCLUDE THE RECORD BLOCKS, LOCATION      .
*.       COUNTER VALUES, CURRENT SECTION ID, CURRENT DYNAMIC STORAGE  .
*.       AREA LIMITS, AND VARIOUS FLAGS.  TEMPORARY WORKAREAS ARE    .
*.       SUPPLIED, ALL WITH 'WORK' INCLUDED IN THEIR NAMES, WHICH    .
*.       CAN BE USED BY ANY ROUTINE , BUT ARE NOT SAFE ACROSS A      .
*.       SUBROUTINE CALL.  NOTE THAT THIS SECTION REQUIRES EQU SYMBOLS.
*.       FROM CNCBLOCK AND THE RECORD BLOCKS TO ASSEMBLE CORRECTLY.   .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> DSECT: CNCBLOCK   CONSTANT CODE BLOCK-DC'S, LITERALS. . . . . . .
*.       LOCATION: EACH CNCBLOCK IS CREATED IN AREA COBLK OF CODTL1.  .
*.       1 OR MORE CNCBLOCKS MAY BECOME PART OF THE RCODBLK CREATED   .
*.       IN AREA IBRCB BY IBASM1, AND 1 CNCBLOCK BECOMES PART OF THE  .
*.       ENTRY FOR EACH DISTINCT LITERAL(SEE LTLENTRY DSECT, LTOPRS   .
*.       CSECT.)                                                     .
*.       NAMES: CNC-----                                             .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> DSECT: CONBLK     CONSTANT DESCRIPTOR CODES BLOCK(CODTL1) . . . .
*.       THIS BLOCK CONTAINS DATA FOR A GIVEN CONSTANT TYPE, AND IS   .
*.       USED BY ASSEMBLER SUBR. CODTL1 IN SCANNING CONSTANTS AND     .
*.       BUILDING CNCBLOCKS DURING ASSEMBLY PASS 1.  THE DATA        .
*.       GIVEN INCLUDES A FLAG BYTE, DEFAULT LENGTH-1, LEFT AND      .
*.       RIGHT DELIMITER CHARACTERS REQUIRED FOR THE CONSTANT, AND    .
*.       MINIMUM AND MAXIMUM VALUES FOR THE LENGTH-1 OF THE CONSTANT. .
*.       THE FLAG BYTE, WITH MODIFICATIONS, BECOMES THE CNCTYPE BYTE  .
*.       OF THE CNCBLOCK CREATED FOR EACH CONSTANT OPERAND.          .
*.       LOCATION: TABLE CONTABL OF CSECT CODTL1                     .
*.       GENERATION: 1 CALL TO MACRO CONG CREATES A CONBLK ENTRY.    .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> DSECT: ECONTROL   EXECUTION CONTROL BLOCK . . . . . . . . . . . .
*.       THIS BLOCK CONTAINS ALL DATA REQUIRED TO DESCRIBE A USER    .
*.       PROGRAM TO BE EXECUTED BY THE ASSIST INTERPRETER (EXECUT).   .
*.       IT CONTAINS SIMULATED USER REGISTERS AND PROGRAM STATUS WORD,.
*.       AN INSTRUCTION STACK , POINTERS TO THE USER PROGRAM CODE,    .
*.       AND VARIOUS FLAGS DESCRIBING THE RUNNING MODE AND OPTIONS    .
*.       ALLOWED TO THE USER PROGRAM.  IT IS CREATED FROM INFORMATION .
*.       FROM THE ASSEMBLER, THE USER PARM FIELD, AND FROM THE       .
*.       OPTIONS IN ASSIST, AND IS MODIFIED BY EXECUT.  IT ALSO      .
*.       PROVIDES ALL DATA NEEDED BY XXXXSNAP TO DO A USER DUMP.     .
*.       LOCATION: IN HIGH END OF DYNAMIC CORE AREA.                 .
*.       NAMES: EC------                                             .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> DSECT: ECSTACKD   SINGLE ENTRY IN ECONTROL INSTRUCTION STACK. . .
*.       THE ECONTROL INSTRUCTION STACK IS A CIRCULAR LINKED LIST     .
*.       WHICH ALWAYS CONTAINS DATA ON UP TO THE LAST 10 INSTRUCTIONS .
*.       INTERPRETED DURING EXECUTION.  IT IS FILLED IN BY EXECUT, AND.
*.       IS USED BY XXXXSNAP TO PROVIDE THE INSTRUCTION TRACE PART    .
*.       OF A USER COMPLETION DUMP.                                   .
*.       LOCATION: INSIDE AREA ECINSTAC IN DSECT ECONTROL.            .
*.       NAMES: EC------     (SAME AS ECONTROL NAME CHARACTERS)       .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> DSECT: ERCOMPCD   COMPLETION CODE/ERROR MESSAGE BLOCK . . . . . .
*.       THIS GIVES FORMAT OF 1 COMPLETION CODE/MESSAGE BLOCK FOR     .
*.       USE IN A USER COMPLETION DUMP BY SUBROUTINE XXXXSNAP.  THE   .
*.       ADDRESS OF THE APPROPRIATE BLOCK IS PLACED INTO WORD ECERRAD .
*.       IN DSECT ECONTROL, AND IS USED THEN BY XXXXSNAP TO PRINT THE .
*.       INFORMATION IN THE ERCOMPCD BLOCK.                           .
*.       LOCATION: INSIDE EXECUT, WILL BE ELSEWHERE(FUTURE).          .
*.       GENERATION: 1 BLOCK CREATED BY 1 CALL TO $ERCGN MACRO.       .
*.       NAMES: ERC-----                                              .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> DSECT: EVCTDSCT   EVALUT TRANSITION TABLE ENTRY . . . . . . . . .
*.       THIS DESCRIBES 1 ENTRY IN 1 ROW OF THE GENERAL EXPRESSION    .
*.       EVALUATOR EVALUT, AND GIVES A SECTION OFFSET @ TO USE, AND   .
*.       EITHER A NEXT STATE(ROW) IN TABLE OR AN ERROR CODE FOR AN    .
*.       ILLEGAL CURRENT STATE/CURRENT VALUE COMBINATION.             .
*.       LOCATION: TABLE EVCTAB IN CSECT EVALUT.                      .
*.       GENERATION: 1 ROW OF EVCTDSCTS IS GENERATED BY 1 EVCG MACRO. .
*.       NAMES: EVCT----                                              .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> DSECT: ICBLOCK    MACHINE INSTRUCTION OBJECT CODE BLOCK.  . . . .
*.       THIS DSECT IS USED TO TRANSMIT DATA FROM ICMOP2 CSECT TO     .
*.       OUTPT2 FOR PRINTING MACHINE INSTRUCTIONS.                    .
*.       LOCATION: TABLE ICYBLOCK IN CSECT ICMOP2 OF ASSEMBLER.       .
*.       NAMES: ICB-----                                              .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> DSECT: IHADCB     DATA CONTROL BLOCK DSECT. . . . . . . . . . . .
*.       DCB DSECT USED BY PARTS OF XXXXIOCO.                         .
*.       GENERATION: DCBD MACRO                                       .
*.       LOCATION: XXSODCB,XXREDCB,XXPNDCB,XXPRDCB                    .
*.       NAMES: DCB-----                                              .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> DSECT: LTBASETB   LITERAL POOL BASE TABLE - 1 FOR EACH POOL . . .
*.       ONE LTBASETB IS CREATED FOR EACH LITERAL POOL, BY LTINT1 OR  .
*.       LTDMP1.  THE TOTAL # CREATED = # LTORGS + 2, WHICH INCLUDES  .
*.       1 FOR THE END STMT, AND 1 EXTRA 1 FOR CODE SIMPLIFICATION.   .
*.       WHEN LTDMP1 IS CALLED, IT FILLS IN THE SECTION ID OF THE     .
*.       SECTION WHERE THE POOL WILL BE ASSEMBLED, THE BEGINNING @ OF .
*.       THE POOL, AND THE OFFSET @ VALUES FROM THE BEGINNING @ TO    .
*.       EACH LITERAL IN THE POOL.  IN ADDITION TO ADDRESS AND SECTION.
*.       ID, THE LTBASETB ALSO CONTAINS THE LIST HEADS FOR 4 LISTS    .
*.       OF LITERAL ENTRIES (LTLENTRY BLOCKS). USED ONLY IN LTOPRS.   .
*.       LOCATION: HIGH END OF DYNAMIC AREA ($ALLOCH MACRO).          .
*.       NAMES: LTB-----                                              .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> DSECT: LTLENTRY   LITERAL TABLE ENTRY FOR EACH LITERAL. . . . . .
*.       1 LTLENTRY BLOCK IS CREATED BY LTENT1 FOR EACH UNIQUE        .
*.       LITERAL IN A GIVEN LITERAL POOL.  THE LTLENTRY BLOCKS ARE    .
*.       ORGANIZED IN 4 LINKED LISTS, WITH LIST HEADS IN THE CURRENT  .
*.       LTBASETB BLOCK.  EACH LTLENTRY INCLUDES THE OFFSET FROM THE  .
*.       BEGINNING OF THE CURRENT LITERAL POOL @ (ENTERED BY LTDMP1), .
*.       A COMPLETE CNCBLOCK DESCRIBING THE LITERAL CONSTANT, AND THE .
*.       CONSTANT IN CHARACTER FORM.  LTGET2 USES THESE BLOCKS TO     .
*.       DETERMINE THE USER PROGRAM ADDRESS FOR ANY DESIRED LITERAL,  .
*.       AND LTDMP2 USES THEM TO PRINT LITERAL POOL LISTING AND       .
*.       HAVE THE CODE ASSEMBLED FOR THE POOL.   USED ONLY IN LTOPRS. .
*.       LOCATION: HIGH END OF DYNAMIC AREA ($ALLOCH MACRO).          .
*.       NAMES: LTL-----                                              .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> DSECT: MACLIB    THIS DSECT GIVES THE FORMAT OF A MACRO         *
*.        LIBRARY ENTRY.                                              *
*.                                                                    *
*.*********************************************************************

*.--> DSECT: MCBOPRST      FORMAT OF OPERATOR STACK ENTRY             *
*.                                                                    *
*.* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

**--> DSECT: MCBSTRMS     FORMAT OF TWO BSU'S FOR EASE                *
*.        OF MANIPULATION IN TERM STACK                               *
*.                                                                    *
*.* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

**--> DSECT: MCBSU     FORMAT OF BASIC SYNTACTIC UNIT                 *
*.                                                                    *
*.* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

**--> DSECT: MCGLBDCT     FORMAT FOR GLOBAL DICTIONARY ENTRY          *
*.                                                                    *
*.*********************************************************************

**--> DSECT: MCLCLDPV    FORMAT FOR LOCAL DICTIONARY DOPE VECTOR      *
*.                                                                    *
*.*********************************************************************

**--> DSECT: MCOPQUAD    FORMAT OF ONE OP ENTRY.  MACRO DEFINITIONS   *
*.        ARE TRANSLATED INTO ONE OPS FOR SUBSEQUENT INTERPRETATION   *
*.                                                                    *
*.* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

**--> DSECT: MCPARENT   FORMAT FOR SYMBOLIC PARAMETER ENTRY           *
*.                                                                    *
*.*********************************************************************

**--> DSECT: MCPAROPR    FORMAT FOR SYMBOLIC PARAMETER DICTIONARY     *
*.        ENTRY.  ONE ENTRY FOR EACH SYM PARAM ON ENTRY TO MEXPND     *
*.                                                                    *
*.* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
**--> DSECT: MCPARSUB     FORMAT FOR DICT ENTRY FOR SUBLIST OPRNDS    *
*.        ONE ENTRY FOR EACH ELEMENT OF SUBLIST OF SYM PARAM ENTRY    *
*.                                                                    *
*.* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

**--> DSECT: MCSEQ     FORMAT OF SEQUENCE SYMBOL ENTRY                *
*.                                                                    *
*.* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

**--> DSECT: MSGBLOCK       ERROR MESSAGE BLOCK  * * * * * * * * * * *S

**--> DSECT: MXPNTSAV       CONTROL FOR LEVEL OF MACRO EXPANSION * *  S
*.       ONE IS ALLOCATED FOR EACH LEVEL OF MACRO CALL                A
*.       NAMES:MXP_____                                               A
*.* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

**--> DSECT: OPCODTB    DESCRIBES 1 ENTRY IN OPOCDE TABLE . . . . . . .
*.       LOCATION: ELEMENTS OF TABLE IN CSECT OPCOD1 OF ASSEMLBER.    .
*.       GENERATION: 1 CALL TO MACRO OPG CREATES AN ELEMENT.          .
*.       SECTIONS OPCTYPE,OPCHEX,OPCMASK CORRESPOND TO SIMILARLY-NAMED.
*.       SECTIONS OF DUMMY SECTION RCODBLK. SEE CSECT OPCOD1.         .
*.       NAMES: OPC-----                                              .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> DSECT: OUCMPRSD   CONTROL BLOCK FOR OUTPUT CMPRS OPTION . . . . .
*.       THIS BLOCK DESCRIBES AREA USED BY OUTPT2 WHEN DOING THE      .
*.       CMPRS LISTING OPTION (2 STMTS/LINE).  IT CONTAINS VARIABLES, .
*.       FLAGS, AND SPACE FOR $OU#NORM PARTIAL CARD IMAGES, WHICH     .
*.       ARE SAVED AND USED FOR THE LEFT-HAND-SIDE OF THE PAGE.       .
*.       THIS BLOCK IS ALLOCATED SPACE ONLY IF THE CMPRS PARM IS      .
*.       USED.  THE @ OUCMPRSD IS STORED IN OUCMPRAD VARIABLE.        .
*.       LOCATION: IN DYNAMIC AREA, ACQUIRED BY $ALLOCH IN OUINT1.    .
*.       NAMES: OUCM----                                              .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> DSECT: OUSTMTIM   STATEMENT IMAGE USED IN OUTPUT  . . . . . . . .
*.       USED IN CMPRS OPTION HANDLER OF OUTPT2 TO ACCESS PORTIONS    .
*.       OF INCOMING STATEMENTS TO BE SAVED.                          .
*.       NAMES: OUST----                                              .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> DSECT: RCODBLK    RECORD CODE BLOCK - VARIABLE DATA FOR STMT. . .
*.       AN RCODBLK IS CREATED BY EITHER IAMOP1 OR IBASM1 DURING      .
*.       ASSEMBLER PASS 1 FOR EVERY STATEMENT WITH AN ACCEPTABLE      .
*.       OPERATION CODE.  IT CONTAINS VARIABLE INFORMATION WHICH      .
*.       DEPENDS ON THE TYPE OF INSTRUCTION, AND MAY INCLUDE HEX      .
*.       MACHINE CODES AND MASKS, ALIGNMENT INFORMATION, LITERAL      .
*.       ADDRESSES, EQU SYMBOL ADDRESSES, AND 1 -10 CNCBLOCKS FOR DC  .
*.       COMMANDS.  THE MOST COMMON LENGTHS ARE 8 AND 12.             .
*.       LOCATION: CREATED IN AREA IARCB(IN IAMOP1) OR IBRCB(IN       .
*.       IBASM1). STORED IN LOW AREA AFTER ITS RSBLOCK BY UTPUT1.     .
*.       FOR MACHINE INSTRUCTIONS, MOVED TO ICRCB(IN ICMOP2) IN PASS 2.
*.       NAMES: RC------                                              .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> DSECT: REBLK      SCAN POINTER/ERROR CODE PAIR BLOCK. . . . . . .
*.       LOCATION: AVREBLK(AVWXTABL DSECT), CREATED BY ERRTAG SUBR.   .
*.       MOVED INTO LOW AREA FOLLOWING CORRESPONDING RCODBLK. MOVED   .
*.       BY UTGET2 BACK INTO AVREBLK AREA IN AVWXTABL DURING PASS 2.  .
*.       *NOTE* ONLY EXIST FOR STATEMENTS HAVING 1 OR MORE ERRORS OR  .
*.       WARNING MESSAGES ATTACHED TO IT.                             .
*.       NAMES: REB-----                                              .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> DSECT: RECORBLK   REPLACE MODULE-DESCRIBES 1 REAL-REPLACE PAIR. .
*.             THIS DSECT DESCRIBES 1 ENTRY IN THE TABLE RECORRAD.    .
*.       WHEN AN ENTRY POINT IS REPLACED, A RECORBLK IS CREATED FOR   .
*.       IT AND FILLED WITH VALUES FROM THE ENTRY POINT'S RFSYMBLK.   .
*.       THE ENTRY ADDRESS OF THE NEW ENTRY IS FOUND FROM THE SYMBOL  .
*.       TABLE (WHICH STILL EXISTS), AND IS SAVED INTO THE RECFPSW    .
*.       FIELD (OR A -1 PLACED HERE TO SHOW THE ENTRY COULD NOT BE    .
*.       FOUND IN THE USER PROGRAM).  USING THE RECAXAD FIELD, WHICH  .
*.       POINTS TO THE ADCON IN AVWXTABL OF THE REAL ROUTINE, THE REAL.
*.       ADCON IS SAVED IN RECADRE, AND IT IS REPLACED BY THE ADDRESS .
*.       OF REFAKE.  A CODE IS PLACED INTO THE HI-ORDER BYTE OF THE   .
*.       WORD IN AVWXTABL, WHICH IS USED BY REFAKE TO IDENTIFY WHICH  .
*.       ENTRY IS CALLED.                                             .
*.             AT THE END OF A REPLACE RUN, THE REAL ADCONS ARE MOVED .
*.       BACK TO THEIR PROPER PLACES IN AVWXTABL, USING THE RECAXAD   .
*.       FIELD OF EACH RECORBLK ELEMENT IN THE RECORRAD TABLE.        .
*.       **NOTE** FIRST SECTION OF DSECT SAME AS DSECT RFSYMBLK.      .
*.       LOCATION: CSECT REMONI, TABLE RECORRAD.                      .
*.       NAMES: REC-----                                              .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> DSECT: RFSYMBLK   REPLACE MODULE: 1 ENTRY IN TABLE CSECT RFSYMS .
*.             EACH SECTION OF RFSYMS GIVES EITHER A REPLACABLE       .
*.       CSECT NAME OR ONE OF ITS ENTRY POINT NAMES.   THE ENTRY      .
*.       POINT ELEMENTS CONTAIN VARIOUS POINTERS WHICH ARE USED TO    .
*.       GIVE OFFSET ADDRESSES FOR REAL ENTRY ADDRESS CONSTANTS OR    .
*.       FOR VARIOUS CHECKING CODE IN THE REPLACE MONITOR.            .
*.       **NOTE** THIS DSECT IS SAME AS FIRST PART OF RECORBLK DSECT. .
*.       GENERATION: 1 CALL TO RFSGN MACRO CREATS 1 CSECT ELEMENT     .
*.             AND 1 TO REC$MAX  ENTRY ELEMENTS.                      .
*.       LOCATION: CSECT RFSYMS.                                      .
*.       NAMES: RFS-----                                              .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> DSECT: RSBLOCK    RECORD SOURCE BLOCK-SOURCE CODE, FLAGS. . . . .
*.       AN RSBLOCK IS CREATED FOR EVERY SOURCE STATEMENT BY INCARD   .
*.       AND CONTAINS DATA COMMON TO EVERY STATEMENT, SUCH AS 1-3     .
*.       SOURCE CARD IMAGES, FLAGS FOR EXISTENCE OF OTHER RECORD      .
*.       BLOCKS.  ONLY RECORD BLOCK NECESSARY FOR A SOURCE STATEMENT. .
*.       LOCATION: CREATED IN AVRSBLOC (AVWXTABL DSECT) BY INCARD,    .
*.       WITH MODIFICATION BY ERRTAG AND MOCON1. MOVED TO LOW  END    .
*.       OF FREEAREA BY UTPUT1, AND REMAINS THERE.                    .
*.       NAMES: RSB-----                                              .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> DSECT: RSCBLK     RECORD SOURCE-CONTINUATIONS, SEQUENCE #'S . . .
*.       CREATED BY INCARD FOR ANY STATEMENT HAVING EITHER SEQUENCE   .
*.       NUMBERS OR CONTINUATION PUNCHES.                             .
*.       LOCATION: CREATED BY INCARD IN AVRSCBLK(AVWXTABL) DURING     .
*.       ASSEMBLY PASS 1. MOVED TO LOW END OF DYNAMIC AREA BY UTPUT1, .
*.       FOLLOWING CORRESPONDING REBLK(IF ONE EXISTS).  REMAINS IN    .
*.       THAT AREA FOR REST OF PROCESSING.                            .
*.       NAMES: RSC-----                                              .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> DSECT: RSOURCE    DESCRIPTION OF A SINGLE SOURCE CARD . . . . . .
*.       USED FOR INPUT PROCESSING BY SUBROUTINE INCARD.              .
*.       LOCATION: AVRSBLOC(AVWXTABL) DURING CREATION OF RSBLOCK.     .
*.       NAMES: RSO-----                                              .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> DSECT: SYMSECT    ASSEMBLER SYMBOL TABLE ENTRY. . . . . . . . . .
*.       CREATED BY ENTRY SYENT1 OF CSECT SYMOPS, AND HAS VALUES ADDED.
*.       BY MOCON1,IBASM1, FOR VALUE, SECTION ID, LENGTH ATTRIBUTE,   .
*.       AND BY ESDOPRS FOR SPECIAL ATTRIBUTES(CSECT,ETC).            .
*.       LOCATION:  FREEAREA HIGH END ($ALLOCH'D).                    .
*.       NAMES: SY------                                              .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> DSECT: X$SLOTFORMAT FOR XGET-XPUT MONITOR TABLE . . . . . . . . .
*        USED IN XDDGET AND XDDPUT TO CONTROL USE OF CERTAIN          .
*           DD NAMES BY USER WITH XGET-XPUT PERMITTED.                .
*                                                                     .
* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> DSECT: XDECIB     CONTROL BLOCK CREATED BY XDECI MACRO. . . . . .
*.       AN XDECIB IS CREATED BY EACH CALL TO THE XDECI MACRO, AND    .
*.       CONTAINS THE @ XXXXDECI, SAVEWORDS FOR REGS R14,R15,R0, AND  .
*.       WORDS FOR RETURN VALUES FOR REGISTER R1, AND THE ARGUMENT REG.
*.       THIS DSECT IS USED ONLY IN MODULE XXXXDECI.                  .
*.       GENERATION: XDECI                                            .
*.       NAMES: XDECI---                                              .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> DSECT: XDECOB     CONTROL BLOCK CREATED BY XDECO. . . . . . . . .
*.       AN XDECOB  IS CREATED FOR EACH XDECO CALL, AND CONTAINS THE  .
*.       @ XXXXDECO MODULE, SAVE WORDS FOR REGS R14,R15,R0, AND   A   .
*.       WORD FOR THE VALUE TO BE CONVERTED TO DECIMAL.               .
*.       XDECOB IS USED ONLY IN CSECT XXXXDECO.                       .
*.       GENERATION: XDECO                                            .
*.       NAMES: XDECO---                                              .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

*.--> DSECT: XHEXIB   CONTROL BLOCK CREATE BY XHEXI . . . . . . . . . .
*.       AN XHEXIB IS CREATED FOR XHEXI CALL, AND CONTAINS THE        .
*.@ XXXXHEXI MODULE, SAVE WORDS R14,R15, R0, AND A WORD VALUE THAT HAS.
*. BEEN CONVERTED                                                     .
*.       XHEXI IS USED ONLY IN CSECT XXXXHEXI                         .
*.       GENERATION XHEXI                                             .
*.       NAMES XHEXI___                                               .
*. . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
*.--> DSECT: XHEXOB   CONTROL BLOCK CREATED BY XHEXO. . . . . . . . . .
*.       AN XHEXOB IS CREATED FOR XHEXO CALL, AND CONTAINS THE @      .
*.  XXXXHEXO MODULE, SAVE WORDS FOR R14-R2 AND THE PLACE TO RETURN    .
*.       XHEXOB IS USED ONLY IN CSECT XXXXHEXO.                       .
*.       GENERATION: XXXXHEXO                                         .
*.       NAMES: XHEXO----                                             .
*. . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> DSECT: XIOBLOCK   CONTROL BLOCK FOR INPUT/OUTPUT MACROS . . . . .
*.       THIS BLOCK IS CREATED FOR ANY I/O MACRO BY THE INNER MACRO   .
*.       XIONR, AND CONTAINS THE ADCON FOR THE DESIRED I/O ENTRYPT,   .
*.       SAVE WORDS FOR MODFIED REGS R14,R15,R0, AND THE LENGTH FOR   .
*.       THE I/O AREA TO BE READ OR WRITTEN.                          .
*.       THIS DSECT IS ONLY USED IN CSECT XXXXIOCO.                   .
*.       GENERATION: BY MACRO XIONR (FOR $READ,$SORC,$PRNT,$PNCH).    .
*.       NAMES: XIO-----                                              .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> DSECT: XSPIEBLK    INTERRUPT COMMUNICATIONS CONTROL BLOCK . . . .
*.       THIS BLOCK CONTAINS EXIT ADDRESSES AND INTERRUPT MASKS FOR   .
*.       USE IN HANDLING THE 15 PROGRAM EXCEPTIONS.  THE INTERRUPT    .
*.       MASK IS EXTENDED TO A FULLWORD FOR EASE OF TESTING AGAINST   .
*.       THE INTERRUPTS THAT WERE DESIRED TO BE TRAPPED.  THE EXIT    .
*.       ADDRESS IS OF LENGTH 3 FOR CHANGING THE PSW(ONLY 3 BYTE @    .
*.       LOCATION:  INSIDE $SPIE MACRO EXPANSION                      .
*.       GENERATION:  ONE XSPIEBLK IS GENERATED FOR EVERY $SPIE       .
*.           EXPANSION EXCEPT LINKAGE TERMINATION & RESTORATION       .
*.       NAMES:  XSP-----                                             .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> DSECT: XXSNAPC    CONTROL BLOCK USED BY THE XSNAP MACRO . . . . .
*.       THIS BLOCK IS CREATED BY EVERY PRINTING XSNAP MACRO.  IT     .
*.       CONTAINS THE  EXACT CONTENTS OF THE GP REGISTERS BEFORE THE  .
*.       XSNAP WAS CALLED, A FLAG BYTE INDICATING DESIRED OUTPUT AND  .
*.       SPECIAL OPTIONS, THE NUMBER OF ADDRESS PAIRS USED IN THE     .
*.       XSNAP STORAGE= OPERAND, THE ADDRESS PAIRS THEMSELVES, AND    .
*.       THE ADDRESS CONSTANT FOR XXXXSNAP.  THE BYTE XXSFLAGS MAY    .
*.       HAVE SEVERAL BITS TURNED ON REQUESTING SPECIAL ASSIST        .
*.       SERVICES, SUCH AS USER DEBUGGING OUTPUT AND USER DUMP.  THE  .
*.       BITS ARE SUPPLIED BY XSNAP OPERAND T(3), AND HAVE            .
*.       MEANING ONLY WHEN USED INSIDE ASSIST WITH THE SPECIAL ASSIST .
*.       VERSION OF THE CSECT XXXXSNAP.                               .
*.       GENERATION: XSNAP MACRO, WITH T= ANY TYPE BUT ST OR STORE.   .
*.       NAMES: XXS-----                                              .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

&MALLOCH	01628240	3
&MALLOCL	01628040	3
$ALIGN	01433000	3
$ALIGR	01477000	3
$ALLOCH	01578000	3
$ALLOCL	01604000	3
$AL2	01325000	3
$CALL	01268400	3
$CKALN	01495000	3
$DALLOCH	01633000	4
$DBG	01305000	4
$DISK	01189500	4
$ERCGN	01196000	4
$GLOC	01511000	4
$GTAD	01778000	4
$LV	01789000	4
$MSG	01210530	4
$PNCH	01217500	4
$PRNT	01231000	4
$READ	01244500	4
$RETURN	01280400	5
$SAVE	01292400	5
$SCOF	01535000	5
$SCPT	01557000	5
$SDEF	01663000	5
$SERR	01680400	5
$SETRT	01717000	5
$SLOC	01523000	5
$SORC	01257000	5
$SPIE	01362230	5
$STV	01817000	6
$TIRC	02061000	6
APCGN	02102120	6
ASPAGE	02156200	7
ASPRNT	02120060	7
ASRECL	02156750	7
ASTIME	02125000	7
ASTIMR	02158300	7
CONG	01833000	7
EVCG	01853000	7
IBPRTAB	01880200	8
ICT	01915000	8
OPG	01929000	8
OPGT	01999000	8
REPRNT	02028080	8
RFSGN	02028340	8
WCONG	02035000	8
XCALL	02163000	9
XCHAR	00009000	9
XDECI	00034080	9
XDECO	00034640	9
XGET	01262056	9
XHEXI HE	00035080	9
XHEXO HE	00035360	9
XIDENT	00041500	10

XIONR	00077000	10
XLOOK	00145000	10
XMUSE	00185000	10
XPUT	01262090	10
XRETURN	00339000	10
XSAVE	00507000	10
XSNAP	00787000	10
XSRNR	01089000	11
XSRTR	01371000	11
XXDKEDCB	01189060	11

```
.*--> MACRO: &MALLOCH     GET CORE IN HIGH FREEAREA.  SAME AS &ALLOCH *
.*        EXCEPT USES AVGEN2CD AS HIGH END POINTER.  USED IN MEXPND   *
.*     &R  IS REG NEW USEABLE @ APPEARS IN                            *
.*     &L  GIVES REGISTER DESIRED LENGTH ISIN                         *
.*     &OVRFL  IS BRANCH @ IF OVERFLOW OCCURS                         *
.*                                                                    *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: &MALLOCL      GET CORE IN LOW FREEAREA.  SAME AS &ALLOCL *
.*        EXCEPT USES AVGEN2CD AS POINTER TO FREE HIGH AREA.  USED IN *
.*        MEXPND                                                      *
.*                                                                    *
.*     &R  GIVES REGISTER WHERE ADDRESS OF NEW USEABLE AREA APPEARS   *
.*     &L  GIVES REGISTER CONTAINING LENGTH DESIRED                   *
.*     &OVRFL IS @ TO BE BRANCHED TO IF OVERFLOW                      *
.*     &LENG IS THE LENGTH TO BE ALLOCATED                           *S
.*                                                                    *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: $ALIGN     GET,ALIGN, RESTORE UPDATED LOCATION COUNTER.  *
.*       USED TO ALIGN LOCATION COUNTER TO H, F, OR D BOUNDARIES.     *
.*       &R WILL CONTAIN ALIGNED VALUE OF LOCATION COUNTER            *
.*       &A GIVES ALIGNMENT REQUIRED , IF IN PARENTHESES, GIVES REG,  *
.*       IF NOT, GIVES DECIMAL NUMBER 1-3-7 FOR H,F,D ALIGN           *
.*       &TAG  IF CODED-MEANS THAT LOCATION COUNTER IS ALREADY IN &R. *
.*       USES MACROS: $ALIGR,$GLOC,$SLOC                              *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: $ALIGR     ALIGN VALUE IN REGISTER (USUALLY LOCCNTR).    *
.*        ALIGN REGISTER MACRO-ALIGN REGISTER &R TO BOUNDARY GIVEN    *
.*       BY VALUE IN REG &A, WHICH HAS 1,3,7 ETC IN IT.               *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: $ALLOCH    GET CORE IN FREEAREA HIGH END (ASSEMLBER).    *
.*       &R  IS REGISTER NEW USABLE ADDRESS APPEARS IN.               *
.*       &L  GIVES REGISTER LENGTH DESIRED IS IN.                     *
.*       &OVRFL IS ADDRESS TO BE BRANCHED TO IF OVERFLOW OCCURS.      *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: $ALLOCL    GET CORE IN LOW FREEAREA (IN ASSEMBLER).      *
.*       &R  GIVES REGISTER WHERE ADDRESS OF NEW USABLE AREA APPEARS  *
.*       &L    GIVES REGISTER CONTAINING THE LENGTH DESIRED.          *
.*       &OVRFL  IS ADDRESS TO BE BRANCHED TO IF OVERFLOW OCCURS.     *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: $AL2       CREATE HALFWORD ADDRESS OFFSET TABLE.         *
.*       USED TO GENERATE LIST OF AL2 ADDRESS CONSTANTS WHICH         *
.*       CONTAIN THE RELATIVE ADDRESS OF EACH ITEM IN &LIST FROM &BASE*
.*       &OFSET GIVES A NUMBER TO BE ADDED OR SUBTRACTED WHEN SETTING *
.*       UP THE EQU FOR THE LABEL,SO THAT INDEXING MAY START ANYWHERE *
.*       &L IS CODED IF THE OFFSET LIST SHOULD BE PRECEDED BY LENGTH  *
.*       SET UP FOR BXLE .                                            *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
.*--> MACRO: $CALL      SUBROUTINE CALL INSIDE ASSIST ASSEMBLER.      *
.*       &ENTRY    ENTRY POINT NAME TO BE CALLED, OS LINKAGE.         *
.*       **NOTE** GENERATES NAME WITH AX PREFIX, SO CAN ONLY BE USED  *
.*       INSIDE ASSEMBLER WHERE AVWXTABL USING HOLDS.                 *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: $CKALN     CHECK LOC-COUNTER ALIGNMENT, BRANCH IF SO.    *
.*       USED TO CHECK ALIGNMENT - &MASK IS 1-3-7, &B IS BRANCH LOC   *
.*       IF LOCATION COUNTER IS PROPERLY ALIGNED.                     *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: $DALLOCH   RETURN CORE-HIGH FREEAREA (IN ASSEMBLER)      *
.*       *NOTE* THIS IS A STACK POP TO BE USED WITH $ALLOCH (PUSH) FOR*
.*       FUTURE USE IN MACRO ASSEMBLER. AS OF 8/9/70 IT IS UNUSED.    *
.*       &R IS A WORK REGISTER, WHICH WILL BE DESTROYED               *
.*       &L REPRESENTS THE LENGTH. IF 1ST CHAR IS '(', WILL BE        *
.*       TAKEN AS REGISTER CONTAINING THE LENGTH, OTHER WISE TO       *
.*       BE AN ACTUAL LENGTH TO BE ADDED.                             *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: $DBG       SET TRACE, DEBUGGING SET VARIABLES FOR ASM.   *
.*       &D        HEX FLAG BYTE FOR USE IN TM INSTRUCTION.           *
.*       &T        IS TRACE MODE FOR AN XSNAP = NO,*,SNAP.            *
.*       SEE MACROS $RETURN,$SAVE,XSRTR FOR GENERATION OF TRACE CODE  *
*        ON ROUTINE ENTRY/EXIT. SEE ALSO ASSIST PROGRAM LOGIC MANUAL. *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: $DISK     CALL DISK UTILITY  * * * * * * * * * * * * * * *
.*       $DISK CALLS MACRO XIONR TO SET UP A BRANCH TO A DISK         *
.*       UTILITY ROUTINE.                                             *
.*       USES MACRO: XIONR                                            *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: $ERCGN     GENERATE COMPLETION CODE BLOCK FOR XXXXSNAP   *
.*       EACH CALL CREATES 1 ENTRY DESCRIBED BY DSECT ERCOMPCD.       *
.*       &CODE     CHARACTER VALUE OF ERROR CODE NUMBER.              *
.*       &MSSG     ERROR MESSAGE TO BE PRINTED                        *
.*       &TYPE     TYPE OF COMPLETION CODE - SYSTEM, ASSIST, OR USER. *
.*       *NOTE* IF &$OPTMS = 0, NO MESSAGE WILL BE GENED, ONLY CODE.  *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: $GLOC      GET LOCATION COUNTER INTO REGISTER.           *
.*       GET LOCATION COUNTER MACRO-PUTS LOCCNTR VALUE IN &RG         *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: $GTAD      LOAD ADCON INTO REGISTER FORM AVWXTABL.       *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: $LV        LOAD VARIABLE LENGTH VALUE INTO REGISTER(ASMB)*
.*       LOAD VARIABLE - PLACES &L BYTES IN &RG FROM &AD              *
.*       HIGH ORDER BYTES ARE ZEROED, USES AVFWORK1                   *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: $MSG                  USED TO GENERATE LINE IN MSG TABLE A
*        &NMBR IS MESSAGE #   (3 DIGITS)                              S
.*       &MSG IS QUOTED STRING OF MESSAGE                             A
.*       &FLAG IS FLAG BYTE                                           A
.* GENERATES:(LENGTH-1 OF MSG) U BYTE +3 FOR LENGTH OF MSG            A
.*       (FLAG BYTE) 1 BYTE                                           S
.*       CHAR FORM OF NMBR 3 BYTES                                    S

.*--> MACRO: $PNCH      PUNCH A CARD, BRANCH IF RECORD OVERFLOW       *
.*       &XAREA,&XNUM-SEE XIONR MACRO FOR EXPLANATION, OR XPNCH WRTUP *
.*       &OVER IS LABEL TO BE BRANCHED TO IF RECORDS EXCEED LIMIT.    *
.*       USES MACROS: XIONR                                           *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: $PRNT      PRINT A LINE, BRANCH IF RECORD OVERFLOWE.     *
.*       &XAREA,&XNUM-SEE XIONR MACRO FOR EXPLANATION, OR XPRNT WRITUP*
.*       &OVER IS LABEL TO BE BRANCHED TO IF RECORDS EXCEED LIMIT.    *
.*       USES MACROS: XIONR                                           *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: $READ      READ CARD DURING EXECUTION, BRANCH IF EOF.    *
.*       &XAREA,&XNUM-SEE XIONR MACRO FOR EXPLANATION, OR XREAD WRITUP*
.*       &EOF      LABEL TO BE BRANCHED TO IF END-FILE OCCURS.        *
.*       USES MACROS: XIONR                                           *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: $RETURN    RETURN FROM SUBROUTINE, OS LINKAGE.           *
.*       SUPPLIES EXTRA DEBUGGING CONTROL AND DEFAULTS TO XRETURN.    *
.*       USES MACROS: XRETURN                                         *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: $SAVE      SUBROUTINE ENTRY SETUP, OS LINKAGE.           *
.*       SUPPLIES EXTRA DEBUGGING CONTROL AND DEFAULTS TO XSAVE MACRO.*
.*       USES MACROS: XSAVE                                           *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: $SCOF      CONVERT REGISTER SCAN POINTER TO OFFSET VALUE.*
.*       SCAN POINTER OFFSET MACRO - PLACE SCAN POINTER REGISTER &SCP *
.*       INTO WORK REGISTER &RG, FIND OFFSET, AND STORE IT INTO &BYTE *
.*       IF &BYTE SPECIFIED.  &AD= WORD GIVING BEGINNING @ FOR OFFSET.*
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: $SCPT      CONVERT OFFSET TO A SCAN POINTER @ INTO REG.  *
.*       GET SCAN POINTER ADDRESS FROM OFFSET-OFFSET IS IN &BYTE,ADDR *
.*       IS CREATED IN &RG.  &AD GIVES BEGINNING @ OF FIELD.          *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: $SDEF      STORE VALUES IN SYMBOL TABLE ENTRY, FLAG DEFN.*
.*       &RVAL     REGISTER CONTAINING SYMBOL VALUE.                  *
.*       &RESD     REGISTER CONTAINING SECTION ID OF SYMBOL.          *
.*       &RLENG    REGISTER CONTAINING LENGTH ATTRIBUTE-1 FOR SYMBOL.**
.*       *NOTE* SYMSECT DSECT MUST HAVE VALID USING AT TIME OF CALL.  *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: $SERR      SET ERROR CODE MESSAGES AND EQU SYMBOLS.      *
.*       CALLED 2 TIMES FOR EACH ERROR EQU, 1 TIME TO SET UP EQU, 1   *
.*       TIME TO CREATE ERROR MESSAGE DC'S IN CSECT OUTPUT OF ASMBLER.*
.*       &ERR      IS LAST 5 CHARACTERS OF ERROR MESSAGE EQU SYMBOL.  *
.*       &MSG      IS THE ERROR MESSAGE ASSOCIATED WITH THE EQU.      *
.*       &NM       IS THE ERROR CODE FOR EXTERNAL USE - AS###.        *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
.*--> MACRO: $SETRT     SET UP TRT TABLE FOR SCANNING IN ASSEMLBER.    *
.*       USED INSIDE ASSIST ASSEMBLER TO CREATE TEMPORARY TRT TABLE IN*
.*       COMMON AREA AWTZTAB (WHICH CONTAINS 256 HEX 0'S).             *
.*       &LIST IS LIST OF CHARACTER/VALUE PAIRS, WITH CHARACTERS       *
.*       ENCLOSED IN QUOTES.  CORRESPONDONG VALUES ARE MOVED INTO      *
.*       CORRESPON:ING LOCATIONS IN 256-BYTE TABLE OF ZEROS.           *
.*       IF VALUE IS OMITTED, ZERO IS ASSUMED.                         *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: $SLOC      SET LOCATION COUNTER TO REGISTER VALUE.        *
.*       SET LOCATION COUNTER MACRO - SETS &RG AS LOCCNTR VALUE        *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: $SORC      READ ASSEMBLER SOURCE CARD, BRANCH IF EOF.     *
.*       &XAREA,&XNUM-SEE XIONR MACRO FOR EXPLANATION, OR XREAD WRITUP*
.*       &EOF      LABEL TO BE BRANCHED TO IF END-FILE OCCURS.         *
.*       USES MACROS: XIONR                                            *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: $SPIE        INTERRUPT COMMUNICATIONS    * * * * * * * * *
.*                                 SCOTT A. SMITH - FALL 1971.         *
.*       MAY BE USED BY OS OR DOS SYSTEMS TO SPECIFY THE ADDRESS       *
.*       OF AN INTERRUPTION EXIT ROUTINE AND TO SPECIFY THE PROGRAM    *
.*       INTERRUPT TYPES THAT ARE TO CAUSE THE EXIT ROUTINE TO BE      *
.*       GIVEN CONTROL.                                                *
.*       &EXIT     LABEL TO BE BRANCHED TO FOR THE INTERRUPTION        *
.*                 EXIT.  ADDRESS MAY BE IN A REGISTER.                *
.*       &TYPES    A LIST OF INTERRUPTION TYPES TO CATCH.  IF THIS     *
.*                 IS NOT SPECIFIED, A DEFAULT VALUE OF  ((1,15))      *
.*                 IS ASSUMED.  THE FORM OF THIS OPERAND IS A LIST     *
.*                 OF OPERANDS SEPARATED BY COMMAS.  THE LIST ITSELF   *
.*                 IS ENCLOSED IN PARENTHESES WITH EACH OPERAND        *
.*                 SPECIFYING A GROUP OF INTERRUPT TYPES TO CATCH.     *
.*                 EACH OF THESE IS EITHER A SINGLE INTEGER BETWEEN    *
.*                 1 AND 15, OR A PAIR OF INTEGERS BETWEEN 1 & 15      *
.*                 REPRESENTING AN INCLUSIVE RANGE OF INTERRUPTS.      *
.*                 EACH PAIR IS ENCLOSED IN PARENTHESES                *
.*       &ACTION=  SPECIFIES THE ACTION THIS MACRO IS TO TAKE.         *
.*             -->INIT: IDENTIFIES THIS AS AN INITIAL $SPIE CALL       *
.*                 AND INITIALIZATION IS TO BE PERFORMED.              *
.*             -->CR: CREATE A NEW $SPIE COMMUNICATION, BUT DO         *
.*                 NOT REINITIALIZE.                                   *
.*             -->(RS,(REG)) RESTORE A PREVIOUS $SPIE COMMUNICATION    *
.*                 LINK USING THE XSPIEBLK AT THE ADDRESS IN THE       *
.*                 REGISTER.  ALL OTHER PARAMETERS ARE IGNORED         *
.*            ***DEFAULT***INIT                                        *
.*       &CE=      THIS SPECIFIES AN OPTIONAL CALLABLE EXIT WHICH      *
.*                 MAY RECEIVE TEMPORARY CONTROL IMMEDIATELY FOLLOW-   *
.*                 ING AN INTERRUPT.  THIS EXIT MUST RETURN.           *
.*       *REGISTERS 14,15,0,1 ARE DESTROYED BY THIS MACRO*             *
.** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: $STV       STORE VARIABLE LENGTH VALUE FROM REGISTER (AS)*
.*       STORE VARIABLE MACRO-STORES &L BYTES FROM LOW ORDER END OF    *
.*       REGISTER &RG INTO ADDRESS &AD.                                *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
.*--> MACRO: $TIRC      GET TIME/RECORDS DATA FROM OPERATING SYSTEM.  *
.*             THIS MACRO USES PSU SVC CALL 250 TO OBTAIN TIME OR     *
.*       RECORDS INFORMATION.  &TYPE IS TIMREM,TIMUSE,RECREM,RECUSE.  *
.*       RESULT IS RETURNED IN R0, IN EITHER RECORDS, OR IN TIMER     *
.*       UNITS OF 26.04 MICROSECOND.  DESTROYS R0,R1,R15.             *
.*       *NOTE* MAY HAVE TO BE REWRITTEN FOR LOCAL CONDITONS.         *
.*       &TYPE CAN ALSO BE OF FORM  (NAME,ADDR)  WHERE ADDR IS AN     *
.*       RX-TYPE ADDRESS, AT WHICH THE MACRO PLACES THE FOLLOWING:    *
.*       BYTES  0-4  : ACCOUNT NUMBER       .... INFORMATION FROM     *
.*       BYTES  5-12 : JOB NAME             .... FROM                 *
.*       BYTES 13-32 : PROGRAMMER NAME      .... JOB CARD             *
.*       THIS FORM NEEDED ONLY IF &$ACCT=1,  AND IS COMPLETELY LOCAL  *
.*       TO PSU CC, THUS MUST BE REWRITTEN IF USED ELSEWHERE.         *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: APCGN      GENERATE 1 APCBLK ELEMENT IN APARMS . . . . . .
.*       GENERATES BLOCK FOR PARM OPTION SCANNING CONTROL, DEPENDING  .
.*       ON DESIRED CHARACTERISTICS OF THE PARM. MAY SKIP GENERATION  .
.*       IF THE REQUIRED OPTION DOES NOT EXIST IN PARTICULAR SYSTEM.  .
.*       ***SEE DSECT APCBLK AND CSECT APARMS (FROM LABEL APFOUND)    .
.*       FOR FURTHER INFORMATION ON HANDLING OF BLOCK CREATED BY THIS..
.*  &PARM       NAME OF THE PARM OPTION.                              .
.*  &AJOFS      NAME OF VARIABLE IN AJOBCON TO BE SET BY THIS PARM    .
.*  &BITS       VALUE USED TO SET FLAG FOR YES/NO TYPE PARMS.         .
.*       IF =PARM AND NOT CALL TYPE, SHOULD BE GIVEN VALUE 0.         .
.*  &G,&GC      USED TO CONTROL GENERATION.  GENERATION IS SKIPPED    .
.*       IF &G EQ &GC, THUS ALLOWING CONDITIONAL ASSEMBLY OF PARMS.   .
.*       &C THRU &Y GIVE TYPE BITS TO BE PLACED INTO APCFLAG.  EACH   .
.*       CORRESPONDS TO 1 OR MORE EQU SYMBOLS, AS LISTED.             .
.*  &C          =1 IF PARM IS NONSTANDARD AND A ROUTINE MUST BE CALLED..
.*       APPLIES ONLY TO =VALUE TYPE PARMS.  THE ROUTINE CALLED MUST  .
.*       BE NAMED APA&PARM.                           (APCCALL)       .
.*  &N          =1 IF VALUE CANNOT BE GIVEN ANOTHER VALUE ONCE IT HAS  .
.*       BEEN SET ONCE.  MAY BE USED BY ANY PARM TYPE.(APCNRSET)      .
.*  &D          =1 IF PARM IS PARM=DECIMAL VALUE.  IF THIS IS CODED   .
.*       AND PARM IS NOT A SPECIAL CALL TYPE, THEN IT IS ASSUMED THAT .
.*       THE VALUE CONVERTED IS TO BE STORED AS A FULLWORD AT THE     .
.*       GIVEN VARIABLE LOCATION IN AJOBCON.          (APCD)          .
.*  &I1         =1 IF PARM IS A YES/NO TYPE AND  1BIT ON CORRESPONDS   .
.*       TO A YES VALUE (1BIT MEANS NO OTHERWISE).    (APCYES1B)      .
.*              =1 IF PARM IS =DECIMAL # PARM, AND MAY NEVER BE       .
.*       INCREMENTED AFTER IT HAS BEEN SET (BUT MAY BE DECREASED).    .
.*       USED PARTICULARLY FOR TIME/RECORDS LIMITS.   (APCNINCR)      .
.*  &Y          =1 IF THE PARM IS A YES/NO TYPE.  OTHERWISE, IT IS    .
.*       AN =PARM OF SOME SORT.                       (APCYESNO)      .
.*  &LK         DENOTES WHICH OF THE POSSIBLE CALLS IS ALLOWED TO SET  .
.*       A VALUE FOR THE GIVEN PARM.  CONSISTS OF 3 BITS: ###, WITH   .
.*       MEANINGS AS FOLLOW:                                          .
.*       100   CAN BE SET BY LIMIT OR DEFAULT VALUE  (APCSETLD)       .
.*       010   CAN BE SET FROM THE PARM FIELD         (APCSETP)       .
.*       001   CAN BE SET BY USER FROM $JOB CARD      (APCSETU)       .
.*             THIS MACRO USED ONLY IN APARMS CSECT.                  .
.*  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
.*--> MACRO: ASPAGE     LINK TO SECTION OF PAGE CONTROL CODE  . . . . .
.*       &CODE IS TWO-DIGIT # GIVING DESIRED SECTION OF PAGE CONTROL  .
.*       CALL IS GENERATED ONLY IF &$PAGE = 1.                        .
.*  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

.*--> MACRO: ASPRNT     PRINT LINE INSIDE MAIN PROG ASSIST. * * * * * *
.*.      ASPRNT SETS UP R0=@ LINE, R1=LENG, CLLS INSUB ASASPRNT OF    *
.*       ASSIST.  MODIFIES REGS R0,R1,R14.                            *
.*.      &XAREA,&XNUM SAME AS THOSE FOR $PRNT = @, LENGTH TO PRINT.   *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: ASRECL     LINK TO RECORD LIMIT CONTROL CODE . . . . . . .
.*       &CODE IS TWO DIGIT NUMBER GIVING SECTION OF ASRECL## CALLED  .
.*  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

.*--> MACRO: ASTIME     UPDATE TIMER,PRINT TIMING MESSAGES(ASSIST).   *
.*       &ASH      NAME OF MESSAGE, IF OMIITED UPDATE TIMER ONLY.     *
.*       &VALUE    NAME OF VALUE TO BE CONVERTED, OMITTED-NO 2ND PART *
.*       *NOTE* ONLY USABLE INSIDE MAIN PROGRAM ASSIST.               *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: ASTIMR     LINK TO TIMER ROUTINES IN MAIN PROGRAM ASSIST *
.*       ASTIMR ALLOWS FOR CONDITIONAL GENERATION OF CALLS TO         *
.*       VARIOUS TIMING MODULES INSIDE ASSIST MAIN PROGRAM, DEPENDING *
.*       ON THE DESIRED TIMING METHOD BEING USED.                     *
.*       &CODE IS 2-DIGIT CODE, GIVING SECTION OF ASTIMR TO BE CALLED*
.*       &TLEVEL IS 0,1,2.  NO CODE IS CREATED IF &$TIMER<&TLEVEL.    *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: CONG       GENERATE CONSTANT CODE TABLE (CSECT CODTL1).  *
.*       USED IN CODTL1 OF ASSEMBLER TO PRODUCE 1 ENTRY IN            *
.*       CONSTANT DESCRIPTION BLOCK.  SEE CONBLK DSECT IN CODTL1.     *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: EVCG       CREATE ROW OF TRANSITION TABLE (CSECT EVALUT) *
.*       &L        LIST OF PAIRS- JUMP LABEL,(ERROR CODE OR STATE #). *
.*       CREATES 1 ROW OF TABLE EVCTAB IN GENERAL EXPRESSION EVALUATOR*
.*       CSECT EVALUT.  SEE EVCTDSCT DSECT FOR ENTRIES IN EACH ROW.   *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: IBPRTAB    GENERATE 1 BLOCK FOR PRINT SCAN LIST  . . . . .
.*.      USED ONLY IN IBASM1. CREATES 1 BLOCK: DSECT IBPSCECT         .
.*.      &OP       OPERAND NAME (ON, OFF, ETC).                       .
.*.      &VO       VALUE TO BE OR'D INTO PRINT BYTE: BIT TO SET ON.OFF.
.*.      &VX       VALUE TO BE XOR'D INTO PRINT CONTROL: EITHER 0     .
.*.                IF BIT ON (&VX OMITTED), OR SAME AS &VO IF * CODED..
.*. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

.*--> MACRO: ICT        CREATE CONTROL CODES(ICYFLAG) VALUES(ICMOP2). *
.*       &TYPE     TYPE OF INSTRUCTION FORMAT ($RR,$RX,ETC).          *
.*       &VALUE    VALUE OF CODE REQUIRED FOR TABLE.                  *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: OPG        CREATE 1 ENTRY IN ASM OPCODE TABLE (OPCOD1).  *
.*       THE GENERATED ENTRY IS DESCRIBED BY DSECT OPCODTB.           *
.*       GENERATES THE 4 FIELDS OF AN OPCODTB ENTRY - OPCTYPE,OPCHEX, *
.*       OPCMASK, AND OPCMNEM.  IF &HEX OR &MASK ARE OMITTED,THEY     *
.*       ARE ASSUMED TO BE 0.   &CODE IS USED FOR INSTRUCTIONS WHICH  *
.*       MAY NOT BE GENERATED. IF USED , IT IS 'D' FOR DECIMAL INSTS, *
.*       'F' FOR FLOATING POINT INSTRUCTIONS, AND 'P' FOR PRIVILEGED  *
.*       OPERATIONS.  IF THE SPECIFIED TYPE IS NOT TO BE GENERATED,   *
.*       THE APPROPRIATE GLOBAL VARIABLE WILL HAVE BEEN SET, AND THE  *
.*       OPCODTB ENTRY WILL NOT BE CREATED.                           *
.*       &CODE = 'M' FOR MACRO OPCODES.                               *
.*       &CODE = 'FX' FOR EXTENDED FLOATING POINT OPCODES.            *
.*       &CODE = 'S370' FOR NON-PRIVILEGED S/370 OPCODESS.            *
.*       &CODE = 'P370' FOR PRIVILEGED S/370 OPCODES.                 *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: OPGT       CREATE 2ND LEVEL OPCODE PTR TABLES (OPCOD1).  *
.*       USES MACROS: $AL2                                            *
.*       NOTE &OPNGN VALUES WERE SET BY OPG MACRO. CALLED 1 TIME ONLY.*
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: REPRNT     PRINT MESSAGE MACRO FOR REMONI USE  . . . . . .
.*.      &MSG  GIVES RX-TYPE ADDRESS OF MESSAGE TO BE PRINTED.        .
.*.      &MSGL GIVES LENGTH OF THE MESSAGE TO BE PRINTED.             .
.*.      MODIFIES REGISTERS  R7, R8,  R14.                            .
.*.      CALLS INSUB REXPRINT.                                        .
.*. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

.*--> MACRO: RFSGN      GENERATE 1 ENTRY OF REPLACE NAME TABLE(RFSYMS).
.*.            RFSGN MACRO IS USED TO GENERATE THE PRIMARY TABLE      .
.*.      OF CSECT NAMES AND THEIR ENTRY POINT NAMES, WHICH IS USED TO .
.*.      DO REPLACEMENT AND CHECKING OF STUDENT-WRITTEN CSECTS.       .
.*.            IF &$REPL=2 AND TYPE=2, RFSGN CREATES AN ELEMENT IN    .
.*.      THE SECOND SECTION OF RFSYMS, WHICH DESCRIBES A CALLABLE     .
.*.      ENTRYPOINT IN REAL ASSIST ROUTINES.                          .
.*.      &CSECT    NAMES A CSECT WHICH CAN BE REPLACED.               .
.*.            IF TYPE=2, NAMES A CALLABLE ENTRY FOR 2ND SECTION.     .
.*.      &ENTRY    IS A LIST OF 1 OR MORE ENTRY POINT NAMES IN &CSECT..
.*.            IF TYPE=2, THIS ONE IS OMITTED.                        .
.*.      &TYPE     = 1 IF &CSECT MAY CALL OTHER CSECTS, OMITTED IF NOT.
.*.            =2 IF CALL IS TO CREATE CALLABLE ENTRY ELEMENT.        .
.*. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

.*--> MACRO: WCONG      CREATE OFFSETS TO CONSTANT SUBR ADCONS-VWXTABL*
.*       CREATE WCONADS TABLE IN VWXTABL FOR USE OF CODTL1 AND CNDTL2 *
.*       IN DOING TABLE-DRIVEN CONSTANT PROCESSING. CALLED 1 TIME ONLY*
.*       &C        LIST OF CONSTANT TYPES ALLOWED.    (A,B,C, ETC).   *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: XCALL      SUBROUTINE CALL, OS LINKAGE, LITERAL FORM.    *
.*       &ENTRY    NAME OF ENTRYPOINT TO BE CALLED.                   *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: XCHAR      RETURN SAFE RIGHT-END SUBSTRING OF A STRING.  *
.*                                     JOHN R. MASHEY-JULY 1969-360/67*
.*       THIS MACRO RETURNS IN &XXCHAR THE &NUM CHARACTERS TAKEN FROM *
.*       THE RIGHT END OF THE CHARACTER STRING &STRING, WITHOUT       *
.*       BLOWING UP IF THERE ARE LESS THAN &NUM CHARS IN &STRING.     *
.*       THIS MACRO IS USED BY XSAVE,XRETURN, AND XSRNR               *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
.*--> MACRO: XDECI      EXTENDED DECIMAL INPUT CONVERSION * * * * * * *
.*             EXTENDED DECIMAL INPUT MACRO - ENABLES PROGRAMS        *
.*       WRITTEN FOR ASSIST TO BE RUN UNDER OS/360 DIRECTLY.          *
.*       USES MODULE XXXXDECI TO SCAN DECIMAL STRING BEGINNING AT     *
.*       &ADDRESS, CONVERT ITS VALUE INTO REGISTER &REG, AND SET      *
.*       REGISTER R1 AS A SCAN POINTER TO THE DELIMITER FOLLOWING THE *
.*       STRING OF DECIMAL DIGITS.  THE CONDITION CODE IS SET BY THE  *
.*       VALUE IN &REG, UNLESS AN ERROR OCCURRS, IN WHICH CASE CC=3.  *
.*       SEE ASSIST USER MANUAL FOR USAGE INSTRUCTIONS.               *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: XDECO      EXTENDED DECIMAL OUTPUT CONVERSION* * * * * * *
.*       USES MODULE XXXXDECO TO CONVERT VALUE IN REGISTER &REG TO    *
.*       AN EDITED 12-BYTE FIELD, WITH SIGN, AT LOCATION &ADDRESS.    *
.*             EXTENDED DECIMAL OUTPUT MACRO - ENABLES PROGRAMS       *
.*       WRITTEN FOR ASSIST TO BE RUN UNDER OS/360 DIRECTLY.          *
.*       SEE ASSIST USER MANUAL FOR USAGE INSTRUCTIONS.               *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: XGET    GET RECORD OFF OF &DDNAME FILE . . . . . . . . . *
.*                                 RICHARD FOWLER AUG, 1972 V.5.0     *
.*       MACRO FOR EASY READING OFF OF ANY DD FILE, READS &XNUM       *
.*       CHARACTERS. CONDITION CODE SET TO 0 NORMALLY, OR TO 1 ON     *
.*       END OF FILE. GENERATION CONTROLLED BY &XGETST.               *
.*       EXECUTION ASSUMES REG 1 POINTS TO DD NAME                    *
.* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * **

.*--> MACRO: XHEXI HEXADECIMAL INPUT CONVERSION MACRO.                *
.*                                WRITTEN BY ALAN ARTZ 4/17/72        *
.*       THIS MACRO TAKES THE VALUE STARTING AT THE ADDRESS GIVEN BY  *
.*  &ADDR AND CONVERTS IT AND PUTS THE HEXADECIMAL VALUE IN &REG.     *
.*  IF THERE ARE MORE THAN 8 DIGITS, R1 POINTS TO THE 9TH AND THE     *
.*  FIRST 8 ARE CONVERTED.  IF THERE IS A NON-BLANK, NON-HEX DIGIT    *
.*  FOUND, R1 POINTS TO THAT CHARACTER AND THE CC=3, OTHERWISE CC SET *
.*  BY VALUE IN REG.                                                  *
.*                                                                    *
.*       CALLS MODULE XXXXHEXI TO DO THE ACTUAL CONVERSIONS           *
.*********************************************************************

.*--> MACRO: XHEXO HEXADECIMAL OUTPUT CONVERSION MACRO                *
.*                                WRITTEN BY ALAN ARTZ 4/17/72        *
.*       THIS MACRO TAKES THE VALUE IN & REG AND CONVERTS IT TO       *
.*  PRINTABLE FORM.                                                   *
.*       IT PUTS THE CONVERTED VALUE IN AN EIGHT BYTE AREA STARTING AT*
.* THE ADDRESS GIVEN IN &ADDR.                                        *
.*      THE CONDITION CODE IS NOT CHANGED AND NETHER ARE THE REGISTERS*
.*                                                                    *
.*       CALLS MODULE XXXXHEXO TO DO THE ACTUAL CONVERSIONS.          *
.*********************************************************************

.*--> MACRO: XIDENT     IDENTIFY ENTRY POINT FOR XSAVE,$SAVE.         *
.*       MACRO USED BY XSAVE TO PRODUCE ID AT AN ENTRY POINT.  WILL   *
.*       USE THE FIRST NON-NULL OPERAND PASSED TO IT AS THE ID.       *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
.*--> MACRO: XIONR      INNER MACRO-$READ,$PNCH,$PRNT,$SORC           *
.*                                 ALSO XGET,XPUT,$GET,AND$PUT        *
.*                                 JOHN R. MASHEY - FEB 1970 - V.4.0  *
.*       XIONR IS USED BY XIOPAK MACROS XREAD,XPRNT,XPNCH TO SET UP   *
.*       THE REQUIRED CODE FOR CALLING THEIR RESPECTIVE SUBROUTINES.  *
.*       *** ARGUMENTS ***                                            *
.*       XNAME     THE NAME OF THE I/O ROUTINE TO BE CALLED.          *
.*       XNUM      THE LENGTH OF XAREA TO BE PRINTED,PUNCHED,ETC.     *
.*       XAREA     THE AREA ON WHICH I/O OPERATION TO BE PERFORMED.   *
.*             MAY BE SPECIFIED BY (0) OR (R0).                       *
.*       XDEFT     DEFAULT VALUE OF XNUM TO BE USED, IF IT IS OMITTED *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: XLOOK      FIND POSITION OF ELEMENT IN LIST.             *
.*                                 JOHN R. MASHEY - FEB 1970 - V.4.0  *
.*       MACRO TO FIND AND RETURN POSTION OF ARGUMENT IN A SUBLIST.   *
.*       &ARG1 ARGUMENT TO BE SEARCHED FOR                            *
.*       &ARGL LIST OF ARGUMENTS FOR &ARG1 TO BE CHECKED FOR IN       *
.*       &XXLOOK   THE FIRST POSITION IN &ARGL IN WHICH &ARG1 IS      *
.*       FOUND, IF ANY.  IF &ARG1 IS NOT IN &ARGL, &XXLOOK = 0.       *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: XMUSE      BASE REGISTER SETUP MACRO FOR XSAVE           *
.*                                 JOHN R. MASHEY - FEB 1970 - V.4.0  *
.*       THIS MACRO IS CALLED BY XSAVE TO HANDLE BR AND AD OPERANDS,  *
.*       AND PRODUCE APPROPRIATE USINGS.  &BR AND &AD ARE FROM XSAVE. *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: XPUT           PUT A RECORD ONTO FILE &DDNAME . . . . .  *
.*                                 RICHARD FOWLER AUG 1972 V.5.0      *
.*       MACRO FOR EASY PRINTING ONTO ANY DD FILE RECORD LENGTH=&XNUM *
.*       IF PRINT FILE, THE FIRST CHARACTER IS USED AS CARRIAGE CONTROL
.*       GENERATION CONTROLLED BY &XPUST                              *
.*       EXECUTION ASSUMES REG 1 POINTS TO DD NAME                    *
.* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * **

.*--> MACRO: XRETURN    GENERAL RETURN MACRO, OS LINKAGE              *
.*                                 JOHN R. MASHEY - FEB 1970 - V.4.0  *
.*       EXTENDED RETURN MACRO - SEE PSU CC WRITEUP - XSAVE/XRETURN   *
.*       FOR EXPLANATION AND USE OF OPERANDS.                         *
.*       USES MACROS: FREEMAIN,XCHAR,XSRNR                            *
.** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: XSAVE      EXTENDED SAVE MACRO - OS LINKAGE.             *
.*                                 JOHN R. MASHEY - FEB 1970 - V.4.0  *
.*       EXTENDED SAVE MACRO - SEE PSU CC WRITEUP - XSAVE/XRETURN     *
.*       FOR DESCRIPTION OF ARGUMENTS FOR THIS MACRO                  *
.*       USES MACROS: GETMAIN,XCHAR,XIDENT,XLOOK,XMUSE,XSRNT,XSRTR    *
.** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: XSNAP      EXTENDED SNAP MACRO-DEBUGGING-DUMPING.        *
.*                                 JOHN R. MASHEY - FEB 1970 - V.4.0  *
.*       XSNAP     IS USED FOR STORING,PRINTING OF REGISTERS AND ANY  *
.*       OTHER ADDRESSIBLE AREAS. XSNAP HARMS NO REGISTERS,CAN BE USED*
.*       IN ANY NUMBER OF CSECTS IN 1 ASSEMBLY,AND PRINTS REGISTERS   *
.*       EXACTLY AS THEY ARE WHEN THE XSNAP IS CALLED.  XSNAP         *
.*       ACTION MAY BE MADE CONDITIONAL EITHER AT ASSEMBLY TIME OR    *
.*       DURING EXECUTE TIME.  SEE WRITEUP FOR OPERAND DESCRIPTION.  *
.*       USES MACROS: XLOOK                                           *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: XSRNR      SAVE/RESTORE REGISTERS FOR XSAVE/XRETURN      *
.*                                 JOHN R. MASHEY- FEB 1970 - V.4.0   *
.*             THIS MACRO IS USED BY XSAVE AND XRETURN TO SET UP      *
.*       REGISTER SAVING AND RESTORATION.                             *
.*       &OP IS THE OPCODE TO BE USED.  I.E.  EITHER L  OR ST.        *
.*       &RG  IS 1 OPERAND FROM THE &RGS OPERAND USED BY XSAVE AND    *
.*             XRETURN.  IT IS EITHER 1 REGISTER, OR A PAIR OF REGS   *
.*             SEPARATED BY A DASH.                                   *
.*       &NO15   =0  STATES THAT A RETURN CODE IS CURRENTLY IN REG 15 *
.*             AND SHOULD NOT BE DISTURBED, REGARDLESS OF HOW THE REGS*
.*             ARE SPECIFIED.                                         *
.*       USES MACROS: XCHAR                                           *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: XSRTR      CREATE SPECIAL ASSIST ENTRY/EXIT TRACE CODE.  *
.*                                     JOHN R. MASHEY-JULY 1969-360/67*
.*       THIS MACRO IS USED BY XSAVE AND XRETURN TO GENERATE THE      *
.*       TRACE CODE CALLS TO XPRNT OR XSNAP, IF THE TR OPERAND IS USED*
.*       *NOTE* THIS IS MODIFIED VERSION FOR USE IN ASSIST ONLY.      *
.*       USES MACROS: XSNAP                                           *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

.*--> MACRO: XXDKEDCB GENERATE TABLE OF DECBS FOR DISK UTILITY  * * * *
.*       THIS MACRO GENERATES A LINKED TABLE OF DECBS.                *
.*       THE BUFFER ADDRESSES ARE PLACED IN THE DECB BY XXXXDKOP      *
.*       USES MACRO: WRITE                                            *
.*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

APPENDIX V. ENTRY AND EXIT CONDITIONS 01/31/73 - 2.1/A

THE ENTRY POINTS ARE LISTED ALPHABETICALLY BY CSECT NAME.

AOBJDK	05098480	3
APARMS	04577050	5
ASSIST	03751000	6
BROPS2	08566000	7
CACONS	08738000	8
CBCONS	08846000	9
CCCONS	08992000	10
CDECNS	09156000	11
CFHCNS	09406000	12
CNDTL2	09584000	13
CODTL1	09886000	14
CPCONS	10388000	15
CVCONS	10586000	16
CXCONS	10712000	17
CZCONS	10858000	18
ERRORS	11044000	19
ESDOPRS	11138000	20
EVALUT	11520000	21
EXECUT	05102440	22
IAMOP1	12316000	23
IBASM1	12466000	24
ICMOP2	13414000	25
IDASM2	14844000	26
INPUT1	15418000	27
LTOPRS	15678000	28
MACFND	46210000	30
MACINT	41655000	31
MACLEX	52520000	32
MACRO1	42025000	33
MACSCN	43945000	34
MCATRM	47620000	35
MCBODY	47910000	36
MCDTRM	47020000	37
MCGNCD	55805000	38
MCGTST	47255000	39
MCSCOP	45330000	40
MCSYSR	46655000	41
MCVSCN	46405000	42
MEXPND	57695000	43
MOCON1	16346000	44
MPCON0	16714000	45
MTCON2	16962000	46
MXERRM	66130000	47
MXINST	60535100	48
MXMVSR	66690000	49
OPCOD1	17070000	50
OUTPUT	17754000	51
REMONI	30144000	52
RFSYSMS	32790000	54
SCANRS	18752000	55
SDTERM	18880100	56
SYMOPS	19196000	57
UTOPRS	19492000	58
VWXTABL	20008100	60

```
**--> CSECT: AOBJDK     OBJECT DECK HANDLING MODULE . . . . . . . . . .
*.                                 JOHN R. MASHEY - 09/01/71          .
*.       THE TWO ENTRIES OF AOBJDK ARE USED TO LOAD OR PUNCH OBJECT   .
*.  DECKS WHICH ARE SUBSETS OF NORMAL S/360 DECKS.   THE TWO ENTRIES  .
*.  MAY OR MAY NOT EXIST, DEPENDING ON FLAGS &$DECK AND &$OBJIN.      .
*.       USES DSECTS: AOBJCARD,AVWXTABL                               .
*.       USES MACROS: $RETURN,$SAVE                                   .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> INSUB: AOBDUMP    DUMP CURRENT USER CARDIMAGE + + + + + + + + + +

**--> INSUB: AOBHEXCO   CONVERT VALUES TO EDITED HEXADECIMAL  + + + + +

**--> ENTRY: AOBJIN     LOAD OBJECT DECK  . . . . . . . . . . . . . . .
*.       ENTRY CONDITIONS                                             .
*.  R12(RAT) = @ ASSEMBLER CONTROL TABLE (AVWXTABL).                  .
*.       EXIT CONDITIONS                                              .
*.  AVRADL,AVRADH,AVRELOC,AVFENTER,AVLOCLOW,AVLOCHIH  ARE SET UP      .
*.  AS THEY WOULD HAVE BEEN HAD THE PROGRAM BEEN ASSEMBLED.           .
*.  AVTAGS1    IS FLAGGED WITH AJNLOAD IF SOME ERROR OCCURRED.        .
*.       NAMES: AOB-----                                              .
*.       USES MACROS: $PRNT,$RETURN,$SAVE,$SORC,XSNAP                 .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> INSUB: AOBPRINT   PRINT 1 LINE OF OUTPUT MESSAGE  + + + + + + + +

**--> ENTRY: AODECK     PUNCH OBJECT DECK FOLLOWING ASSEMBLY  . . . . .
*.       AODECK IS CALLED FOLLOWING A SUCCESSFUL ASS
*.       IF THE DECK OPTION IS SPECIFIED, AODECK IS CALLED FOLLOWING  .
*.  A SUCCESSFULL ASSEMBLY TO PUNCH THE USER PROGRAM OUT IN OBJECT    .
*.  DECK FORM.  THE DECK PUNCHED CONTAINS 1 OR MORE TXT CARDS AND     .
*.  1 END CARD, AND FOLLOWS S/360 DECK FORMAT FAIRLY CLOSELY.         .
*.       **NOTE** THIS FACILITY IS VERY PRIMITIVE, AND THE DECKS      .
*.  PRODUCED CANNOT REALLY BE USED FOR ANYTHING BUT INPUT TO ASSIST,  .
*.  SINCE THERE IS NEITHER EXTERNAL SYMBOL DICTIONARY NOR RELOCATION  .
*.  DICTIONARY PRODUC D.  ALSO, SINCE THE ENTIRE USER PROGRAM IS      .
*.  PUNCHED, OBJECT CARDS ARE PRODUCED FOR SPACE CONTAINING ONLY DS   .
*.  LOCATIONS.  IN SOME CASES, THIS COULD CAUSE HUGE DECKS TO BE      .
*.  PUNCHED.   IF A BETTER SETUP IS DESIRED, ASSEMBLER MODULE UTOPRS  .
*.  COULD BE CHANGED TO PRODUCE SMALLER DECKS, ALTHOUGH RLD ENTRIES   .
*.  WOULD STILL BE DIFFICULT TO PRODUCE.                              .
*.       **NOTE** THE MOST LIKELY USE FOR THIS OPTION IS TO PRODUCE   .
*.  OBJECT DECKS TO BE USED AS UTILITY PROGRAMS FROM RJE TERMINALS.   .
*.       ENTRY CONDITIONS                                             .
*.  R12(RAT) = @ ASSEMBLER CONTROL TABLE (AVWXTABL).                  .
*.       USES MACROS: $PNCH,$RETURN,$SAVE                             .
*.       NAMES: AOD-----                                              .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
*        SPACE 1
* * * * * * * * REGISTER USAGE FOR AODECK * * * * * * * * * * * * * * *
*   R4 = @ CURRENT BLOCK OF CODE TO BE PUNCHED (INIT = AVRADL).       *
*   R5 = CURRENT LENGTH OF CODE REMAINING (INIT =AVRADH-AVRADL)       *
*   R6 = BASE REGISTER                                                *
*   R7 = @ AOBJCARD : OBJECT CARD OUTPUT IMAGE                        *
*   R8 = CURRENT @ OF CODE TO PUNCHED (USER PROGRAM RELATIVE).        *
*   R9 = L'AOTCODE = LENGTH OF NORMAL(ALL BUT LAST) CODE ON CARD      *
*   R12(RAT)= @ ASSEMBLER CONTROL TABLE (AVWXTABL).                   *
*   R13= @ CALLING PROGRAM'S SAVE AREA, UNCHANGED                     *
*   R14= INTERNAL LINK REGISTER                                       *
*   ALL OTHERS ARE UNUSED                                             *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

**--> INSUB: AODPUNCH   PUNCH 1 OBJECT CARD FOR AODECK  + + + + + + + +
**--> CSECT: APARMS     USER PARM FIELD PROCESSING CSECT. . . . . . . .
*.       SCANS USER PARM FIELD, SETS VALUES IN AJOBCON DSECT.         .
*.       ENTRY CONDITIONS                                             .
*.  R9 = @ OF ACTUAL PARM FIELD CHARACTER STRING.                     .
*.  R10= LENGTH OF PARM FIELD AT 0(R9).                               .
*.  R11= ADDRESS OF AJOBCON DUMMY SECTION AREA.                       .
*.       EXIT CONDITIONS                                              .
*.  AJOPARM IN AJOBCON NOW HAS USER PARM FIELD, RIGHT-PADDED WITH ' '..
*.  VARIOUS FLAGS IN AJOBCON ARE NOW SET(SEE CODE STARTING AT APAJUMP).
*.       USES DSECTS: AJOBCON,APCBLK                                  .
*.       USES MACROS: $DBG,$RETURN,$SAVE,$TIRC,APCGN,XDECI            .
*.       *NOTE* AS OF 8/12/70, THIS PROGRAM IS MORE GENERAL THAN      .
*.       CURRENTLY NEEDED, TO ALLOW FOR FUTURE NEW PARM OPTIONS.      .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> INSUB: APDECON    CONVERT DECIMAL PARM VALUE  + + + + + + + + + +
**--> CSECT: ASSIST     MONITOR CONTROL PROGRAM FOR THE ASSIST SYSTEM .
*.       ENTRY CONDITIONS                                             .
*.  R1= @ POINTER TO OS LENGTH/PARM FIELD AREA.                       .
*.       CALLS AOBJIN,AODECK,APARMS,EXECUT,MPCON0,REENDA,REINTA       .
*.       CALLS XXXXFINI,XXXXINIT                                      .
*.       USES DSECTS: AJOBCON,AVWXTABL,ECONTROL                       .
*.       USES MACROS: $DBG,$PRNT,$RETURN,$SAVE,$SORC,$TIRC            .
*.       USES MACROS: ASPAGE,ASPRNT,ASRECL,ASTIME,ASTIMR              .
*.       USES MACROS: FREEMAIN,GETMAIN,STIMER,TTIMER,XCALL,XSNAP,WTL  .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> INSUB: ASASPRNT   CALLED BY ASPRNT MACRO TO PRINT A LINE. + + + +

**--> INSUB: ASFLUSH    FLUSH CARD RDR UNTIL NEXT COMMAND CARD  + + + +

**--> INSUB: ASMSFINI FREE CURRENT DYNAMIC STORAGE AREA + + + + + + + +

**--> INSUB: ASMSINIT   MAIN STORAGE INITIALIZATION + + + + + + + + + +

**--> INSUB: ASPAGE##   PAGE CONTROL CODE FOR PAGE MODE LIMITS  + + + +

**--> INSUB: ASRECL##   RECORD LIMIT CONTROL  + + + + + + + + + + + + +

**--> INSUB: ASTIMER    UPDATE TIMER,PRINT ELAPSED TIME,MESSAGE + + + +

*+--> INSUB: ASTIMR##   TIMING SERVICES IN ASSIST MAIN PROGRAM. + + + +

**--> INSUB: ASTIMSET   SET INTERVAL TIMER ROUTINE  + + + + + + + + + +

**--> INSUB: ASTRP16    COMPUTE VALUES FOR BEFORE EXECUTION + + + + + +
**--> CSECT: BROPS2   2   ALL BASE REGISTER OPERATIONS - ALL PASS 2 . .
*.       USES DSECTS: AVWXTABL                                          .
*.       USES MACROS: $RETURN,$SAVE                                     .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: BRDISP   2   GIVEN VALUE&ESDID, RETURN BASE-DISPLACEMENT .
*.       ENTRY CONDITIONS                                               .
*.  RA = ADDRESS VALUE TO BE DECOMPOSED TO BASE-DISPLACEMENT (24 BITS).
*.  RB = ESDID OF ADDRESS TO BE DECOMPOSED - LOW ORDER BYTE             .
*.       VALUE IS FROM 1-255. 0 CAN BE USED TO MARK NONUSABLE.          .
*.       EXIT CONDITIONS                                                .
*.  RA = BASE-DISPLACEMENT FORM OF ADDRESS, IF ADDRESSABLE              .
*.  RB = 0    NORMAL RETURN - ADDRESS WAS DECOMPOSABLE                  .
*.     = ^0   ADDRESSIBILITY ERROR(NO REG,OR DISP TOO LARGE)            .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: BRDROP   2   DROP A REGISTER FROM USING. . . . . . . . . .
*.       ENTRY CONDITIONS                                               .
*.  RA = NUMBER OF REGISTER TO BE DROPPED FROM USING - = 0-15           .
*.       EXIT CONDITIONS                                                .
*.  RB = 0    THE REGISTER WAS CURRENTLY USABLE                         .
*.  RB = ^0   THE REGISTER WAS NOT CURRENTLY IN USE                     .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: BRINIT   2   INITIALIZE BASE REGISTER TABLES . . . . . . .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: BRUSIN   2   ENTER A REGISTER-VALUE PAIR . . . . . . . . .
*.       ENTRY CONDITIONS                                               .
*.  RA = NUMBER OF REGISTER FOR WHICH USING TO BE SET UP = 0-15         .
*.  RB = ADDRESS DECLARED IN USING FOR GIVEN REGISTER = 0-2**24-1       .
*.  RC = ESDID OF THE USING VALUE, IN LOW ORDER BYTE = 1-255            .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> CSECT: CACONS   1-2 PROCESS A-TYPE ADDRESS CONSTANTS. . . . . . .
*.       USES DSECTS: AVWXTABL                                         .
*.       USES MACROS: $CALL,$RETURN,$SAVE                              .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: CACON1     SCAN ACON, BUT DO NOT ASSEMBLE VALUE. . . . . .
*.       ENTRY CONDITIONS                                              .
*.  RA = SCAN POINTER (ADDRESS OF 1ST CHAR AFTER PREVIOUS DELIMETER)  .
*.       EXIT CONDITIONS                                               .
*.  RA = SCAN POINTER (ADDRESS OF DELIMITER STOPPING SCAN, OR ERROR)  .
*.  RB = 0    CONSTANT WAS LEGAL, NO ERRORS                           .
*.  RB = NONZERO  ==> ILLEGAL CONSTANT ($ERINVCN)                     .
*.       CALLS SCANCO                                                  .
*.       **NOTE** EXPRESSION ENDING IN  )  INSIDE MULTIPLE CONSTANT    .
*.       WILL BE PROCESSED IMPROPERLY, SUCH AS  DC  A(B+(C),D)  .      .
*.       THE CHARACTERS  C)  ARE TREATED AS END OF THE ACON.           .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: CACON2   1-2 SCAN ACON, ASSEMBLE VALUE . . . . . . . . . .
*.  RA = SCAN POINTER (ADDRESS OF 1ST CHAR AFTER PREVIOUS DELIMETER)  .
*.  RB = LENGTH-1 OF 1 CONSTANT OF 1 OPERAND TO BE ASSEMBLED          .
*.       EXIT CONDITIONS                                               .
*.  RA = SCAN POINTER (ADDRESS OF DELIMITER STOPPING SCAN, OR ERROR)  .
*.  RB = 0    CONSTANT WAS LEGAL, NO ERRORS                           .
*.  RB = NONZERO VALUE - ERROR CODE (FROM EVALUT)                     .
*.     = $ERRELOC  IF SECTION ID IS A DSECT, WHICH IS NOT ALLOWED.    .
*.  RC = ADDRESS OF PROPERLY ASSEMBLED CONSTANT                       .
*.  RD = ESDID OF CONSTANT, IF =0 ==> ABSOLUTE EXPRESSION             .
*.       CALLS EVALUT                                                  .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> CSECT: CBCONS   1-2 PROCESS BINARY CONSTANTS. . . . . . . . . . .
*.       USES DSECTS: AVWXTABL                                         .
*.       USES MACROS: $RETURN,$SAVE                                    .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: CBCON1   1   SCAN B CONSTANT, DO NOT ASSEMBLE. . . . . . .
*.       ENTRY CONDITIONS                                              .
*.  RA = SCAN POINTER (ADDRESS OF 1ST CHAR AFTER PREVIOUS DELIMETER)  .
*.       EXIT CONDITIONS                                               .
*.  RA = SCAN POINTER (ADDRESS OF DELIMITER STOPPING SCAN, OR ERROR)  .
*.  RB = 0    CONSTANT WAS LEGAL, NO ERRORS                            .
*.  RB = NONZERO VALUE FOR ERROR CODE - INVALID CONSTANT - ($ERINVCN) .
*.  RC = NUMBER OF BYTES REQUIRED FOR CONSTANT                         .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: CBCON2   1-2 ASSEMBLE BINARY CONSTANT. . . . . . . . . . .
*.       ENTRY CONDITIONS                                              .
*.  RA = SCAN POINTER (ADDRESS OF 1ST CHAR AFTER PREVIOUS DELIMETER)  .
*.  RB = LENGTH-1 OF 1 CONSTANT OF 1 OPERAND TO BE ASSEMBLED           .
*.       EXIT CONDITIONS                                               .
*.  RA = SCAN POINTER (ADDRESS OF DELIMITER STOPPING SCAN, OR ERROR)  .
*.  RC = ADDRESS OF PROPERLY ASSEMBLED CONSTANT                        .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> CSECT: CCCONS   1-2 PROCESS CHARACTER TYPE CONSTANTS. . . . . . .
*.       USES DSECTS: AVWXTABL                                         .
*.       USES MACROS: $RETURN,$SAVE                                    .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: CCCON1   1   SCAN,RETURN LENGTH,DO NOT ASSEMBLE. . . . . .
*.       ENTRY CONDITIONS                                              .
*.  RA = SCAN POINTER (ADDRESS OF 1ST CHAR AFTER PREVIOUS DELIMETER)  .
*.       EXIT CONDITIONS                                               .
*.  RA = SCAN POINTER (ADDRESS OF DELIMITER STOPPING SCAN, OR ERROR)  .
*.  RB = 0    CONSTANT WAS LEGAL, NO ERRORS                            .
*.  RB = NONZERO VALUE FOR ERROR CODE - INVALID CONSTANT - ($ERINVCN) .
*.  RC = NUMBER OF BYTES REQUIRED FOR CONSTANT                         .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: CCCON2   2   SCAN, ASSEMBLE. . . . . . . . . . . . . . . .
*.       ENTRY CONDITIONS                                              .
*.  RA = SCAN POINTER (ADDRESS OF 1ST CHAR AFTER PREVIOUS DELIMETER)  .
*.  RB = LENGTH-1 OF 1 CONSTANT OF 1 OPERAND TO BE ASSEMBLED           .
*.       EXIT CONDITIONS                                               .
*.  RA = SCAN POINTER (ADDRESS OF DELIMITER STOPPING SCAN, OR ERROR)  .
*.  RC = ADDRESS OF PROPERLY ASSEMBLED CONSTANT                        .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> CSECT: CDECNS   1-2 PROCESS D&E TYPE CONSTS . . . . . . . . . . .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: CDECN1   1   SCAN, BUT DO NOT ASSEMBLE D OR E TYPE CONSTS.
*.       ENTRY CONDITIONS                                              .
*.  RA = SCAN POINTER (ADDRESS OF 1ST CHAR AFTER PREVIOUS DELIMETER)  .
*.       EXIT CONDITIONS                                               .
*.  RA = SCAN POINTER (ADDRESS OF DELIMITER STOPPING SCAN, OR ERROR)  .
*.  RB = 0    CONSTANT WAS LEGAL, NO ERRORS                            .
*.  RB = NONZERO VALUE FOR ERROR CODE - INVALID CONSTANT - ($ERINVCN) .
*.       CALLS CDECN2                                                  .
*.       USES DSECTS: AVWXTABL                                         .
*.       USES MACROS: $RETURN,$SAVE                                    .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: CDECN2   1-2 SCAN,ASSEMBLE D&E TYPE CONSTANTS. . . . . . .
*.       ENTRY CONDITIONS                                              .
*.  RA = SCAN POINTER (ADDRESS OF 1ST CHAR AFTER PREVIOUS DELIMETER)  .
*.  RB = LENGTH-1 OF 1 CONSTANT OF 1 OPERAND TO BE ASSEMBLED          .
*.       EXIT CONDITIONS                                               .
*.  RA = SCAN POINTER (ADDRESS OF DELIMITER STOPPING SCAN, OR ERROR)  .
*.  RB = 0    CONSTANT WAS LEGAL, NO ERRORS                            .
*.  RB = NONZERO VALUE FOR ERROR CODE - INVALID CONSTANT - ($ERINVCN) .
*.  RC = ADDRESS OF PROPERLY ASSEMBLED CONSTANT                        .
*.       CALLS SDDTRM                                                  .
*.       USES DSECTS: AVWXTABL                                         .
*.       USES MACROS: $CALL,$RETURN,$SAVE                              .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> CSECT: CFHCNS   1-2 PROCESS FULLWORD-HALFWORD CONSTANTS . . . . .
*.       USES DSECTS: AVWXTABL                                         .
*.       USES MACROS: $RETURN,$SAVE                                    .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: CFHCN1   1   SCAN CONST, DO NOT ASSEMBLE . . . . . . . . .
*.       ENTRY CONDITIONS                                              .
*.  RA = SCAN POINTER (ADDRESS OF 1ST CHAR AFTER PREVIOUS DELIMETER)  .
*.       EXIT CONDITIONS                                               .
*.  RA = SCAN POINTER (ADDRESS OF DELIMITER STOPPING SCAN, OR ERROR)  .
*.  RB = 0    CONSTANT WAS LEGAL, NO ERRORS                            .
*.  RB = NONZERO VALUE FOR ERROR CODE - INVALID CONSTANT - ($ERINVCN) .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: CFHCN2   2   ASSEMBLE F OR H CONST . . . . . . . . . . . .
*.       ENTRY CONDITIONS                                              .
*.  RA = SCAN POINTER (ADDRESS OF 1ST CHAR AFTER PREVIOUS DELIMETER)  .
*.  RB = LENGTH-1 OF 1 CONSTANT OF 1 OPERAND TO BE ASSEMBLED          .
*.       EXIT CONDITIONS                                               .
*.  RA = SCAN POINTER (ADDRESS OF DELIMITER STOPPING SCAN, OR ERROR)  .
*.  RB = 0    CONSTANT WAS LEGAL, NO ERRORS                            .
*.  RB = NONZERO VALUE FOR ERROR CODE - INVALID CONSTANT - ($ERINVCN) .
*.  RC = ADDRESS OF PROPERLY ASSEMBLED CONSTANT                        .
*.       **NOTE** - THIS ROUTINE WILL ASSEMBLE VALUES INTO F OR H     *
*.       CONSTANTS OF LENGTH 1-8, BUT THE VALUE OF ANY CONSTANT MUST  *
*.       BE OF SIZE TO FIT INTO 1 FULLWORD, I.E. THE OTHER FULLWORD   *
*.       MUST EITHER BE ALL 0'S OR ALL 1'S (BINARY).  .               *
*.       **NOTE** IT IS POSSIBLE FOR THIS ROUTINE TO CAUSE A FIXED PT *
*.       OVERFLOW, WHICH WILL CAUGHT AND LAGGED BY SPIE MONITOR IN    *
*.       MAIN PROGRAM MPCON0.                                          *
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> CSECT: CNDTL2   2   CONSTANT PROCESSOR CONTROL - PASS 2 . . . . .
*.       ENTRY CONDITIONS                                              .
*.  RB = NUMBER OF CONSTANT CONTROL BLOCKS TO BE PROCESSED             .
*.  RC = ADDRESS OF FIRST OR ONLY CNCBLOCK TO BE DONE                  .
*.       CALLS CACON2,CBCON2,CCCON2,CDECN2,CFHCN2,CPCON2,CVCON2,CXCON2.
*.       CALLS CZCON2,ERRTAG,OUTPT2,UTPUT2                             .
*.       USES DSECTS: AVWXTABL,CNCBLOCK                                .
*.       USES MACROS: $ALIGR,$CALL,$GLOC,$GTAD,$RETURN,$SAVE           .
*.       USES MACROS: $SCPT,$SLOC                                      .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> CSECT: CODTL1   1   SCAN DUPFAC,TYPE,LENGTH-CALL C ROUTINES . . .
*.       ENTRY CONDITIONS                                             .
*.  RA = SCAN POINTER TO DUPLICATION FACTOR OR CONSTANT TYPE         .
*.  RB = 0    CONSTANT IS IN A DEFINE STORAGE STMT                    .
*.  RB = 4     CONSTANT IS IN A DC STATEMENT                          .
*.  RB = 8     CONSTANT IS A LITERAL - (I.E. DUPLFAC ^= 0, DECIMALS)  .
*.       EXIT CONDITIONS                                              .
*.  RA = SCAN POINTER TO DELIMITER FOLLOWING CONSTANT                 .
*.  RB = 0    LEGAL SPECIFICATION OF CONSTANT                         .
*.  RB = NONZERO VALUE - ERROR CODE - ILLEGAL                         .
*.  RC = ADDRESS OF A CONSTANT CONTROL BLOCK                          .
*.  RE = TOTAL LENGTH OF OPERAND,INCLUDING MULTIPLE OPERANDS,IF ANY   .
*.       CALLS CACON1,CBCON1,CCCON1,CDECN1,CFHCN1,CPCON1,CVCON1,CXCON1.
*.       CALLS CZCON1,EVALUT,SDDTRM                                   .
*.       USES DSECTS: AVWXTABL,CONBLK                                 .
*.       USES MACROS: $CALL,$GTAD,$RETURN,$SAVE,$SCOF,CONG
*        NOTE RESTRICTIONS - DUPLICATION FACTOR AND TOTAL LENGTH MUST *
*        BOTH BE ABLE TO FIT IN HALFWORD EACH. LENGTH MAY BE GREATER  *
*        THAN 256 FOR A DS,BUT LENGTH ATTRIBUTE WILL NOT BE CORRECT   *
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> CSECT: CPCONS   1-2 PROCESS PACKED CONSTANTS. . . . . . . . . . .
*.       USES DSECTS: AVWXTABL                                         .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: CPCON1   1   SCAN,DO NOT ASSEMBLE PACKED CONSTATNT . . . .
*.       ENTRY CONDITIONS                                              .
*.  RA = SCAN POINTER (ADDRESS OF 1ST CHAR AFTER PREVIOUS DELIMETER)  .
*.       EXIT CONDITIONS                                               .
*.  RA = SCAN POINTER (ADDRESS OF DELIMITER STOPPING SCAN, OR ERROR)  .
*.  RB = 0    CONSTANT WAS LEGAL, NO ERRORS                            .
*.  RB = NONZERO VALUE FOR ERROR CODE - INVALID CONSTANT - ($ERINVCN) .
*.  RC = NUMBER OF BYTES REQUIRED FOR CONSTANT                         .
*.       USES MACROS: $RETURN,$SAVE                                    .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: CPCON2   1-2 SCAN AND ASSEMBLE P TYPE CONSTANT . . . . . .
*.       ENTRY CONDITIONS                                              .
*.  RA = SCAN POINTER (ADDRESS OF 1ST CHAR AFTER PREVIOUS DELIMETER)  .
*.  RB = LENGTH-1 OF 1 CONSTANT OF 1 OPERAND TO BE ASSEMBLED          .
*.       EXIT CONDITIONS                                               .
*.  RA = SCAN POINTER TO DELIMITER ENDING SCAN                         .
*.  RC = ADDRESS OF PROPERLY ASSEMBLED CONSTANT                        .
*.       USES MACROS: $RETURN,$SAVE,$SETRT                             .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> CSECT: CVCONS   1-2 PROCESS V-TYPE ADCONS . . . . . . . . . . . .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: CVCON1   1   SCAN V-TYPE CONST, NO ASSEMBLE. . . . . . . .
*.       ENTRY CONDITIONS                                             .
*.  RA = SCAN POINTER (ADDRESS OF 1ST CHAR AFTER PREVIOUS DELIMETER)  .
*.       EXIT CONDITIONS                                              .
*.  RA = SCAN POINTER (ADDRESS OF 1ST CHAR AFTER PREVIOUS DELIMETER)  .
*.  RB = 0    CONSTANT WAS LEGAL, NO ERRORS                           .
*.  RB = NONZERO VALUE - ILLEGAL SYMBOL ($ERINVSY)                    .
*.       USES DSECTS: AVWXTABL                                        .
*.       USES MACROS: $RETURN,$SAVE                                   .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: CVCON2   2   SCAN&ASSEMBLE VCON. . . . . . . . . . . . . .
*.       ENTRY CONDITIONS                                             .
*.  RA = SCAN POINTER TO FIRST CHARACTER OF VCON.                     .
*.       EXIT CONDITIONS                                              .
*.  RA = SCAN POINTER (ADDRESS OF DELIMITER STOPPING SCAN, OR ERROR)  .
*.  RB = 0 ==> NO ERRORS, NONZERO ==> ERROR CODE                      .
*.     = NONZERO ERROR CODE ($ERUNRV  OR  $ERRELOC).                  .
*.  RC = ADDRESS OF PROPERLY ASSEMBLED CONSTANT                       .
*.       CALLS SYFIND                                                 .
*.       CALLS RESYMB (ONLY IF &$REPL=2 AND EXTRN SYMBOL USED).       .
*.       USES DSECTS: AVWXTABL,SYMSECT                                .
*.       USES MACROS: $CALL,$RETURN,$SAVE                             .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> CSECT: CXCONS   1-2 PROCESS HEXADECIMAL CONSTANTS . . . . . . . .
*.       USES DSECTS: AVWXTABL                                         .
*.       USES MACROS: $RETURN,$SAVE                                    .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: CXCON1   1   SCAN HEX CONST, DO NOT ASSEMBLE . . . . . . .
*.       ENTRY CONDITIONS                                              .
*.  RA = SCAN POINTER (ADDRESS OF 1ST CHAR AFTER PREVIOUS DELIMETER)  .
*.       EXIT CONDITIONS                                               .
*.  RA = SCAN POINTER (ADDRESS OF DELIMITER STOPPING SCAN, OR ERROR)  .
*.  RB = 0    CONSTANT WAS LEGAL, NO ERRORS                            .
*.  RB = NONZERO VALUE FOR ERROR CODE - INVALID CONSTANT - ($ERINVCN) .
*.  RC = NUMBER OF BYTES REQUIRED FOR CONSTANT                         .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: CXCON2   1-2 ASSEMBLE HEX CONSTANT . . . . . . . . . . . .
*.       ENTRY CONDITIONS                                              .
*.  RA = SCAN POINTER (ADDRESS OF 1ST CHAR AFTER PREVIOUS DELIMETER)  .
*.  RB = LENGTH-1 OF 1 CONSTANT OF 1 OPERAND TO BE ASSEMBLED           .
*.       EXIT CONDITIONS                                               .
*.  RA = SCAN POINTER (ADDRESS OF DELIMITER STOPPING SCAN, OR ERROR)  .
*.  RC = ADDRESS OF PROPERLY ASSEMBLED CONSTANT                        .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> CSECT: CZCONS   1-2 PROCESS ZONED CONSTS. . . . . . . . . . . . .
*.       USES DSECTS: AVWXTABL                                         .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: CZCON1   1   SCAN, BUT DO NOT ASSEMBLE . . . . . . . . . .
*.       ENTRY CONDITIONS                                              .
*.  RA = SCAN POINTER (ADDRESS OF 1ST CHAR AFTER PREVIOUS DELIMETER)  .
*.       EXIT CONDITIONS                                               .
*.  RA = SCAN POINTER (ADDRESS OF DELIMITER STOPPING SCAN, OR ERROR)  .
*.  RB = 0    CONSTANT WAS LEGAL, NO ERRORS                            .
*.  RB = NONZERO VALUE FOR ERROR CODE - INVALID CONSTANT - ($ERINVCN) .
*.  RC = NUMBER OF BYTES REQUIRED FOR CONSTANT                         .
*.       USES MACROS: $RETURN,$SAVE                                    .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: CZCON2   1-2 SCAN AND ASSEMBLE Z-TYPE CONSTANT . . . . . .
*.       ENTRY CONDITIONS                                              .
*.  RA = SCAN POINTER (ADDRESS OF 1ST CHAR AFTER PREVIOUS DELIMETER)  .
*.  RB = LENGTH-1 OF 1 CONSTANT OF 1 OPERAND TO BE ASSEMBLED           .
*.       EXIT CONDITIONS                                               .
*.  RA = SCAN POINTER (ADDRESS OF DELIMITER STOPPING SCAN, OR ERROR)  .
*.  RC = ADDRESS OF PROPERLY ASSEMBLED CONSTANT                        .
*.       USES MACROS: $RETURN,$SAVE,$SETRT                             .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> CSECT: ERRORS   1-2 ERROR FLAGGING ROUTINES . . . . . . . . . . .
*.       ENTRY CONDITIONS                                              .
*.  RA = SCAN POINTER TO CAUSE OF ERROR                                .
*.  RB = ERROR CODE                                                    .
*.       EXIT CONDITIONS                                               .
*.  RA,RB ARE UNCHANGED BY ERRTAG OR ERRLAB                            .
*.       USES DSECTS: AVWXTABL,RSBLOCK                                 .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: ERRLAB       FLAG ERROR FOR A LABEL. . . . . . . . . . .
*.       ENTRY CONDITIONS-EXIT CONDITIONS - SEE CSECT ERRORS           .
*.       CALLS ERRTAG                                                  .
*.       USES MACROS: $CALL,$RETURN,$SAVE                              .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: ERRTAG       FLAG ERROR AT SCAN POINTER POSITION . . . . .
*.       ENTRY CONDITIONS-EXIT CONDITIONS - SEE CSECT ERRORS           .
*.       USES MACROS: $RETURN,$SAVE,$SCOF                              .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> CSECT: ESDOPRS  1-2 EXTERNAL SYMBOL DICTIONARY&ESDID OPERATIONS .
*.       THIS MODULE HANDLES ALL FLAGGING AND CHECKING OF SECTION     .
*.       AND EXTERNAL ATTRIBUTES, INCLUDING FLAGGING SYMBOL TABLE     .
*.       ENTRIES AND MANIPULATING LOCATION COUNTERS AND SECTION IDS.  .
*.       USES DSECTS: AVWXTABL,SYSMSECT                               .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: ESCSEC       DECLARE A CONTROL SECTION OR DUMMY SECTION. .
*.       ENTRY CONDITIONS                                             .
*.  RB = 0 ==> CSECT                                                  .
*.     = 2 ==> DSECT                                                  .
*.     = 4 ==> START                                                  .
*.  RC = VALUE TO BE USED TO SET LOCATION COUNTER(START ONLY,RB=4)    .
*.       EXIT CONDITIONS                                              .
*.  RB = 0 ==> NO ERRORS.  ^=0 ==> AN ERROR CODE TO BE SET            .
*.  RB = NONZERO VALUE - ERROR CODE - ($ERDPCSE)                      .
*.  AVCESDID  IS INCREMENTED BY 1 OR 2 FOR NEXT VALUE OF REQUIRED TYPE.
*.       I.E. CSECTS HAVE EVEN VALUES, DSECTS ODD ONES.               .
*.  LOCATION COUNTERS ARE MODIFIED (AVLOCHIH,AVLOCNTR).               .
*.       USES MACROS: $ALIGR,$AL2,$GLOC,$RETURN,$SAVE,$SLOC           .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: ESENX1       ENTRY AND EXTRN STATEMENTS- PASS 1. . . . . .
*.       ENTRY CONDITIONS                                             .
*.  RA = SCAN POINTER                                                 .
*.  RB = 0 ==> ENTRY                                                  .
*.     = 2 ==> EXTRN                                                  .
*.       EXIT CONDITIONS                                              .
*.  RA = SCAN POINTER TO BLANK FOLLOWING OPERAND FIELD, OR ERROR      .
*.  RB = 0 ==> NO ERRORS.  ^= 0 ==> ERROR CODE TO BE SET              .
*.  RB = NONZERO VALUE - ERROR CODE - ($ERINVDM,$ERINVSY)             .
*.  ALL LABEL'S IN STMT HAVE SYMSECTS FLAGGED APPROPRIATELY.          .
*.       CALLS SYENT1                                                 .
*.       USES MACROS: $CALL,$GTAD,$RETURN,$SAVE                       .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: ESENX2       ENTRY AND EXTRN STATEMENTS - PASS 2 . . . . .
*.       CHECKS ENTRY/EXTRN STATEMENTS FOR CONFLICTS, ERRORS.         .
*.       ENTRY AND EXIT CONDITIONS EXACTLY SAME AS ESENX1             .
*.             EXCEPT EXIT VALUE OF RB MEANS NOTHING.                 .
*.       CALLS ERRTAG,SYENT1                                          .
*.       USES MACROS: $CALL,$GTAD,$RETURN,$SAVE                       .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: ESINT1       INITIALIZATION . PASS 1 . . . . . . . . . . .
*.       THIS SECTION FOR COMPLETENESS, FUTURE USE. DOES NOTHING 8/70..
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> CSECT: EVALUT   1-2 GENERAL EXPRESSION EVALUATION ROUTINE . . . .
*.        ENTRY CONDITIONS                                              .
*.   RA = SCAN POINTER (ADDRESS OF 1ST CHARACTER OF EXPRESSION)         .
*.        EXIT CONDITIONS                                               .
*.   RA = SCAN POINTER TO DELIMITER STOPPING SCAN, OR ERROR             .
*.   RB = 0 ==> EXPRESSION GOOD, = NONZERO VALUE==>ERROR CODE           .
*.   RC = VALUE OF EXPRESSION, IF IT WAS GOOD                           .
*.   RD = 0 ==> EXPRESSION WAS AN ABSOLUTE EXPRESSION                   .
*.      = ESDID FOR A RELOCATABLE EXPRESSION (1-255)                    .
*.   RE = LENGTH ATTRIBUTE - 1 OF EXPRESSION.                           .
*.        CALLS SDBCDX,SYFIND                                           .
*.        USES DSECTS: AVWXTABL,EVCTDSCT,RCODBLK,RSBLOCK,SYMSECT        .
*.        USES MACROS: $CALL,$GLOC,$RETURN,$SAVE,EVCG                   .
*.                                                                      .
*.        **NOTE** SEE IBM PLM Y26-3700-0, PP. 45-47. EVALUT SOMEWHAT  .
*.        RESEMBLES IEUF7V-EXPRESSION EVALUATION ROUTINE.  NOTE EVALUT .
*.        HAS 1 LESS STATE SETTING, SINCE IEUF7V COND=0 IS UNNEEDED.   .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> CSECT: EXECUT   3   INTERPRETER SECTION . . . . . . . . . . . . .
*.             EXECUT PERFORMS ALL 360 INSTRUCTION SIMULATION DURING  .
*.       INTERPRETIVE EXECUTION OF THE USER PROGRAM.  ALL CONTROL     .
*.       VALUES FOR THIS MODULE ARE CONTAINED IN DSECT ECONTROL, WHICH.
*.       IS PASSED TO EXECUT BY THE CALLING PROGRAM.  THE INSTRUCTION..
*.       SET SIMULATED INCLUDES THE FOLLOWING:                        .
*.             1. STANDARD INSTRUCTION SET                            .
*.             2. DECIMAL INSTRUCTION SET (IF PRESENT ON MACHINE).    .
*.             3. FLOATING POINT INSTRUCTIONS (OPTIONAL).             .
*.             4. X-MACRO PSEUDO INSTRUCTIONS - XDUMP, XLIMD,         .
*.               XPNCH, XPRNT, XREAD.                                 .
*.       THE PRIVILEGED OPERATIONS MAY BE DECODED TO THE POINT OF     .
*.       BRANCHING TO INDIVIDUAL INSTRUCTION HANDLERS, BUT THEY ARE   .
*.       ARE FLAGGED WITH AN 0C2 INTERRUPT AT PRESENT, AND ARE NOT    .
*.       INTERPRETED FURTHER.  THE CODE PRESENT IS FOR FUTURE USE.    .
*.             THE SVC INSTRUCTION IS CURRENTLY FLAGGED WITH AN 0C2 IF.
*.       USED, BUT CODE EXISTS TO HANDLE ALL SVC CALLS IN A TABLE-    .
*.       DRIVEN WAY, USING THE @ OF AN SVC CONTROL TABLE PASSED IN THE.
*.       WORD ECSVCADS IN ECONTROL.  AS OF 8/2/70, THERE ARE NOT SVC  .
*.       ROUTINES, BUT THE CODE EXISTS FOR FUTURE USE.                .
*.             GENERAL CODE IS ALSO PROVIDED FOR ANY ADDITIONAL NEW   .
*.       INSTRUCTIONS OR I/O SIMULATORS BY THE SECTION EXCALL, WHICH  .
*.       ALLOWS CALLS TO EXTERNAL ROUTINES (WHICH WOULD BE USED BY    .
*.       ANY SVC CALLS, IF THERE ARE ANY).                            .
*.       ENTRY CONDITIONS                                             .
*.  R10= @ ECONTROL - EXECUTION CONTROL BLOCK.                        .
*.  ECONTROL CONTAINS ALL INITIAL VALUES FOR REGS,LIMITS,ETC.         .
*.       EXIT CONDITIONS                                              .
*.  ECINTCOD   CONTAINS INTERRRUPT CODE, IF PROGRAM INTERRUPT.        .
*.  ECFLAG1    CONTAINS SPECIAL COMPLETION CODE, IF ANY.              .
*.  ECERRAD  = ADDRESS OF AN ERCOMPCD ERROR COMPLETION CODE BLOCK     .
*.  ECONTROL   CONTAINS ALL OTHER VALUES NEEDED FOR A COMPLETION DUMP..
*.       USES DSECTS: ECONTROL,ECSTACKD                               .
*.       USES MACROS: $AL2,$ERCGN,$PNCH,$PRNT,$READ,$RETURN,$SAVE     .
*.       USES MACROS: $SPIE, XDECI, XDECO, XSNAP                      .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> CSECT: IAMOP1   1   MACHINE OPERATIONS - PASS 1 . . . . . . . . .
*.       THIS IS 1 OF 2 PASS 1,LEVEL 2 PROGRAMS.  IT PERFORMS ALL     .
*.       PASS 1 MACHINE INSTRUCTION PROCESSING, INCLUDING ALIGNMENT   .
*.       OF THE LOCATION COUNTER, SCANNING FOR LITERAL CONSTANTS,     .
*.       AND BUILDING AN RCODBLK FOR THE STATEMENT.  THE RCODBLK      .
*.       INCLUDES THE INSTRUCTION FORMAT TYPE, THE MACHINE CODE FOR   .
*.       THE GIVEN INSTRUCTION, MASK (EXTENDED MNEMONICS), FLAGS      .
*.       AND ALIGNMENT VALUES NEEDED, THE LENGTH ATTRIBUTE-1 FOR THE  .
*.       INSTRUCTION, AND THE ADDRESS OF A LITERAL CONSTANT IN THE    .
*.       LITERAL TABLE, IF THERE IS ONE USED.                         .
*.       ENTRY CONDITIONS                                             .
*.  RA = SCAN POINTER (ADDRESS OF 1ST CHARACTER OF OPERAND FIELD)     .
*.  RC = ADDRESS OF OPCODE CONTROL TABLE ENTRY FOR OPCODE USED        .
*.       EXIT CONDITIONS                                              .
*.  RB = 0    NO ERRORS WERE ENCOUNTERED                              .
*.     = >0   ERRORS WERE FOUND IN STATEMENT                          .
*.  RC = @ RECORD CODE BLOCK(RCODBLK) FOR THE STATEMENT.              .
*.       THE RCODBLK HAS ALL VALUES FILLED IN EXCEPT RCLOC(IARCLOC).  .
*.  RD = LENGTH OF CODE - TO BE ADDED AFTER ALIGNMENT DONE            .
*.       CALLS ERRTAG,LTENT1,SCANEQ                                   .
*.       USES DSECTS: AVWXTABL,OPCODTB                                .
*.       USES MACROS: $CALL,$CKALN,$GLOC,$LTENT1,$RETURN,$SAVE,$SLOC  .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> CSECT: IBASM1   1   ASSEMBLER INSTRUCTIONS - PASS 1 . . . . . . .
*.       THIS MODULE IS 1 OF THE 2 PASS 1,LEVEL 2 ROUTINES OF THE     .
*.       ASSIST ASSEMBLER.  IT PERFORMS ALL PROCESSING FOR ASSEMBLER  .
*.       INSTRUCTIONS DURING PASS 1, INCLUDING SCANNING, MODIFYING    .
*        LOCATION COUNTERS, AND BUILDING AN RCODBLK FOR THE STMT.     .
*.       ENTRY CONDITIONS                                             .
*.  RA = SCAN POINTER (ADDRESS OF 1ST CHARACTER OF OPERAND FIELD)     .
*.  RC = ADDRESS OF OPCODE CONTROL TABLE ENTRY FOR OPCODE USED        .
*.       EXIT CONDITIONS                                              .
*.  RB = 0    NO ERRORS WERE ENCOUNTERED                              .
*.     = >0   ERRORS WERE FOUND IN STATEMENT                          .
*.  RC = ADDRESS OF RECORD CODE BLOCK (RCB)                           .
*.  RD = LENGTH OF CODE - TO BE ADDED AFTER ALIGNMENT DONE            .
*.       CALLS CCCON1,CODTL1,ERRLAB,ERRTAG,ESCSEC,ESENX1              .
*.       CALLS EVALUT,LTDMP1,SDBCDX,SDDTRM                            .
*.       USES DSECTS: AVWXTABL,CNCBLOCK,IBPSECT,OPCODTB,SYMSECT       .
*.       USES MACROS: $AL2,$ALIGR,$CALL,$CKALN,$GLOC,$RETURN,$SAVE    .
*.       USES MACROS: $SDEF,$SLOC,IBPRTAB                             .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> CSECT: ICMOP2   2   MACHINE OPERATIONS - PASS 2 . . . . . . . . .
*.       THIS MODULE IS 1 OF THE 2 PASS 2,LEVEL 2 ROUTINES IN THE     .
*.       ASSIST ASSEMBLER.  IT PROCESSES ALL MACHINE INSTRUCTIONS IN  .
*.       THE SECOND PASS, SCANNING ALL THE OPERAND FIELDS AND CREATING.
*.       THE OBJECT CODE FOR THEM.  IT ALSO DOES THE SETUP REQUIREED  .
*.       FOR OUTPT2 TO PRODUCE THE PRINTED LISTING. THIS ROUTINE HAS  .
*.       MANY SPECIAL-CASE SECTIONS WHICH ARE USED FOR SPEED, AND     .
*.       WHICH COULD USE LESS SPACE IF CALLS TO THE GENERAL EXPRESSION.
*.       EVALUATOR EVALUT WERE USED INSTEAD.                          .
*.       ENTRY CONDITIONS                                             .
*.  RA = SCAN POINTER (ADDRESS OF 1ST CHARACTER OF OPERAND FIELD)     .
*.  RC = ADDRESS OF RECORD CODE BLOCK(RCODBLK) FOR STATEMENT          .
*.  RE = ADDRESS OF RECORD SOURCE BLOCK(RSBLOCK) FOR STATEMENT        .
*.       CALLS BRDISP,ERRTAG,EVALUT,LTGET2,SDBCDX,SDDTRM              .
*.       CALLS SDBCDX,SYFIND,OUTPT2,UTPT2                             .
*.       USES MACROS: $AL2,$CALL,$GLOC,$RETURN,$SAVE,ICT              .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> CSECT: IDASM2   2   ASSEMBLER INSTRUCTIONS - PASS 2 . . . . . . .
*.       THIS MODULE IS 1 OF THE 2 PASS 2,LEVEL 2 ROUTINES IN THE     .
*.       ASSIST ASSEMBLER.  IT PERFORMS ALL PROCESSING OF ASSEMBLER   .
*.       INSTRUCTIONS IN THE SECOND PASS.  IT PRODUCES SOME OBJECT    .
*.       CODE, AND DOES SETUP FOR PRINTING.  MOST OF THE WORK HAS     .
*.       ALREADY BEEN DONE IN THE CORREPONDING PASS 1 MODULE, IBASM1. .
*.       ENTRY CONDITIONS                                             .
*.  RA = SCAN POINTER (ADDRESS OF 1ST CHARACTER OF OPERAND FIELD)     .
*.  RC = ADDRESS OF RECORD CODE BLOCK(RCODBLK) FOR STATEMENT          .
*.  RE = ADDRESS OF RECORD SOURCE BLOCK(RSBLOCK) FOR STATEMENT        .
*.       CALLS BRDROP,BRUSIN,CCCON2,CNDTL2,ERRTAG,ESENX2,EVALUT,LTDMP2.
*.       CALLS OUTPT2,UTPUT2                                          .
*.       USES DSECTS: AVWXTABL,RCODBLK,RSBLOCK,SYMSECT                .
*.       USES MACROS: $AL2,$CALL,$GLOC,$RETURN,$SAVE,$SDEF,$STV       .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> INSUB: IDEVAL     EVALUATE RELOCATABLE EXPRESSION + + + + + + + +

**--> INSUB: IDREGET    CONVERT REGISTER, CHECK VALIDITY  + + + + + + +
**--> CSECT: INPUT1   1   INPUT AND MANIPULATION OF SOURCE CARDS. . . .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: INCARD       CALLED TO GET CARD AND CREATE RSBLOCK . . . .
*.       THIS ENTRY READS 1 STATEMENT (1-3 CARDS), AND SETS UP THE    .
*.       RECORD BLOCKS RSBLOCK, AND RSCBLK (IF CONTINUATIONS OR       .
*.       SEQUENCE NUMBERS ARE USED).  IT IS CALLED DURING PASS 1 OF   .
*.       THE ASSEMBLY.  IF AN ENDFILE INDICATION IS ENCOUNTERED, IT   .
*.       CREATES A PSEUDO ENDCARD, SINCE THE MAIN PROGRAM OF PASS 1   .
*.       MOCON1 ONLY STOPS AFTER AN END CARD IS FOUND.  AS OF 8/17/70,.
*.       INCARD IS THE ONLY ASSEMBLER ENTRY DOING CARD READING.       .
*.       IN SETTING UP THE RSBLOCK, INCARD CONCATENATES THE SECTIONS  .
*.       OF A CONTINUED STATEMENT, AND REMOVES BLANKS TO SOME DEGREE  .
*.       FROM THE TRAILING EDGE OF THE STATEMENT.  IT ALSO INSERTS    .
*.       THE 3 CHARACTERS BLANK,APOSTROPHE,BLANK AFTER THE LAST       .
*.       NONBLANK CHARACTER IN THE SOURCE STATEMENT.  THIS IS CRUCIAL .
*.       TO THE PROPER SCANNING OF THE SOURCE STATEMENT WITHOUT       .
*.       REQUIRING LENGTHS TO BE CARRIED FROM ROUTINE TO ROUTINE.     .
*.                                                                    .
*.             IF THE MACRO PROCESSOR EXISTS (&$MACROS=1), INCARD     .
*.       ALSO HANDLES RECOVERY OF GENERATED STMTS (CREATED BY MEXPND ..
*.       IN THE DYNAMIC-HIGH AREA).                                   .
*.             IF A MACRO LIBRARY FACILITY EXISTS (&$MACSLB=1),       .
*.       INCARD CAN BE SWITCHED TO READ FROM IT, INSTEAD OF $SORC.    .
*.       EXIT CONDITIONS                                              .
*.  RA = SCAN PTR TO ERROR, ONLY IF RB ^= 0.  NO MEANING IF RB = 0.   .
*.  RB = 0     NO ERRORS FOUND IN STATEMENT BY INCARD                 .
*.  RB = ERROR CODE (NONZERO) OF ERROR.  RA HAS SCAN PTR OF IT.       .
*.  AVSOLAST = @ BLANK IMMEDIATELY BEFORE ' IN THE 4-BYTE FIELDWHICH  .
*.       INCARD PLACES AFTER THE SOURCE STMT TO STOP SCANNING OVERRUN..
*.       USES DSECTS: AVWXTABL,RSBLOCK,RSCBLK,RSOURCE                 .
*.       USES MACROS: $RETURN,$SAVE,$SORC                             .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> INSUB: INCRSMV    SAVE CON/SEQNO INTO RSCBLK  + + + + + + + + + +
**--> CSECT: LTOPRS   1-2 ALL LITERAL TABLE OPERATIONS. . . . . . . . .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: LTDMP1   1   DUMP LITERALS ON FINDING LTORG AND END. . . .
*.       LTDMP1 IS CALLED BY IBASM1 TO FIND LENGTH OF THE CURRENT     .
*.       LITERRAL POOL, AND AVANCE THE CURRENT POOL PTR TO THE NEXT 1..
*.       EXIT CONDITIONS                                               .
*.  RA = TOTAL LENGTH REQUIRED FOR THE LITERAL BLOCK                   .
*.       CALLS MOSTOP                                                  .
*.       USES DSECTS: AVWXTABL,LTBASETB,LTLENTRY                       .
*.       USES MACROS: $ALIGN,$ALLOCH,$CALL,$GLOC,$RETURN,$SAVE         .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: LTDMP2   2   DUMP LITERALS IN PASS 2 . . . . . . . . . . .
*.       LTDMP2 IS CALLED BY IDASM2 DURING PASS 2, WHENEVER A LTORG   .
*.       OR END STMT IS FOUND, TO PRODUCE THE OBJECT CODE AND LISTING .
*.       OF ANY LITERALS IN THE CURRENT LITERAL POOL.  THE CURRENT    .
*.       POOL BASE POINTER IS ADVANCED TO THE NEXT LTBASETB.          .
*.       CALLS CNDTL2                                                  .
*.       USES DSECTS: AVWXTABL,LTBASETB,LTLENTRY                       .
*.       USES MACROS: $CALL,$GLOC,$RETURN,$SAVE,$SLOC                  .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: LTEND1   1   CLEANUP AFTER PHASE 1 PREPARE FOR PHASE 2 . .
*.       THIS ENTRY SETS UP FOR ASSEMBLER PASS 2 LITERAL PROCESSING.  .
*.       USES MACROS: $RETURN,$SAVE                                    .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: LTENT1   1   ENTER A LITERAL INTO THE TABLE. . . . . . . .
*.       THIS ENTRY IS CALLED DURING PASS 1 TO SCAN A LITERAL BY      .
*.       IAMOP1.  THE LITERAL IS SCANNED BY CODTL1, AND IT IS ENTERED .
*.       IF IT IS NOT ALREADY PRESENT.  NOTE THAT NO DUPLICATES       .
*.       ARE EVER KEPT IN THE SAME POOL, EVEN FOR A-TYPE CONSTANTS    .
*.       WITH LOCATION COUNTER REFERENCES.                             .
*.       ENTRY CONDITIONS                                              .
*.  RA = SCAN POINTER (ADDRESS OF = IN LITERAL)                        .
*.       EXIT CONDITIONS                                               .
*.  RA = SCAN POINTER (ADDRESS OF ERROR OR DELIMETER)                  .
*.  RB = 0 IF LITERAL LEGAL, ERROR CODE OTHER WISE                     .
*.  RC = ADDRESS OF LITERAL TABLE ENTRY                                .
*.       CALLS CODTL1,MOSTOP                                           .
*.       USES DSECTS: AVWXTABL,CNCBLOCK,LTBASETB,LTLENTRY,RSBLOCK      .
*.       USES MACROS: $ALLOCH,$CALL,$RETURN,$SAVE,$SCPT                .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: LTGET2   2   GET ADDRESS OF LITERAL IN ASSEMBLY. . . . . .
*.       LTGET2 IS CALLED BY ICMOP2 EACH TIME A LITERAL IS FOUND IN   .
*.       SCANNING MACHINE INST OPERANDS DURING PASS 2. IT RETURNS THE .
*.       ATTRIBUTES OF THE LITERAL, INCLUDING THE USER PROGRAM @ FOR  .
*.       THE LITERAL, THE SECTION ID OF THE LITERAL, AND THE LENGTH   .
*.       ATTRIBUTE OF THE LITERAL. ICMOP2 SUPPLIES A POINTER TO THE   .
*.       LTLENTRY OF THE LITERAL, WHICH HAD BEEN SAVED IN THE         .
*.       STATEMENT'S RCODBLK .                                         .
*.       ENTRY CONDITIONS                                              .
*.  RA = SCAN POINTER TO 1ST CHAR OF LITERAL =                         .
*.  RC = @ LITERAL TABLE ENTRY IN LITERAL TABLE(WAS SAVED IN RCB)     .
*.       EXIT CONDITIONS                                               .
*.  RA = SCAN POINTER TO CHARACTER AFTER LITERAL                       .
*.  RB = ESDID OF CSECT IN WHICH LITERAL EXISTS                        .
*.  RC = ADDRESS OF LITERAL (PROGRAM ADDRESS-FOR LISTING,ETC)          .
*.  RD = IMPLIED LENGTH-1 OF THE LITERAL(LOW ORDER BYTE, OTHERS INDTR).
*.       USES DSECTS: AVWXTABL,LTBASETB,LTLENTRY                       .
*.       USES MACROS: $RETURN,$SAVE                                    .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: LTINT1   1   INITIALIZE LITERAL TABLE IF NEEDED. . . . . .
*.  ALLOCATES AND ZEROS 1ST LITERAL POOL BASE TABLE. INITS 1ST AND    *
*.  CURRENT BLOCK POINTERS TO 1ST LTBASETB.                           *
*.       CALLS MOSTOP                                                  .
*.       USES DSECTS: AVWXTABL,LTBASETB                                .
*.       USES MACROS: $ALLOCH,$RETURN,$SAVE                            .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> CSECT: MACFND      THIS ROUTINE IS GENERAL SEARCH PROCEDURE      *
*.        WHICH CAN SCAN THE MACRO LIBRARY, GLOBAL AND LOCAL           *
*.        DICTIONARIES AND THE SYMBOLIC PARAMETER LIST.  THE CALLING  *
*.        ROUTINE DETERMINES WHICH LIBRARY BY PLACING THE APPROPRIATE *
*.        POINTER IN RC.                                               *
*.                                                                     *
*.        ENTRY CONDITIONS                                             *
*.     RC = @ OF FIRST ENTRY OF LIST TO BE SEARCHED                    *
*.                                                                     *
*.        EXIT CONDITIONS                                              *
*.     RB = 0 IF ENTRY IS FOUND                                        *
*.        = $ERUNDEF IF ENTRY IS NOT FOUND                             *
*.     RC = @ OF ENTRY IF FOUND ELSE @ OF FINAL ENTRY IF NOT FOUND    *
*.        USES MACROS: $SAVE, $RETURN                                  *
*.        USES DSECTS: MACLIB, AVWXTABL                                *
*.                                                                     *
*.REGISTER USAGE                                                       A
*.RC-MACLIB BASE REGISTER, LIST TO BE SEARCHED                         A
*.RAT- MAIN TABLE DSECT USING                                          A
*.RB-RETURN REGISTER                                                   A
*.                                                                     A
*. NAMES=MACFN___                                                      A
*.                                                                     A
*.                                                                     *
*.*********************************************************************
```

```
**--> CSECT: MACINT     THIS ROUTINE IS CALLED IN INITIALIZATION      *
*.       PHASE OF ASSIST.  IT PERFORMS CERTAIN REQUIRED STORAGE       *
*.       ALLOCATION AND SETS POINTERS AVGEN1CD AND AVGEN2CD.          *
*.       OVERFLOW MESSAGE FOR GENERAL USE IS ALSO CREATED.            *
*.                                G.M.CAMPBELL - SUMMER - 1972        *
*.                                                                    *
*.       USES MACROS:  $ALLOCL, $SAVE, $RETURN, $CALL                 *
*.        USES DSECTS: AVWXTABL                                       *
*.                                                                    *
*.       REGISTER USAGE:                                              S
*.             WORK REGS:  RA,RB                                      S
*.                                                                    *
*.*********************************************************************

**--> INSUB: MCINITOV       OVERFLOW ROUTINE + + + + + + + + + + + + +S
```

```
**--> CSECT: MACLEX     THIS PROCEDURE SCANS A MCRO STATEMENT AND     *
*.        CONVERTS IT INTO BSU'S.  ALSO CHECKS FOR SUCH ERRORS AS TWO *
*.        TERMS OR TWO OPERATORS IN A ROW.  WHERE NECESSARY IT INSERTS*
*.        CATENATION OPERATORS WHERE CATENATION IS IMPLICIT           *
*.                                                                    *
*.        ENTRY CONDITIONS                                            *
*.     RA = @ OF FIRST CHARACTER OF EXPRESSION                        *
*.     RC = @ ON NEXT AVAILABLE BSU IN WORKSPACE                      *
*.                                                                    *
*.        EXIT CONDITIONS                                             *
*.     RA = @ OF DELIM PAST EXPRESSION IF NO ERROR                    *
*.        = @ OF ERROR IF ERROR PRESENT                               *
*.     RB = 0 IF OKAY                                                 *
*.        = $ERMSSGE IF ERROR                                         *
*.     RC = @ OF NEXT AVAILABLE SPACE FOR BSU                         *
*.                                                                    *
*.        CALLS MCGTST,MCDTRM,SDBCDX,MCSYSR,MCATRM,MCGTST             *
*.        USES DSECTS: AVWXTABL,MCBSU,MCPARENT,MCGLBDCT,MCLCLDPV      *
*.        USES MACROS: $SAVE,$RETURN,$ALLOCL,$SCOF,$SCPT,$CALL,$SETRT *
*.                                                                    *
*.       REGISTER USAGE                                               A
*.        WORK REGS: R0,R1,R2,RY,RZ,RB,RC,RE                          A
*.        USED FOR TRT: R1,R2                                         A
*.        RW-BASE REG FOR BSU                                         A
*.        R13 BASE REG FOR THIS CSECT                                 A
*.        RAT- BASE REGISTER FOR MAIN TABLE                           A
*.       RX-UNUSED                                                    A
*.       RD-?                                                         A
*.                                                                    A
*** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

**--> INSUB: MCLXBMP       BUMP POINTER  + + + + + + + + + + + + + + +S

**--> INSUB: MCLXCATI    ROUTINE TO INSERT CONCATINATION + + + + + + +A
```

```
**--> CSECT: MACRO1     CALLED BY MAIN CONTROL WHEN MACRO OPCODE      *
*.        ENCOUNTERED.  AT PRESENT (DEC 31, 1971) ONLY MACRO          *
*.        DEFINITIONS ARE ALLOWED, NO CONDITIONAL ASSEMBLY.  MACRO1   *
*.        CREATES ENTRY IN MACLIB FOR FUTURE EXPANSION BY MEXPND      *
*.        ENTRY CONDITIONS                                            *
*.     RA = SCAN POINTER @ OF OPERAND                                 *
*.     RC = @ OPCODTB ENTRY FOR OPERATION                             *
*.                                                                    *
*.        CALLS MACSCN,OUTPT2,MACFND,ERRTAG,ERRLAB,MCVSCN,MCSCOP,     *
*.              MCBODY                                                *
*.        USES MACROS: $SAVE,$RETURN,$CALL,$ALLOCL                    *
*.        USES DSECTS: RSBLOCK,OPCODTB,AVWXTABL,MACLIB,MCPARENT       *
*.                                                                    *
*.       REGISTER USAGE:                                              S
*.             WORK REGS: R0,R1,R2,RA,RB,RD,RE                        S
*.             BASE REGS: RAT,RW,RX,RY,R13,RC                         S
*.             UNUSED: RZ                                             S
*.                                                                    S
*.*********************************************************************

**--> INSUB: MACRORD       MACRO READER  + + + + + + + + + + + + + + +S
```

```
**--> CSECT: MACSCN    SCANS MACRO INSTRUCTION STATEMENT. IDENTIFIES  *
*.        LABEL, OPCODE, OPERAND AND COMMENT (IF ANY) FIELDS.         *
*.        LOCATION OF EACH FIELD STORED IN AVMFLD_.  LENGTH OF EACH   *
*.        FIELD STORED IN AVMFLDL_.  TYPE OF EACH FIELD PLACED IN     *
*.        AVMFLDT_.  FIELDS ARE SET TO ZERO IF NOT PRESENT.           *
*.        AVMFLDT1 CONTAINS '&' IF VARIABLE SYMBOL AND '.' IF SEQUENCE*
*.        SYMBOL ELSE ZERO.  AVMFLDT2 CONTAINS 'I' IF OPCODE IS       *
*.        SUSPECTED MACRO INSTRUCTION, 'M' IF MACRO OPCODE (AIF,      *
*.        AGO, SETA, ETC), 'O' IF OPCODE IS REGULAR ASSEMBLER OR      *
*.        MACHINE INSTRUCION AND X'00' IF ANYTHING ELSE.              *
*.        SCANS NON STND CONTINUATION FILDS AND PLACES VALUES IN      *
*.        AVMFLD5 THRU AVMFLD8                                        *
*.                                                                    *
*.        ENTRY CONDITIONS                                            *
*.     RA = @ OF FIRST CAHARACTER OF STATEMENT                        *
*.        EXIT CONDITIONS                                             *
*.     RA = SAME AS ENTRY CONDITIONS                                  *
*.     RB = 4 IF COMMENT STATEMENT, 8 IF MACRO COMMENT, ELSE ZERO     *
*.     RC = @ OF OPCODTB ENTRY IF OPCODE = M OR O                     *
*.                                                                    *
*.        USES MACROS: $CALL, $SAVE, $RETURN, $SETRT                  *
*.        USES DSECTS: AVWXTABL, OPCODTB                              *
*.       CALLS ERRTAG,MCATRM,OPFIND                                   S
*.   NAMES: MAC----- OR MC------                                      S
*.             BASE REGS:  R13,RAT,RX,RC                             S*
*.             WORK REGS:  R1,R2,RA,RB,RW,RZ                         S*
*.*********************************************************************

**--> INSUB: MACSCBLN       SCAN FOR NON-BLANK CHAR  + + + + + + + + +S

**--> INSUB: MACSCHEK       CHECK FOR NON-STD COND CARD  + + + + + + +S

**--> INSUB: MACSCMMT       SCAN COMMENT FIELD + + + + + + + + + + + +S

**--> INSUB: MACSCOPR       FIND AND SCAN OPERAND + + + + + + + + + ++S

**--> INSUB: MACSCSTR       SCAN ARBITRARY STRING  + + + + + + + + + +S
```

```
**--> CSECT: MCATRM        THIS ROUTINE SCANS A TERM AND DETERMINES   *
*.        WHETHER IT IS A VALID ATTRIBUTE, IE I', K', L', N', S' OR T'*
*.        THE LENGTH (L'), SCALE (S') AND INTEGER (I') ATTRIBUTES ARE *
*.        NOT IMPLEMENTED AND ARE SO FLAGGED.                         *
*.                                                                    *
*.        ENTRY CONDITIONS                                            *
*.     RA = @ OF FIRST CHAR OF TERM                                   *
*.                                                                    *
*.        EXIT CONDITIONS                                             *
*.     RA = @ OF DELIM PAST QUOTE IF VALID ATTRIBUTE ELSE SAME AS     *
*.          ENTRY.                                                    *
*.     RB = 0 IF ATTRIBUTE                                            *
*.        = -4 IF NOT ATTRIBUTE                                       *
*.        = $ERMESSAGE IF NOT IMPLEMENTED                             *
*.     RC = TYPE OF ATTRIBUTE                                         *
*.                                                                    *
*.        USES MACROS: $SAVE, $RETURN                                 *
*.        USES DSECTS: AVWXTABL                                       *
*.                                                                    *
*.*********************************************************************
```

```
**--> CSECT: MCBODY     PROCESSES THE BODY OF MACRO DEFINITION.       *
*.        CALLED FORM MACRO1 AFTRR PROTOTYPE STATEMENT PROCESSED.    *
*.        INITIALIZES LOCAL DICTIONARY FOR CURRENT DEFINITION.        *
*.        PROCESSES EACH STATEMENT TILL MEND STATEMENT ENCOUNTERED.   *
*.        TERMINATES AND RETURNS AT THAT POINT                        *
*.                                                                    *
*.       IN OPEN-CODE MODE, ($MCOCFL1 ON IN MCLBFLG2),                *
*.       MCBODY ONLY PROCESSES STMT IN RSBLOCK                        *
*.       IF AVPRSAVE IS SET IN AVPRINT1, IT CALL MXMVSR               *
*.       TO SAVE STMT IN HIGH AREA, ELSE IT PRINTS IT IMMEDIATELY     *
*.        ENTRY CONDITIONS                                            *
*.     RC = @ OF MACLIB ENTRY OF CURRENT MACRO DEFINITION             *
*.                                                                    *
*.        USES MACROS: $SAVE,$RETURN,$CALL,$ALLOCL,$ALLOCH,$SCOF,     *
*.                     $SETRT                                         *
*.        USES DSECTS: AVWXTABL,MACLIB,MCLCLDPV,OPCODTB,RSBLOCK,MCBSU,*
*.                     MCSEQ,MCGLBDCT,MCOPQUAD                        *
*.        CALLS INCARD,ERRTAG,MACSCN,ERRLAB,MCVSCN,MACFND,SDDTRM,     *
*.              MCSYSR,MACLEX,MCGTST,OUTPT2,MCGNCD                    *
*.                                                                    *
*.  REGISTER USAGE *************************                          A
*.R13- BASE REGISTER AND SAVEAREA POINTER                             A
*.RAT-MAIN TABLE DSECT USING                                          A
*.RX- MACLIB DSECT USING                                              A
*.RY- LOCAL DICTIONARY DSECT UING                                     A
*.RZ-OPCODE TABLE DSECT USING                                         A
*.RB,RE,RA- WORK REGISTERS                                            A
*.R1,R2 USED IN TRT'S                                                 A
*.RET- RETURN REGISTER USED FOR INSUBS                                A
*.* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

**--> INSUB: MCBDBMP       BUMPS BSU POINTER + + + + + + + + + + + + +A

**--> INSUB: MCBDCATI   CREATE CONCOT BSU+ + + + + + + + + + + + + + +A

**--> INSUB: MCBDFLD  CREATES A PRINT BSU+ + + + + + + + + + + + + + +A

**--> INSUB: MCBDPR     PRINT STATEMENTS + + + + + + + + + + + + + + +A

**--> INSUB: MCBDSCAN SCANS STATEMENTS IN A MOCOR DEFINITION + + + + +A

**--> INSUB: MCBDSCFN    LOOKS FOR FIELDS  + + + + + + + + + + + + + +A

**--> INSUB: MCB01      CHECK LCLX,BLX FOR LABEL, OPCODE  + + + + + + J

**--> INSUB: MCB02      OBTAIN DIMENSION OF GBLX OR LCLX STMT + + + + J

**--> INSUB: MCB03: CHECK DIMENSION SIZE FOR GBLX,LCLX+ + + + + + + + J
```

```
**--> CSECT: MCDTRM     DECIMAL CONSTANT CONVERSION.  MCDTRM DECIDES  *
*.        SCAN POINTER IS POINTING AT LEGAL DECIAMAL TERM AND IF SO,  *
*.        CONVERTS TO BINARY FORM. HANDLES VALUES UP TO 2**31-1       *
*.                                                                    *
*.        ENTRY CONDITIONS                                            *
*.     RA = @ OF FIRST CHAR OF TERM                                   *
*.                                                                    *
*.        EXIT CONDITIONS                                             *
*.     RA = @ OF DELIMITER BEYOND CONSTANT                            *
*.        = SAME AS ENTRY IF ERROR                                    *
*.     RB = 0 IF CONSTANT WAS LEGAL                                   *
*.        = $ER MSSGE IF ILLEGAL TERM                                 *
*.     RC = VALUE OF CONSTANT, 0 TO 2**31-1                           *
*.                                                                    *
*.        USES DSECTS: AVWXTABL                                       *
*.        USES MACROS: $SAVE, $RETURN                                 *
*.REGISTER USAGE                                                      A
*.R12 -BASE REG                                                       A
*.RAT-MAIN TABLE DSECT USING                                          A
*.RD- SCAN POINTER                                                    A
*.                                                                    A
*.NAMES=MCD_____                                                      A
*.                                                                    A
*** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
**--> CSECT: MCGNCD    CONVERTS STRING OF BSU'S TO INTERNAL CODE      *
*.        IN ONE-OP FORM.  ONE-OPS ARE QUADRUPLES WITH OPRTR, TWO     *
*.        OPRNDS AND RESULT FIELD.   ADDRESS OF CURRENT GENERATED INST*
*.        IS IN AVMCRINS.  GEERATED CODE IS POINTED TO BY MCCODLNK    *
*.        FIELD IN MACLIB.  BSU STRING LOCATED IN AVMWRK1             *
*.                                                                    *
*.        ENTRY CONDITIONS                                            *
*.     RC = @ OF CURRENT MACLIB ENTRY                                 *
*.                                                                    *
*.        USES MACROS: $CALL,$SAVE,$RETURN,$SCOF,$SCPT,$ALLOCL,$ALLOCH*
*.        USES DSECTS: AVWXTABL,MCBSU,MCBSTRMS,MCBOPRST,MCOPQUAD,     *
*.                     MACLIB,MCSEQ                                   *
*.        CALLS MACFND, ERRTAG,                                       *
*.                                                                    *
*.        REGISTER USAGE:                                             S
*.             WORK REGS:  R0,R1,RA,RB,RC,RE                          S
*.             TRT BYTE REG:  R2                                      S
*.             RW  - BASE REG FOR BSU                                 S
*.             RX  - BASE REG FOR OPRND STACK                         S
*.             RY  - BASE REG FOR OPRTR STACK                         S
*.             RZ  - BASE REG FOR ONE-OP ENTRY                        S
*.             RAT - BASE REG FOR MAIN TABLE                          S
*.             R1  - BASE REG FOR MACLIB                              S
*.             RD  - UNUSED                                           S
*.                                                                    S
*.* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

**--> INSUB: MCGNALLO       ALLOCATE LOW CORE  + + + + + + + + + + + +S

**--> INSUB: MCMVTRMS       CREATE ONE BINARY ONE-OP + + + + + + + + +S

**--> INSUB: MCSEQSCN       ENTER SEQ SYMBOL IN DICT + + + + + + + + +S
```

```
**--> CSECT: MCGTST    THIS ROUTINE TAKES A STRING AS DELINEATED BY   *
*.        BEGINNING AND END POINTERS, OBTAINS STORAGE DYNAMICALLY AND *
*.        MOVES THE STING.  IF INSIDE QUOTES DOUBLE QUOTES WILL BE    *
*.        CRUNCHED TO ONE QUOTE                                       *
*.                                                                    *
*.        ENTRY CONDITIONS                                            *
*.     RA = @ OF FIRST CAHRACTER OF STRING                            *
*.     RB = @ OF DELIMITER PAST STRING                                *
*.                                                                    *
*.        EXIT CONDITIONS                                             *
*.     RA = @ OF DELIMITER PAST STRING                                *
*.     RC = @ OF STRING IN NEW STORAGE                                *
*.     RD = LENGTH OF STRING                                          *
*.                                                                    *
*.        USES MACROS: $SAVE, $RETURN, $ALLOCL                        *
*.        USES DSECTS: AVWXTABL                                       *
*.                                                                    *
*.   REGISTER USAGE                                                   A
*.       RAT-MAIN TABLE USING                                         A
*.       RA,RB,RC,RD-AS IN ENTR/EXIT CONDITIONS                       A
*.       RE,R1,R3-WORK REGISTERS                                      A
*** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
**--> CSECT: MCSCOP     THIS ROUTINE SCANS A MACRO INSTRUCTION        *
*.        OPERAND.  THE OPERAND MUST CONFORM TO A STANDARD VALUE AS   *
*.        LAID DOWN IN SECTION 8 OF IBM GC28-2514                     *
*.                                                                    *
*.        ENTRY CONDITIONS                                            *
*.       AVMBYTE1: FLAG $MSBLIST EXPECTED SET IF ALREADY INSIDE SUBLISS
*.                                                                    *
*.        EXIT CONDITIONS                                             *
*.     RA = DELIM PAST OPRND IF STND VALUE ELSE POINTS AT ERROR       *
*.     RB = 0 IF STANDARD VALUE ELSE $ER MESSAGE                      *
*.     RC = LENGTH OF OPERAND IF OKAY                                 *
*.     RD = TYPE OF OPERAND.  IN THIS CASE TYPE WILL BE ONE OF        *
*.          'O' (NULL), 'N' (SELF-DEFINING TERM) OR 'U' (ALL OTHERS)  *
*.             CAN BE 'S' AFTER SCANNING   (1ST SUBPOPERAND           S
*.     RE = VALUE OF SELF DEFINING TERM                               *
*.       AVMBYTE1: FLAG $MINQUOT HAS INDETERMINATE VALUE.             S
*.        USES MACROS: $SAVE, $RETURN, $SETRT, $CALL                  *
*.        USES DSECTS: AVWXTABL                                       *
*.        CALLS SDBCDX                                                *
*.                                                                    *
*.*********************************************************************

**--> INSUB: MCSET#     MODIFY TRT TABLE AWTZTAB  + + + + + + + + + + S
```

```
**--> CSECT: MCSYSR    SCANS SUSPECTED VARIABLE SYMBOL FOR LEGALITY.  *
*.        IF VARIABLE SYMBOL THEN PLACES IN AVMSYMBL.  THEN SEARCHES  *
*.        GLOBAL, LOCAL AND SYMBOLIC PARAMETER DICTIONARIES FOR SYMBOL*
*.                                                                    *
*.        ENTRY CONDITIONS                                            *
*.     RA = @ OF FIRST CHARACTER OF SYMBOL                            *
*.                                                                    *
*.        EXIT CONDITIONS                                             *
*.     RA = @ OF DELIMITER PAST VARIABLE SYMBOL IF OKAY               *
*.        = SAME AS ENTRY IF NOT VARIABLE SYMBOL OR IF NOT FOUND      *
*.     RB = $ERUNDEF  IF SYMBOL IS NOT FOUND                          *
*.     RB = 0 IF SYMBOL IS FOUND IN ONE OF THE DICTIONARIES           *
*.        = SET TO -4 IF RA DOES NOT POINT AT VARIABLE SYMBOL         *
*.     RC = POINTER TO SYMBOL ENTRY IF FOUND                          *
*.     RD = $GLOBAL IF SYMBOL PRESENT IN GLOBAL DICTIONARY            *
*.        = $LOCAL IF SYMBOL FOUND IN LOCAL DICTIONARY                *
*.        = $SYMPAR IF SYMBOL IS SYMBOLIC PARAMETER                   *
*.        = $SYSTEM IF SYMBOL IS SYTEM VARIABLE                       *
*.                                                                    *
*.        USES MACROS: $CALL, $SAVE, $RETURN                          *
*.        USES DSECTS: MCGLBDCT, MACLIB,AVWXTABL                      *
*.        CALLS MCVSCN, MACFND                                        *
*.                                                                    A
*.REGISTER USAGE ***************                                      A
*.R13 -BASE REGISTER AND SAVEAREA POINTER                             A
*.RC- BASE REGISTER FOR GLOBAL DSECT                                  A
*.RX- BASE REGISER FOR MACRO DICTIONARY                               A
*.                                                                    A
*.NAMES=MCSY____                                                      A
*.                                                                    A
*.* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
**--> CSECT: MCVSCN     THIS ROUTINE SCANS A STRING AND CHECKS        *
*.        FOR A LEGAL VARIABLE SYMBOL.  IF OKAY, SYMBOL IS MOVED INTO *
*.        AVMSYMBL IN AVWXTABL WHERE IT WILL BE UTILIZED IN SEARCHES. *
*.                                                                    *
*.        ENTRY CONDITIONS                                            *
*.     RA = @ OF FIRST CHARACTER OF STRING                            *
*.                                                                    *
*.        EXIT CONDITIONS                                             *
*.     RA = @ OF DELIMITER PAST SYMBOL IF LEGAL                       *
*.        = SAME AS ENTRY IF NOT VARIABLE SYMBOL                      *
*.     RB = 0 IF OKAY, <0 IF NOT VARIABLE SYMBOL,                     *
*.        = $ER MESSAGE IF ILLEGAL SYMBOL                             *
*.        USES MACROS: $SAVE, $RETURN                                 *
*.        USES DSECTS: AVWXTABL                                       *
*.                                                                    *
*.REGISTER USAGE                                                      A
*.RAT- MAIN TABLE DSECT USING                                         A
*.R1,R2 USED IN TRT'S                                                 A
*.RB- SET AS IN EXIT CONDITIONS ABOVE                                 A
*.                                                                    A
*.NAMES=MCVS____                                                      A
*.                                                                    A
*.*********************************************************************
```

```
**--> CSECT: MEXPND   EXPANDS MACRO DEFINITION. RECURSIVE. ACQUIRES   *
*.        STORAGE FROM LOW DYNAMIC AREA FOR STANDARD SAVE AREA AND    *
*.        LOCAL VARIABLES.  RELEASES STORAGE ON EXIT.  PUTS GENERATED *
*.        STATEMENTS IN HIGH STORAGE.  AVGEN1CD POINTS TO FIRST BYTE  *
*.        AFTER FIRST STATEMENT.  AVGEN1CD POINTS TO 1ST BYTE OF LAST *
*.        STATEMENT GENERATED                                         *
*.                                                                    *
*.        USES MACROS:  $MALLOCL, $MALLOCH, $CALL, $SAVE, $RETURN,    *
*.                      $AL2                                          *
*.        USES DSECTS: MACLIB, MCGLBDCT, MCOPQUAD, MCPAROPR, MCPARSUB *
*.                     AVWXTABL, MXPNTSAV, MCPARENT, RSBLOCK          *
*.        CALLS ERRTAG, MCSCOP,MXMVSR,MACSCN,MACFND,MXMVSR,MXERRM,    *
*.              ERRTAG,MEXPND,DECTRM                                  *
*.                                                                    *
*.* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

**--> INSUB: MXPNOSYM       DETERMINS IF STRING IS ORDINARY  + + + + +S

**--> INSUB: MXPNRDR   + + + + + + + + + + + + + + + + + + + + + + + +S

**--> INSUB: MXPNSBSC  + + + + + + + + + + + + + + + + + + + + + + + +S

**--> INSUB: MXPOPKPR  + + + + + + + + + + + + + + + + + + + + + + + +S

**--> INSUB: MXPOPSCN  + + + + + + + + + + + + + + + + + + + + + + + +S
```

```
**--> CSECT: MOCON1   1   MAIN CONTROL - ASSEMBLER PASS 1 . . . . . . .
*.       MOCON1 PROVIDES OVERALL CONTROL FOR PASS 1 OF THE ASSIST     .
*.       ASSEMBLER, AND SUPERVISES OR PERFORMS THE FOLLOWING:         .
*.        1. READING INPUT CARDS, CREATING RECORD BLOCKS (INCARD).    .
*.        2. SCANNING LABELS, ENTERING THEM IN SYMBOL TABLE(SYENT1).  .
*.        3. SCANNING CARD FOR THE OPCODE, IF ANY.                    .
*.        4. FINDING OPCODE IN OPCODE TABLE (OPFIND).                 .
*.        5. SCANNING FOR OPERAND FIELD, SAVING SCAN POINTER.         .
*.        6. 2ND LEVEL INSTRUCTION PROCESSING (IAMOP1,IBASM1).        .
*.        7. DEFINING ATTRIBUTES, VALUE OF LABEL, IF REQUIRED.        .
*.        8. UPDATING LOCATION COUNTER TO NEXT LOCATION.              .
*.        9. STORING RECORD BLOCKS FOR STMT (UTPUT1).                 .
*.                                                                    .
*.       NOTE: PRINT CONTROL/COMMENTS STMTS ARE PROCESSED COMPLETELY  .
*.       DURING PASS 1 AND NOT SAVED, IF POSSIBLE.                    .
*.                                                                    .
*.       CALLS ERRLAB,ERRTAG,IAMOP1,IBASM1,INCARD,OPFIND,SYENT1,UTPUT1.
*.       CALLS OUTPT2
*.       USES DSECTS: AVWXTABL,OPCODTB,RCODBLK,RSBLOCK                .
*.       USES MACROS: $CALL,$GLOC,$GTAD,$PRNT,$RETURN,$SAVE           .
*.       USES MACROS: $SCOF,$SDEF,$SLOC                               .
*.       CALLS ERRLAB,ERRTAG,IAMOP1,IBASM1,INCARD,OPFIND,SYENT1       .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> INSUB: MOOPAMPC CHECK STATEMENT FOR SET VARIABLE SUBSTITUTION  *J

**--> ENTRY: MOSTOP     CALLED IF DISASTROUS ERROR OCCURS IN PASS 1 . .
*.       RESTORES CONDITIONS FOR MOCON1, NOTE OVERFLOW OCCURRENCE.    .
*.  ENDS EXECUTION FOR PASS 1, FLAGGING PROGRAM NONEXECUTABLE.       .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> CSECT: MPCON0   0   MAIN PROGRAM CONTROL-INIT,SET UP TABLES,ETC..
*.       MPCON0 INITIALIZES AVWXTABL DSECT VALUES FOR WHOLE ASSEMBLY, .
*.       SETS A $SPIE TO INTERCEPT SOME TYPES OF INTERRUPTS, SETS THE .
*.       PROGRAM AMSK TO ONLY HAVE FIXED-OVERFLOW INTRPTS, AND CALLS  .
*.       ALL THE SUBROUTINES REQUIRED FOR AN ASSEMBLY IN A TABLE-     .
*.       DRIVEN MANNER, USING A LIST OF POINTERS TO ADDRESS CONSTNATS..
*.       AFTER THE ASSEMBLY IS COMPLETED, IT PRINTS VARIOUS STATISTICS.
*.       AND THEN RETURNS CONTROL TO THE ASSIST MONITOR.  NOTE THAT   .
*.       MPCON0 IS THE ONLY CSECT IN THE ASSEMBLER WHICH ACTUALLY     .
*.       REFERS TO AJOBCON, ALTHOUGH OTHERS USE EQU FLAGS FROM IT.    .
*.       ENTRY CONDITIONS                                             .
*   R12(RAT)= @ VWXTABL CSECT, INITIALIZED BY ASSIST CONTROL PROG.    .
*   AVAJOBPT,AVECONPT HAVE BEEN INITIALIZED IF NEEDED BY ASSIST.      .
*.       CALLS ESINT1,LTINT1,OPINIT,SYINT1,UTINT1,OUINT1,MOCON1       .
*.       CALLS LTEND1,UTEND1,BRINIT,MTCON2                            .
*.       CALLS OUEND2,SYEND2,UTEND2                                   .
*.       USES DSECTS: AVWXTABL                                        .
*.       USES MACROS: $AL2, $CALL, $PRNT, $RETURN, $SAVE, $SPIE       .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> CSECT: MTCON2   2   MAIN CONTROL - ASSEMBLER PASS 2 . . . . . . .
*.       MTCON2 IS THE CONTROL PROGRAM FOR THE 2ND PASS OF THE ASSIST .
*.       OF THE ASSIST ASSEMBLER.  IT IS RELATIVELY SMALL, SINCE      .
*.       MOST OF THE WORK HAS BEEN DONE IN PASS 1.   IT PERFORMS OR   .
*.       SUPERVISES THE FOLLOWING ACTIONS, FOR EACH SOURCE STMT:      .
*.        1. RETRIEVES POINTERS TO THE RECORD BLOCKS (UTGET2).        .
*.        2. SETS UP THE LOCATION COUNTER AND OPERAND SCAN POINTER.   .
*.        3. CALLS 2ND LEVEL INSTRUCTION PROCESSORS(ICMOP2,IDASM2).   .
*.        4. PRINTS ANY STATEMENT WITH NO RCODBLK (OUTPT2).           .
*.       FINISH BY ROUNDING UP LENGTH OF PROG TO DOUBLEWORD BOUNDARY. .
*.       CALLS ICMOP2,IDASM2,OUTPT2,UTGET2                            .
*.       USES DSECTS: AVWXTABL,RCODBLK,RSBLOCK                        .
*.       USES MACROS: $CALL,$RETURN,$SAVE,$SLOC                       .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> CSECT: MXERRM    CALLED DURING MACRO GENERATION TO GENERATE     *
*.        ERROR MESSAGES NOT HANDLED BY ERRTAG                        *
*.                                                                    *
*.        ENTRY CONDITIONS                                            *
*.       RA-SCAN PTR                                                  A
*.     RB = ERROR TYPE                                                *
*.     RC = OPERAND VALUE OR LOCATION                                 *
*.     RD = LENGTH OF STRING IF CHAR VALUE                            *
*.       RE-@ MXPNTSACV                                               A
*.                                                                    *
*.       EXIT CONDITIONS                                              A
*.       RB=0 ==> OK                                                  A
*.       RB=4 ==> STORAGE OVERFLOW CAUSED MESSAGE SELECTED IS PLACED  A
*.       IN RSBLOCK, THEN MOVED OUT TO HIGH AREA BY MXMVSR            A
*.                                                                    A
*.        USES MACROS: $CALL, $AL2, $SAVE, $RETURN                    *
*.        CALLS MXMVSR                                                *
*.        USES DSECTS: RSBLOCK, MXPNTSAV, MCOPQUAD, AVWXTABL          *
*.                                                                    *
*.* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
**--> CSECT: MXINST       EXECUTE INSTRUCTIONS IN MACRO DEF          *S
*.             ENTRY CONDITIONS:                                     *S
*.       RC = @ MXPNTSAV                                             *S
*.             EXIT CONDITIONS:                                      *S
*.       RB = 0                    MEND OR MEXIT FOUND               *S
*.            4                    INNER MACRO CALL                  *S
*.            8                    KILL THIS MACRO NEST              *S
*.           12                    KILL ALL MACROS                   *S
*.           16                    STORAGE OVERFLOW                  *S
*.                                                                   *S
*.* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * **S

**--> INSUB: MXADDR       THIS ROUTINE ACCEPTS A ONE-OP  + + + + + + +S

**--> INSUB: MXARITH MXARITH PRODUCES ARITH ONE-OP                    A

**--> INSUB: MXINERRM       CALLS MXERRM TO HANDLE ERROR MESSAGES  + +S
```

```
**--> CSECT: MXMVSR    MOVES GENERATED STMT FROM RSBLOCK TO HIGH FREE *
*.        AREA. AVGEN2CD POINTS TO BEGINNING OF STMT                  *
*.                                                                    *
*.        EXIT CONDITIONS                                             *
*.     RB = ZERO IF OKAY ELSE 4 IF OVERFLOW                           *
*.                                                                    *
*.        USES MACROS: $SAVE, $RETURN, $MALLOCH                       *
*.       USES DSECTS RSBLOCK,REBLK,AVGEN1CD,AVGEN2CD                  A
*.                                                                    *
*.       REGISTER USAGES                                              A
*.       RAT-MAIN TABLE USING                                         A
*.       RW-SOURCE BLK USING                                          A
*.       RX-ERROR BLK USING                                           A
*.       R1,RB-BYTE REGISTERS                                         A
*.       RA-WORK REGISTER                                             A
*.                                                                    A
*.* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
**--> CSECT: OPCOD1   1   OPCODE TABLES AND LOOKUP CODE . . . . . . . .
*.       THIS MODULE CONTAINS THE CODE,TABLES TO IDENTIFY OPCODES.    .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: OPFIND   1   LOOK UP AN OPCODE . . . . . . . . . . . . . .
*.       ENTRY CONDITIONS                                             .
*.  RA = SCAN POINTER TO 1ST CHARACTER OF OPCODE                      .
*.       EXIT CONDITIONS                                              .
*.  RA = SCAN POINTER TO 1ST BLANK FOLLOWING LEGAL OPCODE,OR SAME AS O.
*.       ENTRY IF OPCODE WAS NOT RECOGNIZED.                          .
*.  RB = 0 IF THE OPCODE WAS FOUND IN OPCODE TABLE                    .
*.  RB = NONZERO VALUE - ERROR CODE FOR ILLEGAL OPCODE ($ERIVOPC)     .
*.  RC = ADDRESS OF OPCODTB ENTRY FOR THE OPCODE, IF IT WAS FOUND     .
*.       USES DSECTS: AVWXTABL,OPCODTB                                .
*.       USES MACROS: $RETURN,$SAVE,OPG,OPGT                          .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: OPINIT   1   INITILIAZE OPCODE ROUTINE IF NEEDED . . . . .
*.       AS OF 8/17/70, THIS ENTRY DOES NOTHING. IT IS INCLUDED FOR   .
*.       COMPLETENESS, POSSIBLE MODIFICATION REQUIRING INITIALIZATION..
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> CSECT: OUTPUT       PRINTED LISTING ROUTINE . . . . . . . . . . .
*.      OUTPUT HANDLES THE FORMATTING AND PRINTING OF THE ASSEMBLY   .
*.      LISTING FOR THE ASSIST ASSEMBLER.                            .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: OUEND2   2   PRINT ENDING STATISTICS FOR ASSMBLY . . . . .
*.      OUEND2 IS CALLED AT THE END OF THE ASSEMBLY TO PRINT SUMMARY .
*.      OF ERRORS AND WARNINGS ISSUED.  FIRST LINE PRINTED GIVES     .
*.      TOTAL # OF STMTS FLAGGED, TOTAL # ERRORS, TOTAL # WARNINGS.  .
*.      IF MAXIMUM # ERRORS IS EXCEEDED, ANOTHER LINE IS PRINTED.    .
*.      USES DSECTS: AVWXTABL                                        .
*.      USES MACROS: $PRNT,$RETURN,$SAVE                             .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: OUINT1   1   INITIALIZATION ENTRY - CALLED BEFORE PASS 1 .
*.      OUINT1 IS CALLED TO INITIALIZE FLAG VALUES AND COUNTERS      .
*.      USED IN OUTPUT, INCLUDING LISTING CONTROL, STATEMENT #,      .
*.      PAGE COUNT, WITHIN-PAGE LINE COUNT, AND TITLE AREA.          .
*.      USES DSECTS: AVWXTABL                                        .
*.      USES MACROS: $RETURN,$SAVE                                   .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> INSUB: OUTLNSA/        PRINT 1 LINE (WITH HEADING IF NEEDED)+ + +

**--> ENTRY: OUTPUT2      PRINT 1 STATEMENT,WITH CODE AS NEEDED,ERROR .
*.      OUTPT2 PRINTS 1 STATEMENT, WITH ANY ERROR MESSAGES NEEDED,   .
*.      PRINTS TITLES AND HEADINGS WHEN REQUIRED, PERFORMS PAGE AND  .
*.      LINE COUNTING, MAINTAINS LISTING CONTROL STATUS, AND KEEPS   .
*.      COUNTS OF NUMBER OF STATEMENTS FLAGGED, TOTAL # ERRORS,      .
*.      TOTAL # WARNING MESSAGES.                                    .
*.      ENTRY CONDITIONS                                             .
*.  RB = PRIMARY CALL TYPE CODE                                      .
*.     = 0    ($OUMACH) MACHINE INSTRUCTIONS                         .
*.     = 2    ($OUCONS) CONSTANTS,CNOPS,ETC. PRINT LOCATION COUNTER,CO.
*.     = 4    ($OULIST) - LISTING CONTROL - EJECT,SPACE,PRINT,TITLE  .
*.     = 6    ($OUCOMM) - COMMENTS,ETC.-DO NOT HAVE LOCATION COUNTER .
*.  RC = AN INFORMATION ADDRESS OF SOME TYPE                         .
*.     = @ OBJECT CODE (RB=0,2)                                      .
*.     = @ # LINES TO SPACE (RB=4,RE=0)                              .
*.     = @ PRINT CONTROL CODE BYTE (RB=4,RE=2) I.E. PRINT            .
*.     = @ TITLE CODE (RB=4,RE=4)                                    .
*.  RD = #-1 OF BYTES OF OBJECT CODE OR TITLE                        .
*.  RE = SECONDARY CODE OR ADDRESS                                   .
*.     = SECONDARY CODE FOR LISTING CONTROL OPERATIONS               .
*.     = 0    SPACE OR EJECT                                         .
*.     = 2    PRINT                                                  .
*.     = 4    TITLE                                                  .
*.      USES DSECTS: AVWXTABL,ICBLOCK,RCODBLK,RSBLOCK,RSCBLK,REBLK   .
*.      USES MACROS: $AL2,$PRNT,$RETURN,$SAVE,$SERR                  .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> INSUB: OUXCMINT   ENTRY POINT FOR CMPRS HANDLING  + + + + + + + +

**--> INSUB: OUXPRNT    LOW-LEVEL PRINT ROUTINE- 121-BYTE LINE  + + + +
**--> CSECT: REMONI     REPLACE MONITOR CONTROL PROGRAM . . . . . . . .
*.       REMONI HANDLES MOST OF THE DETIALS REQUIRED FOR A STUDENT TO .
*.  WRITE AN ASSIST CSECT, HAVE IT ASSEMBLED BY ASSIST, AND THEN RUN  .
*.  A TEST PROGRAM.  THE ENTRYPOINTS OF HIS PROGRAM ARE CALLED ALONG  .
*.  WITH THE ORIGINALS, AND HIS RESULTS CHECKED FOR ACCURACY.  WHILE  .
*.  ADDRESS CONSTANT MODIFCATION IS PERFORMED, THE ENTIRE PROCESS IS  .
*.  STILL A SERIALLY RESUABLE PROGRAM.  SEE THE ASSIST REPLACE USER'S .
*.  GUIDE   FOR DETAILS ON USING THE REPLACE MONITOR.                 .
*.       NAMES: RE------           MAIN CODE BODY AND INSUBS.         .
*.       NAMES: RG------           CHECKING CODE FOR RETURN VALUES.   .
*.       NAMES: RH------           EXTERNAL CALL CHECKING (&$REPL=2)  .
*.       CALLS SYFIND                                                 .
*.       USES DSECTS: AJOBCON,AVWXTABL,ECONTROL,RECORBLK,RFSYMBLK     .
*.       USES MACROS: $CALL,$PRNT,$RETURN,$SAVE,REPRNT,XDECO,XSNAP    .
*.                                                                    .
*.       OVERALL REGISTER CONVENTIONS AND USAGE.                      .
*.  R0,R1,R2,R3,R4,R15  WORK REGISTERS                                .
*.  R5 = @ RECORBLK ELEMENT FOR CURRENT ENTRY BEING PROCESSED.        .
*.  R6 = BASE REGISTER FOR MAIN CODE OF EACH REMONI ENTRY POINT.      .
*.  R7,R8      USUAL PARAMETER REGS FOR INTERNAL SUBROUTINES.         .
*.  R9 = LINK REGISTER FOR INSUBS WHICH MUST CALL OTHERS WITH R14.    .
*.  R10= @ ECONTROL (EXECUTION CONTROL BLOCK, USER PSEUDO REGISTERS.  .
*.  R11= @ AJOBCON DSECT (MAIN JOB CONTROL TABLE)                     .
*.  R12(RAT)= @ VWXTABL CSECT (AVWXTABL DSECT).                       .
*.  R13= SAVE AREA ADDRESS, BASE REGISTER FOR DATA, INTERNAL SUBRS.   .
*.  R14= INTERNAL LINK REGISTER. LOCAL WORK REGISTER.                 .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: REENDA     REPLACE MODULE: POST-ASSEMBLY PROCESSING  . . .
*.       REENDA IS CALLED JUST AFTER AN ASSEMBLY IS COMPLETED.        .
*.       IF THE RUN IS NOT A REPLACE RUN, NOTHING IS DONE.            .
*.       IF IT IS REPLACE PHASE A, THE ASSEMBLED PROGRAM WAS A REPLACE.
*.  VERSION OF AN ASSIST MODULE, SO CHECK AND MODIFY ASSEMBLER ADCONS..
*.       IF THE RUN IS IN PHASE B, THE ASSEMBLY JUST FINISHED WAS     .
*.  A TEST PROGRAM, SO PRINT PERFORMANCE STATISTICS FOR THE MODULE.   .
*.       ENTRY CONDITIONS                                             .
*.  R11= @ AJOBCON (MAIN JOB CONTROL BLOCK).                          .
*.  R12(RAT)= @ VWXTABL CSECT (AVWXTABL DSECT).                       .
*.       CALLS SYFIND                                                 .
*.       USES DSECTS: AJOBCON,AVWXTABL,RECORBLK,RFSYMBLK,SYMSECT      .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> INSUB: REEXDECO   CONVERT NUMBER TO DECIMAL   + + + + + + + + + +

**--> ENTRY: REFAKE     INTERCEPT REPLACED CALLS, CHECK REAL/USER . . .
*.       ENTRY CONDITIONS                                             .
*.  R15(BITS 0-7)= OFFSET CODE # FOR SPECIFIC ENTRY BEING CALLED.     .
*.  R0-R14     ARE AS DESCRIBED IN ASSEMBLER CALLING CONVENTIONS.     .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> INSUB: REFRFC     TEST ECRFLAG AND PRINT NEEDED INFOR + + + + + +

**--> INSUB: REGCRARE   USING RERGEFLG, FLAG AND PRINT REG MSG  + + + +

**--> INSUB: REGC1213   CHECK USER REGS 12-13,FLAG RERGEFLG + + + + + +

**--> ENTRY: REINTA     INITIALZE BEFORE ASSEMBLER CALLED . . . . . . .
```

```
*.       THIS ENTRY IS CALLED 1 TIME BEFORE ASSIST ASSEMBLER IS CALLED.
*.  IT CHECKS FOR PRESENCE OF REAL ADDRESS CONSTANTS IN VWXTABL, AND  .
*.  REPLACES THEM IF THEY HAVE BEEN MODIFIED IN PREVIOUS REPLACE RUN. .
*.       IT ALSO MAY SET FLAGS IN AVWXTABL IF THE SYSTEM IS IN        .
*.       REPLACE PHASE A (ASSEMBLE REPLACEMENT PROGRAM AND LINK IT).  .
*.       ENTRY CONDITIONS                                             .
*.  R11= @ AJOBCON (MAIN JOB CONTROL BLOCK).                          .
*.  R12(RAT)= @ VWXTABL CSECT (AVWXTABL DSECT).                       .
*.                                                                    .
*.  AVWXTABL: HAS BEEN COMPLETELY INITIALIZED BY MAIN PROGRAM ASSIST. .
*.       THIS PERMITS REINTA TO MODIFY ASSEMBLER CONTROL FLAGS IF     .
*.       NEEDED TO MAKE ASSEMBLER PERFORM REQUIRED ACTIONS.           .
*.       USES DSECTS: AJOBCON,AVWXTABL                                .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

*+--> INSUB: REREAL     REPLACE REAL ADCONS IN VWXTABL IF NOT THERE.  +

**--> INSUB: REREGS     FORMAT PARAMETER REGS AND PRINT THEM  + + + + +

**--> ENTRY: RESYMB     ENTER CODE IN SYMBOL TABLE OF CALLABLE ENTRY. .
*.       RESYMB IS CALLED FROM CVCON2 IF A SYMBOL FLAGGED EXTRN IS    .
*.  USED IN A VCON.  IT PLACES A CODE INTO THE SYVALUE ENTRY OF THE   .
*.  SYMBOLS SYMSECT.  THIS CODE (THE OFFSET TO A CALLABLE ENTRY       .
*.  ELEMENT IN THE SECOND SECTION OF RFSYMS), IS USED FOR CHECKING    .
*.  WHEN THE USER PROGRAM ACTUALLY CALLS THE ROUTINE.                 .
*.       ENTRY CONDITIONS                                             .
*.  RA = @ SYMSECT FOR THE EXTRN SYMBOL.                              .
*.  ALL OTHER REGS: SAME AS ASSEMBLER REGISTER CONVENTIONS.           .
*.       EXIT CONDITIONS                                              .
*.  RA = @ SAME SYMSECT, BUT CODE HAS BEEN ENTERED IN SYVALUE.        .
*.  RB = 0     IF SYMBOL WAS LEGITAMATE.                              .
*.     = 4     IF SYMBOL WAS NOT LEGITAMETE ENTRY TO BE CALLED.       .
*.       NAMES: RES-----                                              .
*.       USES DSECTS: RFSYMBLK,SYMSECT                                .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> INSUB: REXCON3    CONVERT 3 BYTES OF REGISTER R7 TO HEX.  + + + +

**--> INSUB: REXPRINT   PRINT MESSAGE + + + + + + + + + + + + + + + + +

**--> INSUB: RGENTS     CHECK USER VALUES IN PARAMETER REGISTERS+ + + +

**--> INSUB: RGRAADDR   CHECK LEGITAMACY OF SCAN PTR RA + + + + + + + +

**--> INSUB: RGRCADDR   CHECK RC FOR @ INSIDE USER PROG.+ + + + + + + +

**--> INSUB: RHENTS     CHECK PARM REGS PASSED TO CALLED PROGRAM+ + + +

**--> INSUB: RHRAADDR   CHECK RA FOR REASONABLE @ + + + + + + + + + + +
**--> CSECT: RFSYSMS    TABLE OF CSECT-ENTRY NAMES-REPLACE  . . . . . .
*.             RFYSMS (SECT.1) HAS AN ELEMENT FOR EACH CSECT WHICH CAN.
*.       BE DYNAMMICALLY REPLACED BY A USER-WRITTEN ROUTINE. EACH     .
*.       ELEMENT CONTAINS THE NAME OF THE CSECT, THE NUMBER OF        .
*.       ENTRY POINTS IN IT, AND A LIST OF ENTRY POINT NAMES AND      .
*.       OFFSETS TO THEIR ADCONS IN AVWXTABL, SO THEY CAN BE CHANGED. .
*.             THE 2ND SECTION IS PRESENT IF &$REPL=2.  IT LISTS ALL  .
*.       ENTRYPOINTS WHICH CAN BE CALLED FROMA USER PROGRAM, WITH     .
*.       OFFSET @ PTRS TO THEIR ADCONS IN AVWXTABL, AND TO CODE IN    .
*.       SECTION RHENTS OF REMONI.  THIS CODE IS USED TO CHECK THE    .
*.       REGISTERS PASSED BY THE USER TO THE CALLED PROGRAM.          .
*.             THE 3RD SECTION IS ALSO PRESENT ONLY IF &$REPL=2. IT   .
*.       HAS LABELS OF THE FORM RI&CSECT, WITH &CSECT BEING ONE WHICH .
*.       NOT ONLY CAN BE REPLACED, BUT CAN ALSO CALL OTHER ROUTINES.  .
*.       EACH ELEMNT CONTAINS A HALFWORD WITH THE NUMBER OF DIFFERENT .
*.       SUBROUTINE ENTRIES WHICH THIS CSECT IS PERMITTED TO CALL,    .
*.       FOLOWED BY THAT # OFFSET VALUES TO THE ELEMENTS IN THE 2ND   .
*.       SECTION OF THOSE ENTRIES IT CAN CALL.  REMONI OBTAINS AN     .
*.       OFFSET FROM RFSYMS TO RI&CSECT FROM THE RFSYMBLK BELONGING   .
*.       TO THAT CSECT.  NOTE, IF A CSECT CAN CALL NO OTHER, THE      .
*.       VALUE SAVED IS = 0.                                          .
*.       NAMES: RF------                                              .
*.       NAMES: RI------           (IN SECTION 3, IF &$REPL=2)
*.       DSECT RFSYMBLK IS USED TO DESCRIBE EACH ENTRY IN SECTS.1&2.  .
*.       USES MACROS: $AL2,RFSGN                                      .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> CSECT: SCANRS   1-2 SCANNING ROUTINES . . . . . . . . . . . . . . .
*.       SCANRS CONTAINS VARIOUS UTILITY SCANNING ROUTINES. ALL 3      .
*.       ENTRIES TERMINATE SCANNING ON FINDING A BLANK. 1 ENTRY ALSO  .
*.       STOPS FOR A COMMA, AND THE OTHER STOPS FOR AN EQUALS SIGN.   .
*.       ****NOTE**** THIS ROUTINE MODIFIES TABLE AWTZTAB IN AVWXTABL..
*.       IT MAY THEN CALL SDBCDX WITHOUT RESETTING THE TABLE. THIS    .
*.       IS AN EXCEPTION TO THE RULE OF NOT PERMITTING MODIFICATION   .
*.       TO AV------ SECTIONS WHEN CALLING ANOTHER MODULE.            .
*.       CALLS SDBCDX                                                 .
*.       USES DSECTS: AVWXTABL                                        .
*.       USES MACROS: $CALL,$RETURN,$SAVE,$SETRT                      .
*.       NAMES: SCAN----                                              .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: SCANBL       SCAN TO BLANK ONLY. . . . . . . . . . . . . .
*.       ENTRY AND EXIT CONDITIONS SAME AS SCANEQ                     .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: SCANCO       SCAN TO COMMA OR BLANK (USED BY A-TYPE ADCON.
*.       ENTRY AND EXIT CONDITIONS SAME AS SCANEQ                     .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: SCANEQ       SCAN TO = OR BLANK(USED BY IAMOP1 FOR LITERA.
*.       ENTRY CONDITIONS                                             .
*.  RA = SCAN POINTER                                                 .
*.       EXIT CONDITIONS                                              .
*.  RA = SCAN POINTER TO = OR BLANK, OR ERROR IF ANY                  .
*.  RB = 0    IF SCAN OK, = ERROR CODE IF ERROR FOUND(IN SELF-DEF TRM).
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> CSECT: SDTERM     SELF-DEFINING TERM CONVERSIONS. . . . . . . . .
*.       SDTERM INCLUDES AN ENTRY POINT FOR CONVERTING EACH TYPE OF   .
*.       SELF-DEFINING TERM, AND AN ENTRY POINT WHICH FIRST DECIDES   .
*.       WHICH TYPE(IF ANY) THE SCAN POINTER IS POINTING AT, THEN     .
*.       BRANCHES TO THE CORRECT SECTION TO CONVERT THE TERM.         .
*.       USES DSECTS: AVWXTABL                                        .
*.       USES MACROS: $RETURN,$SAVE                                   .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: SDBCDX   1-2 DETERMINE TYPE OF SELF-DEFINING TERM-CHECK. .
*.       DECIDE TYPE OF SELF-DEFINING TERM, BRANCH TO RIGHT SECTION.  .
*.       ENTRY CONDITIONS                                             .
*.  RA = SCAN POINTER TO BEGINNING OF TERM- TO C,B,X, OR 1ST DIGIT    .
*.       EXIT CONDITIONS                                              .
*.  RA = SCAN POINTER TO DELIMITER BEYOND TERM,(NOT ' ENDING B,C,X)   .
*.  RB = 0    SELF DEFINING TERM WAS LEGAL                            .
*.  EB = >0 - ERROR CODE - ILLEGAL TERM ($ERSDINV)                    .
*.  RB = -4 ==> SCAN POINTER DID NOT POINT AT SELF-DEFINING TERM      .
*.  RC = VALUE OF SELF-DEFINING TERM, FROM 0 TO 2**24-1               .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: SDBTRM   1-2 SCAN, COMPUTE BINARY SELF-DEFINING TERM . . .
*.       ENTRY,EXIT CONDITONS SAME AS SDBCDX, EXCEPT RB >= 0 ON EXIT. .
*.       NAMES: SDB-----                                              .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: SDCTRM   1-2 SCAN, COMPUTE CHARACTER SELF-DEFINING TERM. .
*.       ENTRY,EXIT CONDITONS SAME AS SDBCDX, EXCEPT RB >= 0 ON EXIT. .
*.       NAMES: SDC-----                                              .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: SDDTRM   1-2 CHECK OR CONVERT DECIMAL SELF-DEFINING TERM .
*.       ENTRY,EXIT CONDITONS SAME AS SDBCDX, EXCEPT RB >= 0 ON EXIT. .
*.       NAMES: SDD-----                                              .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: SDXTRM   1-2 SCAN, COMPUTE HEXADECIMAL SELF-DEFINING TERM.
*.       ENTRY,EXIT CONDITONS SAME AS SDBCDX, EXCEPT RB >= 0 ON EXIT. .
*.       NAMES: SDX-----                                              .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> CSECT: SYMOPS   1-2 ALL NORMAL SYMBOL TABLE OPERATIONS. . . . . .
*.       SYMOPS BUILDS, MAINTAINS, AND RETRIEVES FROM THE SYMBOL       .
*.       TABLE OF THE ASSIST ASSEMBLER.  THE SYMBOL TABLE IS A VIRUTAL.
*.       SCATTER TABLE, WITH CHAIN ORDERING BY A SECONDARY HASH CODE. .
*.       ALL SYMBOLS ARE HASHED INTO A SMALL PRIMARY POINTER TABLE.   .
*.       EACH WORD IN THE PRIMARY TABLE POINTS TO A LINKED LIST OF    .
*.       SYMBOLS HASHING TO THAT LOCATION IN THE PRIMARY TABLE. THE   .
*.       SYMBOLS ARE ORDERED ON THE LIST IN DESCENDING ORDER BY THE   .
*.       VALUE OF A SECOND HASH CODE, WHICH IS KEPT IN THE LINK       .
*.       POINTER POINTING TO THE SYMBOL TO WHICH IT BELONGS.  THIS    .
*.       METHOD IS USED BECAUSE MAKES NO ASSUMPTIONS ABOUT THE FINAL  .
*.       SIZE OF THE FINAL SYMBOL TABLE, PERMITTING ALLOCATION OF     .
*.       ENTRIES FROM THE DYNAMIC AREA.  IT ALSO PERMITS A VERY FAST  .
*.       (3 FAST INSTRUCTIONS) MAJOR SEARCH LOOP, WHICH STILL GIVES   .
*.       GOOD PERFORMACNE EVEN WITH A SMALL INITIAL POINTER TABLE     .
*.       AND LONG LISTS OF SYMBOLS.                                   .
*.       CALLS MOSTOP                                                 .
*.       USES DSECTS: AVWXTABL,SYMSECT                                .
*.       USES MACROS: $ALLOCH,$CALL,$RETURN,$SAVE                     .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: SYEND2   2   CLEANUP AT END OF PASS 2. . . . . . . . . . .
*.       *** FUTURE USE - WILL COMPUTE SYMBOL TABLE STATISTICS.       .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: SYENT1   1   ENTER A SYMBOL INTO TABLE,RETURN ADDRESS. . .
*.       ENTRY CONDITIONS                                             .
*.  RA = SCAN POINTER TO FIRST CHARACTER OF THE SYMBOL                .
*.  RB = NUMBER OF CHARACTERS IN THE SYMBOL  =  1 - 8                 .
*.       EXIT CONDITIONS                                              .
*.  RA = ADDRESS IN THE SYMBOL TABLE WHERE SYMBOL IS                  .
*.  RB = 0    THE SYMBOL WAS ALREADY PRESENT IN THE TABLE             .
*.     = 4    THE SYMBOL WAS NOT ALREADY PRESENT IN THE TABLE         .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: SYFIND   1-2 LOOK UP SYMBOL,REPORT PRESENCE/ADDRESS. . . .
*.       ENTRY CONDITIONS                                             .
*.  RA = SCAN POINTER TO FIRST CHARACTER OF THE SYMBOL                .
*.  RB = NUMBER OF CHARACTERS IN THE SYMBOL  =  1 - 8                 .
*.       EXIT CONDITIONS                                              .
*.  RA = ADDRESS OF THE SYMBOL IN THE SYMBOL TABLE, IF IT IS THERE    .
*.  RB = 0    THE SYMBOL IS IN THE TABLE                              .
*.     = 4    THE SYMBOL IS NOT IN THE TABLE                          .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: SYINT1   1   INITIALIZE SYMBOL TABLE . . . . . . . . . . .
*.       OBTAINS SPACE FOR INITIAL POINTER TABLE, ZEROES IT.          .
*.       ALSO SAVES THE ADDRESS OF THE INITIAL POINTER TABLE.         .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> CSECT: UTOPRS   1-2 UTILITY DATA SET ROUTINES . . . . . . . . . .
*.             THIS MODULE PERFORMS ALL THE HANDLING WHICH WOULD      .
*.       NORMALLY BE DONE USING SECONDARY STORAGE FOR INTERMEDIATE    .
*.       SOURCE RECORDS AND FOR OBJECT CODE.  IT USES THE LOWER END   .
*.       OF THE DYNAMIC CORE AREA TO STORE THE RECORD BLOCKS (RSBLOCK,.
*.       RSCBLK,REBLK) RESULTING FROM THE SOURCE PROGRAM, AND PLACING .
*.       THEM DURING PASS 1 SO THAT THE OBJECT CODE CAN BE OVERLAID   .
*.       INTO THE SAME AREA.  I.E. IN NO CASE WILL THE RECORDS BLOCKS .
*.       FOR A SOURCE STATEMENT BE PLACED NEARER THE BEGINNING OF THE .
*.       AREA THAN THE OBJECT CODE RESULTING FROM THE STATEMENT.      .
*.                                                                    .
*.       CODE FOR THIS MODULE DEPENDS HEAVILY ON &$DISKU, WHICH       .
*.       CAN ALLOW UTOPRS TO USE DISK FOR INTERMEDIATE STORAGE.       .
*.       &$DISKU = 0 ==> EVERYTHING IN CORE (NORMAL ASSIST).          .
*.       &$DISKU = 1 ==> USER HAS INCROEE/DISK OPTION (DISKU,NODISKU) .
*.       &$DISKU = 2 ==> ALWAYS GO TO DISK, NO INCORE CODE EXISTS.    .
*.                                                                    .
*.       USES MACROS: $DISK,$RETURN,$SAVE                             .
*.       CALLS XXXXDKOP,XXXXDKRD,XXXXDKE1,XXXXDKWT                    .
*.       USES DSECTS: AVWXTABL                                        .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: UTEND1   1   END PASS 1, PREPARE FOR PASS 2 OF ASSEMBLER .
*.       UTEND1 RESETS CORE POINTERS AND CALCULATES RELOCATION FACTOR..
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: UTEND2   2   CLEANUP AFTER PHASE 2 DONE. . . . . . . . . .
*.       UTEND2 IS CALLED AT THE END OF ASSEMBLY PASS 2.  IT ASSURES  .
*.       THAT ANY DS STATEMENTS ENDING THE PROGRAM WILL BE FILLED IN  .
*.       WITH 5'S, LIKE ANY OTHER DS'S FOLLOWED BY CODE (THE VERY LAST.
*.       STRING OF DS'S MAY NOT BE CAUGHT BY UTPUT2).  IT DOES THIS BY.
*.       CALLING UTPUT2 WITH SOME NONEXISTENT OBJECT CODE.            .
*.       CALLS UTPUT2                                                 .
*.       USES DSECTS: AVWXTABL                                        .
*.       USES MACROS: $SAVE                                           .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: UTGET2   2   GET FROM UTILITY DUIRNG PASS 2. . . . . . . .
*.       UTGET2 IS CALLED DURING PASS 2 TO RETRIEVE THE ADDRESSES OF  .
*.       THE SET OF RECORD BLOCKS BELONGING TO THE NEXT STATEMENT. A  .
*.       CHECK IS REQUIRED FOR ANY OFFSET ADJUSTMENT MADE BY UTPUT1,  .
*.       WHICH MADE SURE THAT NO RECORD BLOCK COULD BE OVERLAID BY    .
*.       ITS OWN CODE.
*.       EXIT CONDITIONS                                              .
*.  RC = @ RSBLOCK (THE ONLY BLOCK DEFINITELY PRESENT).               .
*.  RE = 0    NORMAL RETURN.      RE = 4 ==> END-FO-FILE-QUIT         .
*.  AVRSBPT,AVRCBPT,AVRSCPT NOW POINT TO THEIR BLOCKS, IF THEY EXIST. .
*.  AVREBLK HAS HAD THE REBLK MOVED INTO IT, IF THERE WAS ONE.        .
*.  AVREBPT IS NOT CHANGED, STILL POINTS AT AVREBLK, AS ALWAYS.       .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: UTINT1   1   INITIALIZE UTILITY ROUTINES . . . . . . . . .
*.       INITIALIZES UT POINTER TO BEGINNING OF RECORD BLOCK AREA.    .
*.       USES DSECTS: AVWXTABL                                        .
*.       USES MACROS: $RETURN,$SAVE                                   .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> INSUB: UTPMOVE    MOVE 1 RECORD BLOCK INTO DYNAMIC AREA + + + + +

**--> ENTRY: UTPUT1   1   WRITE TO UTILITY DURING PASS 1. . . . . . . .
*.       UTPUT1 MOVES ALL EXISISTING RECORD BLOCKS FOR A STATEMENT    .
*.       INTO THE LOW END OF THE DYNAMIC CORE AREA, AT THE END OF     .
*.       PROCESSING EACH STATEMENT DURING PASS 1.  THE BLOCKS ARE     .
*.       NEVER PLACED CLOSER TO THE BEGINNING OF THE RECORD BLOCK     .
*.       AREA THAN ANY OBJECT CODE WHICH COULD BE PRODUCED BY THE     .
*.       STATEMENT.  THIS MAKES IT SAFE IN PASS 2 TO JUST MOVE        .
*.       OBJECT CODE INTO THE SAME OVERALL AREA, WITH NO FEAR OF      .
*.       OVERWRITING RECORD BLOCKS STILL NEEDED FOR THE SAME OR       .
*.       LATER STATEMENTS.  THE BLOCKS ARE PLACED IN THIS ORDER:      .
*.       RSBLOCK, (RCODBLK), (REBLK), (RSCBLK)   WITH THE BLOCKS      .
*.       IN ( ) PLACED IF THEY EXIST.  **NOTE** BLOCKS RSBLOCK AND    .
*.       RCODBLK ARE ALWAYS ALIGNED TO FULLWORD BOUNDARY.             .
*.       CALLS MOSTOP                                                 .
*.       USES DSECTS: AVWXTABL,RSBLOCK                                .
*.       USES MACROS: $ALIGR,$CALL,$GLOC,$RETURN,$SAVE                .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: UTPUT2       PRODUCES AND RELOCATES OBJECT CODE. . . . . .
*.       UTPUT2 MOVES OBJECT CODE PRODUCED BY THE ASSEMBLER INTO IT   .
*.       PROPER LOCATION IN THE OBJECT PROGRAM, APPLYING DUPLICATION  .
*.       FACTOR AT THIS TIME, IF NECESSARY.  BECAUSE OF THE WAY THE   .
*.       ASSIST INTERPRETER EXECUT WORKS, AND BECAUSE OF THE PSEUDO   .
*.       START CARD USED BY THE REPLACE MONITOR, NO RELOCATION NEED   .
*.       EVER BE DONE BY THIS PROGRAM, MAKING IT FAST AND SMALL.  THE .
*.       MODULE ALSO FILLS IN  AREAS OF THE OBJECT PROGRAM HAVING NO  .
*.       CODE WITH CHARACTER 5'S, WHICH HELP REDUCE THE SIZE OF ANY   .
*.       COMPLETION DUMPS, AND AID DEBUGGING  (X'F5F5F5' SHOWS UP     .
*        DISTINCTIVELY IN A DUMP, AND IS NOT A LEGAL INSTRUCTION).    .
*.       ENTRY CONDITIONS                                             .
*.  RA = PROGRAM LOCATION COUNTER OF THE OBJECT CODE                  .
*.  RC = @ ASSEMBLED CODE IN MEMORY                                   .
*.  RD = LENGTH-1 OF OBJECT CODE                                      .
*.  RE = DUPLICATION FACTOR FOR THE CODE - 1 OR GREATER               .
*.       USES DSECTS: AVWXTABL                                        .
*.       USES MACROS: $RETURN,$SAVE                                   .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> CSECT: VWXTABL    MAIN ASSEMBLER COMMUNICATION TABLE. . . . . . .
*.       THIS IS ACTUAL TABLE THAT AVWXTABL DSECT CORREPSONDS TO.     .
*.       SEE AVWXTABL COMMENTS FOR DESCRIPTION.                       .
*.       USES MACROS: WCONG                                           .
*.       NAMES: X------, W------, V------                             .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> CSECT: XDDGET(ENTRY XDDPUT) * * * * * * * * * * * * * * * * * * *
*        XGET - XPUT MONITOR.  USES TABLE XDDTABDE TO CONTROL         *
*        I/O THROUGH USER CALLS TO XGET & XPUT.                       *
*        CALLS $READ,$PRNT,$PNCH,XGET,XPUT MACROES.                   *
*    E.X.                                                             *
*        THE MONITOR WILL NOT PERMIT A USER TO XGET A $READ FILE,     *
*        INSTEAD, THE MONITOR WILL CALL $READ AND THE USER WILL       *
*        NOT KNOW ABOUT IT.                                           *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
**--> CSECT: XDDTABLE * * * * * * * * * * * * * * * * * * * * * * * * *
*       CONTAINS INFORMATION ON EACH FILE FOR THE MONITOR             *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
**--> CSECT: XXDDFINI CLOSES XGET-XPUT FILES* * * * * * * * * * * * * *
*        LIKE XXXXFINI, CALLED AT SAME TIME.                          *
*        BUT CLOSES ONLY THE FILES NANDLED BY XGET-XPUT               *
*                                                                     *
*        SEARCH TABLE 'XDDTABLE FOR FILES THAT ARE OPEN AND ARE HANDLED
*              BY XGET-XPUT.                                          *
*        WHEN FOUND, CLOSE THEM THROUGH XGET-XPUT. BLANK OUT FIRST BYTE
*              OF NAME IN TABLE.  IF NOY PERMANENT, AND NOT OPEN,     *
*              JUST WIPR OOT FIRST BYEE.                              *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
**--> CSECT: XXXXDECI   EXTENDED DECIMAL INPUT CONVERSION MODULE. . . .
*.       XXXXDECI IS CALLED BY MACRO XDECI TO PERFORM SCANNING AND    .
*.       CONVERSION OF DECIMAL STRINGS.                               .
*.       ENTRY CONDITIONS                                             .
*.  R14= ADDRESS OF XDECIB DSECT CREATED BY CALLING XDECI.            .
*.  R15= ENTRY POINT ADDRESS (=V(XXXXDECI)                            .
*.       EXIT CONDITIONS                                              .
*.  XDECIR1,XDECIRV VALUES ARE FILLED IN FOR REGS.                    .
*.  CC   IS SET ACCORDING TO SIGN OF RESULT, OR = 3 IF ERROR.         .
*.       USES DSECTS: XDECIB                                          .
*.       NAMES: XXDI----                                              .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> CSECT: XXXXDECO   EXTENDED DECIMAL OUTPUT CONVERSION MODULE . . .
*.       XXXXDECO IS CALLED BY MACRO XDECO TO CONVERT A REGISTER      .
*.       VALUE TO EDITED DECIMAL, IN A 12-BYTE AREA, WITH SIGN.       .
*.       ENTRY CONDITIONS                                             .
*.  R14= ADDRESS OF XDECOB DSECT CREATED BY XDECO                     .
*.  R15= ENTRY POINT ADDRESS (=V(XXXDECO)                             .
*.       EXIT CONDITIONS                                              .
*.  EDITED 12-BYTE RESULT OF REGISTER ARGUMENT STORED AT ADDRESS ARG. .
*.       USES DSECTS: XDECOB                                          .
*.       NAMES: XXDO----                                              .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> CSECT: XXXXHEXIEXTENDED HEXADECIMAL INPUT CONVERSION MODULE . . .
*.       XXXXHEXI IS CALLED BY MACRO XHEXI TO SCAN THE INPUT STRING   .
*.   AND CONVERT IT TO HEXADECIMAL INPUT.                             .
*.       ENTRY CONDITIONS                                             .
*. R14= ADDRESS OF A STORAGE AREA WITH R14-R1 STORED                  .
*. R15= ENTRY POINT ADDRESS (V(XXXXHEXI))                             .
*. R0= ADDRESS OF STRING TO BE SCANNED.                               .
*.       EXIT CONDITIONS:                                             .
*.  VALUE OF CONVERTED STRING IN STORAGE  AREA POINTED TO BY R14,     .
*. STORED IN 16 PASSED R14 OR IN XHEXINUM.                            .
*. R1= ENDING ADDRESS OF STRING, I.E. FIRST NON-HEXADECIMAL DIGIT.    .
*.  CC SET=3 IF ERROR                                                 .
*.       USES DSECT XHEXIB.                                           .
*.       NAMES: XXHI____                                              .
*. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  .
```

```
*.--> CSECT: XXXXHEXOEXTENDED HEXADECIMAL OUTPUT CONVERSION MODULE . ..
*.       XXXXHEXO IS CALLED BY MACRO XHEXO TO CONVERT A REGISTER VALUE.
*.  TO EDITED HEXADECIMAL IN AN 8-BYTE AREA.                            .
*.       ENTRY CONDITIONS:                                              .
*.  R14= ADDRESS OF SAVEAREA FOR CALLING MACRO                          .
*. R15= ENTRY POINT ADDRESS.                                            .
*.  R0 ADDRESS OF AREA WHERE CONVERTE STRING GOES                       .
*. REGISTER VALUE IN XHEXOREG                                           .
*.       EXIT CONDITIONS:                                               .
*.  8-BYTE CONVERTED NUMBER OF REGISTER ARGUMENT STORED AT ADDRESS      .
*.  ARGUMENT                                                            .
*.       USES DSECT XHEXOB.                                             .
*.    NAMES:XXHO----                                                    .
*. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  .
```

```
**--> CSECT: XXXXIOCO   ASSIST INPUT/OUTPUT CONTROL PROCESSING. . . . .
*              XXXXIOCO CONTAINS ALL ACTUAL INPUT/OUTPUT OPERATIONS.  *
*         XXXXINIT AND XXXXFINI ARE USUALLY CALLED ONCE EACH, TO      *
*         PERFORM INITIALIZATION AND TERMINATION RESPECTIVELY.        *
*         THE ENTRIES XXXXSORC,XXXXREAD,XXXXPNCH,XXXXPRNT ARE CALLED  *
*         TO READ SOURCE CARDS,READ DATA CARDS, PUNCH CARDS, OR PRINT *
*         LINES DURING EXECUTION.  THE DCB'S FOR READ AND PNCH ARE NOT*
*         OPENED UNLESS THEY ARE USED, AND IF USED WITHOUT WORKABLE   *
*         OPEN'S, THEY DEFAULT BACK TO SORC AND PRNT, RESPECTIVELY.   *
*         THESE 4 ENTIRES SHARE A COMMON BASE REGISTER (R13,ALSO @ SAVE*
*         AREA), COMMON VALUES OF R11 (@ AJOBCON) AND R12 ( CONSTANT 1)*
*         COMMON EXIT CODE.  SORC AND READ SHARE SOME COMMON CODE (GET)*
*         AND PNCH AND PRNT SHARE SOME COMMON CODE (PUT).             *
*         THESE ROUTINES ARE DESIGNED TO ACCEPT THE XIOBLOCK SET UP BY*
*.        THE XIONR MACRO($READ,$PRNT,$PNCH,$SORC). LOCATE MODE IS    .
*         USED TO MINIMIZE MOVEMENT OF CARD AND LINE IMAGES.          *
*.        *NOTE* REMOTE OPEN/CLOSE PARM LISTS ARE USED TO SAVE SPACE. .
*.        UNDER A DOS SYSTEM, NO SUCH LIST EXISTS DUE TO THE NON-     .
*.        EXISTENCE OF MACRO EXECUTE FORMS FOR THE CLOSING OF DTF'S   .
*.        USES MACROS:  DCB,DCBD(OS) .OR. DTF--(DOS)  (OVERALL USE)   .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> INSUB: XXFIXUP     UPDATE BCB POINTERS TO NEXT BUFFER + + + + + +

**--> INSUB: XXIOPENO   OPEN OPTIONAL DATA SET, FIX FLAGS + + + + + + +

**--> ENTRY: XXXXDKE1    COMPLETE PASS1 PROCESSING, SET UP FOR PASS 2 .
*.        XXXXDKE1 IS CALLED FROM UTEND1.  XXXXDKE1 WRITES LAST BUFFER .
*.        OR IF NO PREVIOUS WRITES WERE PERFORMED,  PASSES UTGET2 THE .
*.        INITIALE ADDRESS OF THE ONLY BUFFER USED.  IF AT LEAST 1    .
*.        WRITE TO DISK WAS DONE,  XXXXDKE1POINTS THE DISK TO START   .
*.        AND READS N-1 BUFFERS FROM THE DISK AND SETS UP FOR         .
*.        PASS 2 OF THE ASSIST ASSEMBLER.                             .
*.                                                                    .
*.        REGISTER ASSIGNMENTS                                        .
*.              R14-> XIOBLOCK POINTER REGISTER                       .
*.              R15-> TEMP. BASE REGISTER                             .
*.              R2-> COUNTER WORK REGISTER                            .
*.              R3-> DECB POINTER                                     .
*.              R4-> BUFFER POINTER                                   .
*.              R8-> WORK REGISTER                                    .
*.                                                                    .
*.        USES DSECTS:XXIOBLOCK,  AVWXTABL                            .
*.        USES MACROS: READ, WRITE, POINT, CHECK                      .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: XXXXDKOP     INITIALIZES FOR DISK UTILITY RUN . . . . . ..
*.              ALL XXXXDK ENTRIES BY RICHARD FORD, PAUL WEISSER.     .
*.        XXXXDKOPIS CALLED FROM UTINT1 IF THE DISK UTILITY OPTION    .
*.        IS ENABLED.  IT PERFORMS A STANDARD FORM OPEN ON THE DISK   .
*.        UTILITY DCB,  INITIALIZES ANY VARIABLES USED BY THE DISK    .
*.        UTILITY ROUTINES. XXXXDKOP ALSO COMPLETES THE DECB'S CREATED .
*.        FOR BUFFER POOL MANAGEMENT BY FILLING IN THE RESPECTIVE     .
*.        BUFFER ADDRESS. IN BATCH MODE XXXXDKOP RESETS THE DISK DATA .
*.        SET WITH A POINT MACRO INSTRUCTION.                         .
*.                                                                    .
*.                                                                    .
*.       REGISTER ASSIGNMENTS                                         .
*.             R13-> SAVE AREA POINTER                                .
*.             R14-> XIOBLOCK POINTER REGISTER                        .
*.             R15-> TEMP. BASE REGISTER                              .
*.             R0-> HOLDS LOW END POINTER TO BUFFER AREA              .
*.             R1-> WORK REGISTER                                     .
*.             R3-> HOLDS NUMBER OF BUFFERS FOR LOOP CONTROL          .
*.             R7-> BASE REGISTER FOR AVWXTABL                        .
*.                                                                    .
*.       USES MACROS: POINT (OS), POINTS (DOS)                        .
*.       USES DSECTS: AVWXTABL, XXIOBLOCK                             .
*.                                                                    .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: XXXXDKR  RETURN A SET OF RECORD BLOCKS TO UTGET2 . . . . .
*.       XXXXDKRD IS CALLED BY UTGET2 WHEN IT HAS COMPLETED           .
*.       PROCESSING A SET OF RECORD BLOCKS. XXXXDKRD RETURNS THE      .
*.       ADDRESS OF THE NEXT BUFFER TO BE PROCESSD VIA THE BUFFER     .
*.       CONTROL BLOCK AND RE-FILLS THE BUFFER WHICH WAS JUST         .
*.       PROCESSED. WHEN ALL BLOCKS HAVE BEEN READ, XXXXDKRD CON-     .
*.       TINUES TO ACCEPT CALLS UNTIL ALL BUFFERS HAVE BEEN           .
*.       PROCESSED, AT WHICH TIME AN END-OF-FILE INDICATION           .
*.       (CC=1) IS RETURNED.                                          .
*.                                                                    .
*.       REGISTER ASSIGNMENTS                                         .
*.             R13-> BASE REGISTER AND SAVE AREA POINTER              .
*.             R14-> XIOBLOCK POINTER REGISTER                        .
*.             R15-> TEMP. BASE REGISTER                              .
*.             R2-> WORK REGISTER FOR COUNTER                         .
*.             R3-> DECB POINTER                                      .
*.             R4-> BUFFER POINTER                                    .
*.                                                                    .
*.       USES MACROS: READ, CHECK                                     .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: XXXXDKWT     WRITE A FULL BUFFER TO DISK . . . . . . . .
*.       XXXXDKWT IS CALLED FROM UTPUT1 WHEN PASS1 HAS FILLED A       .
*.       BUFFER.  XXXXDKWT WRITEESTHE BUFFER TO DISK AND UPDATES      .
*.       THE BUFFER MANAGEMENT TABLE WHICH RETURNS THE ADDRESS OF     .
*.       THE NEXT AVAILABLE BUFFER TO UTPUT1.                         .
*.                                                                    .
*.       REGISTER ASSIGNMENTS                                         .
*.             R13-> BASE REGISTER AND SAVE AREA POINTER              .
*.             R14-> XIOBLOCK POINTER REGISTER                        .
*.             R15-> TEMP. BASE REGISTER                              .
*.             R3-> POINTER TO CURRENT DECB                           .
*.             R4-> BUFFER POINTER                                    .
*.             R5-> BUFFER LENGTH USED ACCUMULATOR                    .
*.             R6-> POINTER TO OLD DECB                               .
*.                                                                    .
*.       USES DSECTS: AVWXTABL, XXIOBLOCK                             .
*.       USES MACROS: WRITE, CHECK                                    .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: XXXXFINI   CLOSE ALL DCB'S WHICH ARE OPEN. . . . . . . . .
*.       XXXXFINI USES THE OPEN/CLOSE PARM LIST BUILT DURING EXECUTION.
*.       TO CLOSE ALL DCB'S CURRENTLY OPEN.  USES 1 EXECUTE TYPE OPEN..
*.       DOS GENERATIONS HAVE NO OPEN/CLOSE LIST, SO A CHECK MUST BE  .
*.       MADE TO SEE WHICH DCB'S MUST BE CLOSED.                      .
*.       ENTRY CONDITIONS                                             .
*.  R11= @ AJOBCON DUMMY SECTION                                      .
*.       EXIT CONDITIONS                                              .
*.  AJIO-- FLAGS ARE ALL ZEROED OUT.                                  .
*.       USES DSECTS: AJOBCON                                         .
*.       USES MACROS: $RETURN,$SAVE,CLOSE                             .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: XXXXINIT   INITIAL OPEN FOR READER/PRINTER . . . . . . . .
*.       OPENS PRINTER,SOURCE CARD RDR.  INITIALIZES XXIOCPTR, WHICH  .
*.       ALWAYS HAS BEGINNING @ OF OPEN/CLOSE PARM LIST (OS GEN. ONLY).
*.       ENTRY CONDITIONS                                             .
*.  R11= @ AJOBCON DUMMY SECTION                                      .
*.  AJIO-- FLAGS IN AJOBCON ARE ALL ZEROS.                            .
*.       EXIT CONDITIONS                                              .
*.  AJIOSO,AJIOPR FLAGGED WITH AJIOPEN IF DCB'S OPEN)D PROPERLY.      .
*.       USES MACROS: $RETURN,$SAVE,OPEN                              .
*.       USES DSECT: AJOBCON                                          .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: XXXXPNCH   PUNCH A CARD, OPENING IF REQUIRED . . . . . . .
*.       CALLED BY $PNCH MACRO TO PUNCH A CARD (DDNAME FT07F001).  IF .
*.       THE DCB XXPNDCB CANNOT BE OPENED, OR IF NOPUNCH WAS USED IN  .
*.       THE USER PARM FIELD, THE CARD IS PRINTED (DDNAME FT06F001)   .
*.       WITH ' CARD-->' PRECEDING IT TO NOTE USAGE.                  .
*.       ENTRY CONDITIONS - SAME AS ENTRY XXXXREAD                    .
*.       EXIT CONDITIONS                                              .
*.  CC=  0     NORMAL RETURN, CARD WAS PUNCHED OR RPINTED             .
*.  CC=  1     RECORD LIMIT HAS BEEN EXCEEDED, CARD PUNCHED ANYWAY    .
*.       USES DSECTS: AJOBCON,IHADCB,XIOBLOCK                         .
*.       USES MACROS: OPEN,PUT                                        .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: XXXXPRNT   PRINT ONE LINE OF OUTPUT. . . . . . . . . . . .
*.       CALLED BY $PRNT MACRO TO PRINT 1 LINE, USING DDNAME FT06F001..
*.       ENTRY CONDITIONS - SAME AS ENTRY XXXXREAD                    .
*.       EXIT CONDITIONS - SAME AS XXXXPNCH                           .
*.       USES DSECTS: AJOBCON,IHADCB,XIOBLOCK                         .
*.       USES MACROS: PUT                                             .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: XXXXREAD   READ 1 CARD AT USER EXECUTION TIME. . . . . . .
*.       OPENS CARD READER(DDNAME FT05F001) IF NOT ALREADY OPEN, OR   .
*.       USES OPEN READER (DDNAME FT00F001) TO GET 1 CARD, USING THE  .
*.       COMMON CODE SECTION XXIOGET.  IF NODATA WAS SPECIFIED IN THE .
*.       USER PARM FIELD, NO OPEN WILL BE DONE FOR  FT05F001, BUT     .
*.       SYSIN WILL BE USED INSTEAD.  CALLED BY $READ MACRO.          .
*.       ENTRY CONDITIONS                                             .
*.  R0 = @ I/O AREA WHERE DATA TO BE READ/WRITTEN                     .
*.  R14= @ XIOBLOCK CREATED BY THE CALLING XIONR MACRO.               .
*.  R15= ENTRY POINT ADDRESS                                          .
*.       EXIT CONDITIONS                                              .
*.  CC=  0     NORMAL RETURN, CARD WAS READ AND TRANSFERRED TO USER   .
*.  CC=  1     ENDFILE ON READER.  IF ASSIST JCL, SAVED IN AJOBCON.   .
*.       USES MACROS:   GET,OPEN                                      .
*.       USES DSECTS: AJOBCON,XIOBLOCK                                 .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: XXXXSORC   READ A CARD DURING ASSEMBLY TIME. . . . . . . .
*.       CALLED BY MACRO $SORC TO READ CARD FOR ASSEMBLER, USING       .
*.       ALREADY OPEN DCB (DDNAME SYSIN).                              .
*.       ENTRY CONDITIONS - SAME AS THOSE FOR ENTRY XXXXREAD.          .
*.       EXIT CONDITIONS  - SAME AS THOSE FOR ENTRY XXXXREAD.          .
*.       USES DSECTS: AJOBCON,XIOBLOCK                                 .
*.       USES MACROS: GET                                              .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> CSECT: XXXXSNAP   DEBUGGING OUTPUT, COMPLETION DUMP . . . . . . .
*.       THIS MODULE PROVIDES ALL REGISTER AND STORAGE DUMPING FOR    .
*.       DEBUGGING PURPOSES, BOTH FOR INTERNAL ASSIST DEBUGGING, AND  .
*.       FOR USER PROGRAMS DURING EXECUTION.  IT IS CALLED BY THE     .
*.       MACRO XSNAP (XDUMP PSEUDO-INSTRUCTION FOR USER PROGRAMS),    .
*.       AND PRODUCES A USER DUMP OR DEBUGGING OUPUT IF THE CALLING   .
*.       XSNAP SPECIFIED A BINARY VALUE FOR OPERAND T(3).             .
*.       ENTRY CONDITIONS                                             .
*.  SEE XSNAP CONTROL BLOCK AND POINTERS ON ENTRY TO XSNAP COMMENTS.  .
*.  ALSO, IF SPECIAL ASSIST OUTPUT IS DESIRED I.E. T(3) IS USED, THE  .
*.  WORD IN XXSRGSAV WHERE REGISTER R10 WAS SAVED MUST CONTAIN THE    .
*.  ADDRESS OF THE ECONTROL DUMMY SECTION, WHICH SUPPLIES VALUES      .
*.       EXIT CONDITIONS                                              .
*.  ALL REGISTERS AND CONDITION CODE ARE RESTORED TO ORIGINAL VALUES  .
*.  AFTER EXECUTION OF THE INSTRUCTION AT THE RETURN POINT.           .
*.       USES DSECTS: ECONTROL,XXSNAPC                                .
*.       USES MACROS: $PRNT(IF &$DEBUG=1), OPEN,PUT(IF&$DEBUG=0)      .
*.       NAMES: XX------ , ALL NAMES ADDED FOR ASSIST: XXAS----       .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> ENTRY: XXXXSNIN   XXXXSNAP INITIALIZATION ENTRY . . . . . . . . .
*.       CALLED TO INITIALIZE 'XSNAP - CALL' NUMBER TO 1 (IN CASE     .
*.       BATCHED RUNS ARE USED).                                      .
*.       ENTRY CONDITIONS                                             .
*.  R14= RETURN ADDRESS                                               .
*.  R15= @ XXXXSNIN                                                   .
*.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
**--> CSECT: XXXXSPIE       INTERRUPT CONTROL & COMMUNICATIONS  . . . .
*                                  SCOTT A SMITH - FALL 1971.          .
*.       THIS IS CALLED ONLY FROM THE MACRO EXPANSION OF $SPIE.  IT    .
*.       CONTAINS THE ONLY MACROS THAT CAUSE LINKAGE TO BE SET UP      .
*.       BETWEEN THE SUPERVISOR AND THE EXIT ROUTINE FOR INTERRUPT     .
*.       HANDLING.  THE INITIAL COMMUNICATIONS ARE NEVER MADE UNLESS   .
*.       AT LEAST ONE $SPIE IS EXPANDED.  ONLY ONE ACTUAL SUPERVISOR   .
*.       CALL IS NECESSARY.  ALL OTHER $SPIE EXPANSIONS JUST MANI-     .
*.       PULATE THE CONTROL BLOCKS GENERATED BY THAT EXPANSION.        .
*.       **NOTE**  XXXXSPIE CONTAINS THE ONLY OCCURENCES OF THE        .
*.       MACROS  SPIE (OS) OR STXIT (DOS)                              .
*.       NAMES:  XSP-----                                              .
*.                                                                     .
*.       THIS ENTRY HANDLES THE UPDATING OF THE POINTER TO THE         .
*.       ACTIVE XSPIEBLK .                                             .
*.       ENTRY CONDITIONS                                              .
*.  R1 = @ NEWLY CREATED ACTIVE XSPIEBLK (OR RESTORED XSPIEBLK)        .
*.  R14= RETURN ADDRESS                                                .
*.  R15= @ ENTRY POINT                                                 .
*.       EXIT CONDITIONS                                               .
*.  R1 = @ LAST PREVIOUS ACTIVE XSPIEBLK                               .
*.     = 0 , IF NO PREVIOUS XSPIEBLK'S EXISTED                         .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**--> INSUB: XXXXSPEX              INTERRUPT EXIT ROUTINE              *

**--> ENTRY: XXXXSPIN     INITIALIZATION OF INTERRUPT COMMUNICATIONS. .
*.       THE ONLY NECESSARY SPIE(OS) OR STXIT(DOS) IS EXECUTED HERE    .
*.       TO CATCH ALL INTERRUPTS AND TO REQUEST THE RETURN OF CONTROL  .
*.       TO THE SAME EXIT ROUTINE HANDLER.  AS SUBSEQUENT $SPIE'S      .
*.       ARE ISSUED, NO SVC IS NEEDED; JUST AN ANALYSIS OF THE         .
*.       STATUS OF THE ACTIVE CONTROL BLOCK(XSPIEBLK) BY THE COMMON    .
*.       INTERRUPT EXIT ROUTINE.                                       .
*.       USES MACROS: SPIE(OS) OR STXIT(DOS),$SAVE,$RETURN             .
*.       ENTRY CONDITIONS                                              .
*.  R14= RETURN ADDRESS                                                .
*.  R15= @ ENTRY POINT                                                 .
*.. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

APPENDIX VI. INTERNAL DEBUGGING AIDS

ASSIST contains various debugging facilities to be used by any programmer making modifications to the ASSIST source program. A macro is provided for dumping registers and storage, conditional on flags set at execution time. Program tracing may be performed, again depending on execution-time flag settings. Debugging flags may be set by supplying a value to the DEBUG option of the user PARM field, during either pass of the assembly, or during user program execution. This appendix supplies the programmer with the information needed to create a debug version of ASSIST, make use of the debug code already in the ASSIST program, and possibly write additional debugging code of his own.

A. GENERATION OF DEBUG CODE

All debug code currently provided in ASSIST is controlled by a single SET symbol, &$DEBUG, as follows:

&$DEBUG=1	No debugging code is generated anywhere in ASSIST.
&$DEBUG=0	Debugging code is generated in many sections of ASSIST, depending on local requirements.

All debugging output is done using the macro XSNAP, which includes performing tests on data flags before actually producing output. This macro is used by itself in some places, and is also called to perform tracing functions on entry and exit to some subroutines. The usage of this macro is described in a separate writeup. Briefly, it prints labeled output which can show the registers, any number of storage areas, and information such as the PSW where it was called from, without destroying any registers or the condition code. If needed, it can make execution-time tests to determine whether or not output should be printed.

It is suggested that any additional debugging code added to ASSIST should be made conditional on the value of &$DEBUG, or at least upon another SET variable, in order to maintain a source program with large quantities of debugging code in it which can be completely suppressed for creation of a production system.

B. SET SYMBOLS AND MACROS USED IN DEBUGGING

This section describes debugging SET symbols, macros, and the interactions among them.

1. SET SYMBOLS

The following are also listed in Appendix II.

a. &DEBUG

This symbol is set by the macro $DBG, and always contains a character string which is a hexadecimal self-defining term. It is used by the XSNAP macro called by the XSRTR macro in a TM instruction.

b. &TRACE

This symbol gives the type of tracing to be performed at subroutine entry/exit. It is set by $DBG macro, and contains one of the following three values:

```
NO   no trace code will be generated by an XSRTR macro.
*    the XSNAP called by XSRTR will do nothing but print a
     message noting entry/exit of the given routine.
SNAP the XSNAP will not only print the message as above, but
     will also print out the GP registers.
```

2. MACROS

a. $DBG

This macro is called to set the values of &DEBUG and &TRACE, which will be used as set by all $SAVE and $RETURN macro calls, until the next $DBG call is made. In most cases, $DBG is called one time at the beginning of each control section, to set the debugging output desired during that section.

b. $RETURN

This is the extended RETURN macro for exiting from a subroutine. It calls macro XRETURN, supplying certain defaults. XRETURN calls XSRTR macro, which generates any tracing code required.

c. $SAVE

This is an extended SAVE macro, which calls XSAVE macro, which then calls XSRTR to generate any trace code required.

d. XSNAP

This is the primary debugging macro, and is a slightly modified version of the XSNAP macro which can be used by anyone writing in assembler. It is used both by itself, and as an inner macro for XSRTR to generate all debugging code. It generates a call to the module XXXXSNAP to perform any output formatting and printing. All XSNAP calls in a section of code can be nullified by setting the GBLB variable &XSNAPST to 1, with the exception of the XSNAP calls which have a value in the third position of the T= operand. These calls are always generated, since they are used to produce completion dumps or user execution-time debugging dumps.

e. XSRTR

This macro is called to create trace code for $SAVE or $RETURN, and is a modified version of the normal one. No trace code is created if &$DEBUG=1, or if &TRACE=NO. The XSNAP call it creates performs a TM AVDEBUG,&DEBUG , so it is currently used only in the assembler.

C. INDIVIDUAL SECTION DEBUGGING CODE DESCRIPTION

For each major portion of the ASSIST program, this section notes the debug flags used, the methods by which they are set, how they are tested, and every place in each control section where debugging output may be produced. It also notes any other debugging code present. Please recall that none of the code mentioned below is generated if &$DEBUG is set to 1 , i.e., a pure production program. If necessary, &$DEBUG can be changed at various places, in order to produce debug code in some sections but not in others. However, the current version does not do this anywhere.

1. MAIN CONTROL AND SERVICE SUBPROGRAMS

a. FLAG BYTE: AJODEBUG (in AJOBCON dsect)

b. FLAG SETTING: supply the following option in the PARM field on the EXEC card of a user program:

```
DEBUG=decimal#
```

A debug version of APARMS accepts this option, and stores the last byte of the number's value into the byte AJODEBUG for later use.

c. FLAG TESTING: various XSNAP calls perform Test under Mask instructions referencing AJODEBUG, to determine if output should be produced or not, i.e. use operand IF=(AJODEBUG,O,mask,TM) .

d. DEBUG OUTPUT LOCATIONS: the following lists all debug output code in this section of ASSIST, describing locations, test values used, and output produced.

CSECT	MASK VALUE(HEX)	OUTPUT LABEL/LOCATION/OUTPUT PRODUCED/PURPOSE
ASSIST	08	'AFTER TIME/RECORDS SET' immediately after PARM field has been printed, near label ASPNP. prints registers, all of AJOBCON control block This is used to show the status of all overall job control values, after time and records limits have been set, and PARM field scanned.
ASSIST	02	'ECONTROL BEFORE EXECUT' after ECONTROL section has been completely filled in before user program is executed, between labels ASEXECAL and ASDUMPCL. prints registers, all of ECONTROL dummy section. This is used to show complete status of user program ,just before execution.
ASSIST	04	'USER STORAGE BEFORE EXEC(FAKE ADDR)' after ECONTROL section has been completed, just after just previous output code. prints all of user storage, giving the addresses as they are in the assembly listing. This is used to make sure the assembler has done assembly and loading the user program properly.

CSECT	MASK VALUE(DEC)	OUTPUT LABEL/LOCATION/OUTPUT PRODUCED/PURPOSE
APARMS	01	'APMSCAN' after label APMSCAN, i.e., just before scanning of next option in the PARM field is done. prints registers, section of AJOBCON from AJOPARMA to AJIOFLAG, sufficient to get flags set here. This is used to check scanning and conversion code inside APARMS.
APARMS	01	'APFOUND' after label APFOUND, i.e., after an option has bee scanned and found in the table of legal options. prints registers, and the APCBLK of the option found (APCBLK contains the option name and flags). This is used to check table lookup, and make sure all registers are set correctly.

e. ADDITIONAL DEBUG CODE (PRODUCES NO OUTPUT)

The following lists code which is conditional on &$DEBUG, but produces no output. Note that some of this code may be absolutely necessary to allow setting of debug flags.

CSECT	PURPOSE/LOCATION/DESCRIPTION
ASSIST	Zero memory. just before label ASJINIT, after the single large block of memory has been obtained for a workarea. Zeroes workarea, which is useful for debugging, and also minimizes size of any dump caused.
APARMS	Decode DEBUG= PARM option. in individual parameter field analysis section, between labels APADUMP and APAI. stores value given by DEBUG= into AJODEBUG byte. **NOTE** this section is required, since this is the only code which sets flag AJODEBUG.
XXXXSNAP	Produce separate debug mode output. consists of the following sections: just before XXOPENOK: opens a DCB. after XXPRINT: performs PUT of a line. after XXCOUNT: provides separate DCB for XXXXSNAP. The extra code here permits the programmer to obtain internal debugging code on a separate output device if he so desires. If &$DEBUG=1 (i.e., production mode), XXXXSNAP uses the $PRNT macro (XXXXPRNT csect) to produce its output, which means that its output will be interspersed with any output produced by the rest of the system. If this is not desired, a separate DD card and DCB can thus be used to provide separate output.

2. THE ASSEMBLER

a. FLAG BYTE: AVDEBUG (in AVWXTABL dummy section)

b. FLAG SETTING: this flag is given a value when a special opcode is encountered in the user program, during either the first or second pass of the assembly. This opcode is as follows:

```
          DEBUG  pass#,self-defining term
```

pass# is either 1 or 2, signifying during which pass AVDEBUG should be set. It will have no effect on the other pass.
self-defining term is the value to which AVDEBUG is set.

The actual setting of AVDEBUG is done either in IBASM1 or in IDASM2, for pass 1 or 2 respectively, and takes effect immediately. Note that this requires code in IBASM1, IDASM2, and OPCOD1 to be created as debug version modules, as noted in section e.

c. FLAG TESTING: various XSNAP macros perform Test under Mask instructions referencing AVDEBUG. In addition, many of the $SAVE and $RETURN macros used eventually generate XSNAP calls also. In most cases, each csect uses only one or two masks for testing. The XSNAP operand used in this case is IF=(AVDEBUG,O,mask,TM) .

d. DEBUG OUTPUT LOCATIONS: the following lists the control sections of the assembler which may produce debugging output. Two types of entries are given. If a mask and mode are given, this means that each entry and exit point in that csect uses the given mask to test AVDEBUG, and produces the output described (see &DEBUG and &TRACE). If the information given applies only to certain entry/exit points, this is noted, and it is assumed that entries/exits not mentioned print nothing. The second type of description gives the mask used in an XSNAP interior to the control section. The information here is similar to that given in section C.1.d. of this Appendix.

CSECT	MASK(HEX),TRACE MODE	OUTPUT LABEL/LOCATION/OUTPUT PRODUCED/ PURPOSE (for interior XSNAP's)
BROPS2	A0,SNAP	all entries/exits
CACONS	A0,SNAP	all entries/exits
CBCONS	A0,SNAP	all entries/exits
CCCONS	A0,SNAP	all entries/exits
CDECNS	A0,SNAP	all entries/exits
CFHCNS	A0,SNAP	all entries/exits
CNDTL2	A0,SNAP	entry/exit
CODTL1	A0,SNAP	entry/exit

CSECT	MASK(HEX),TRACE MODE	OUTPUT LABEL/LOCATION/OUTPUT PRODUCED/ PURPOSE (for interior XSNAP's)
CPCONS	A0,SNAP	all entries/exits
CVCONS	A0,SNAP	all entries/exits
CXCONS	A0,SNAP	all entries/exits
CZCONS	A0,SNAP	all entries/exits
ERRORS	80,SNAP	(at entry point ERRTAG only-others none)
ESDOPRS	90,*	all entries/exits
EVALUT	B0,SNAP	entry/exit
	B0	'EVCNEXTA' just after label EVCNEXTA prints registers, internal variable storage from EVOPRS to EVALQ, which includes operator, term, and sign code/ id stacks, plus paren count and number of terms remaining allowed. This is used to obtain the complete status of the expression evaluation just after the current row in the transition table has been set, but before the next character(s) are examined.
	B0	'EVDJUMP' just before label EVDJUMP' prints contents of GP registers, address of code section to be executed next. This is done just before a jump is done to an individual processing code section or error routine, depending on the current state(row of transition table) and the type of symbol or delimiter just found.
	B0	'EVFRCOEX' just after label EVFRCOEX in arithmetic computation section. prints GP registers. This shows all results of a single evaluation of two term values and the single operator for them.

CSECT	MASK(HEX),TRACE MODE	OUTPUT LABEL/LOCATION/OUTPUT PRODUCED/ PURPOSE (for interior XSNAP's)
IAMOP1	90,*	entry/exit
IBASM1	90,*	entry/exit
ICMOP2	90,*	entry/exit
IDASM2	90,*	entry/exit
INCARD	90,*	entry/exit
LTOPRS	A0,*	all entries/exit
	84	(no particular label) in section LTDMP1, just after label LTD1E. prints GP registers, contents of 1 literal entry (LTLENTRY). This occurs during pass 1, when the locations are being assigned in order to all literals in the current literal pool. It can be used to make sure sections LTENT1 and LTDMP1 are working correctly.
	84	(no particular label) in middle of section LTGET2. prints registers, contents of the literal table entry (LTLENTRY) which has just been requested by calling program. This is used to make sure that the address of each literal table entry has been saved correctly, and that LTGET2 is retrieving literal information properly.
MOCON1	90,*	entry/exit
	88	(no particular label) between labels MONOLB2 and MOPUT, i.e. just before UTPUT1 is called to save all record blocks for the current source statement. prints two blocks of storage: 12 bytes of the RCODBLK created by either IAMOP1 or IBASM1, and the section of the assembler control table from AVLOCNTR to AVDWORK1. This XSNAP displays the most important pass I variables of the assembler, and also all record blocks before they are saved. As such, it checks many sections of pass I processing code for correctness.

CSECT	MASK(HEX),TRACE MODE	OUTPUT LABEL/LOCATION/OUTPUT PRODUCED/ PURPOSE (for interior XSNAP's)
MPCON0	,NO	no tracing is done at all
MTCON2	90,*	entry/exit
OPCOD1	90,*	entry/exit
OUTPUT	C0,SNAP	entry/exit of OUINT1, entry of OUTPT2
	C0,*	all other entries/exits
	note this module should have some XSNAP's added.	
SCANRS	90,*	all entries/exits
SDTERM	90,SNAP	all entries/exits
SYMOPS	90,*	all entries/exits
UTOPRS	C0,SNAP	exit from UTGET2, entry to UTPUT2
	C0,*	all other entries/exits

e. ADDITIONAL DEBUG CODE (PRODUCES NO OUTPUT)

The following lists code sections of the assembler which only exist if &$DEBUG=0, but produce no output.

CSECT PURPOSE/LOCATION/DESCRIPTION

IBASM1 Provide execution of DEBUG opcode during Pass I.
Code section between sections IBDROP and IBDC. Jump table entry after IBAJUMP in constant area.
The code section saves the pass# used in the DEBUG command, evaluates the self-defining term, and saves it also. If the pass#=1, it stores the value into AVDEBUG.

IDASM2 Provide execution of DEBUG opcode during Pass II.
Code section between sections IDDC and IDDS, jump table entry after IDAJUMP, in constant area.
The code section sets AVDEBUG to the value given, if the DEBUG opcode used a pass# of 2.

INPUT1 Zero record blocks.
Immediately after entry to INCARD.
Zeroes 256 bytes in AVWXTABL which will contain all the record blocks for next statement, except for RCODBLK. This aids debugging of INCARD, and other sections of code which set values in record blocks.

OPCOD1 Provide debug commands.
In opcode table, two places:
after label OP4D (DIAG).
after label OP5D (DEBUG).
These permit the assembler to recognize two special commands for debugging. DEBUG is used to set AVDEBUG in the assembler. DIAG is used to set a flag during user program execution. These are absolutely necessary for doing debug work on the assembler and interpreter, since no other ways exist to set the flags.

ASPLM662-1

3. THE INTERPRETER

a. FLAG BYTE: ECFLAG2 (in ECONTROL dsect)

b. FLAG SETTING: ECFLAG2 is set by executing a Diagnose instruction, i.e., an SI instruction with opcode X'83'. The immediate field of the instruction is moved to ECFLAG2. Note that this opcode can be generated by use of the DIAG mnemonic assembled by a debug version of the assembler (see section C.2.e. of this Appendix), or the opcode can be coded in hexadecimal and obtained without using a debug version of the assembler. However, the flag setting code only exists in a debug version of the interpreter.

c. FLAG TESTING: various XSNAP's use a TM instruction to test the flag, e.g., the XSNAP operand used is IF=(ECFLAG2,O,mask,TM) .

d. DEBUG OUTPUT LOCATIONS: the following lists all debug output locations in the current version of the interpreter.

CSECT	MASK VALUE(HEX)	OUTPUT LABEL/LOCATION/OUTPUT PRODUCED/PURPOSE
EXECUT	20	'SPIE' just after SPIE exit label EXSPIEXT. prints GP registers, 32 bytes of the PIE (Program Interrrupt Element) supplied to the exit routine by the operating system. This is used in debugging the interpreter to make sure that interrupts are handled properly, and are occurring where they should.
	80	'PRIMARY FETCH' after common code to fetch next instruction and do preliminary decoding, i.e., between labels EXFEXENT and EXEXLEN. prints GP registers, contents of new ECSTACKD block for instruction just fetched, and section of ECONTROL from ECFPREGS to ECILIMP, which shows simulated user registers and most execution flag flag variables. This XSNAP essentially displays the complete status of the simulated user machine, with the exception of the user storage area.
	40	'USER AREA' just before label EXEXLEN, i.e., before the instruction which has just been fetched is executed. prints entire contents of user's simulated storage area, with real memory addresses, which are not normally the ones shown on the user assembly listing. This XSNAP is used for debugging instructions which modify memory in any way. For instance, this can be useful if new SVC routines or I/O routines are added to the interpreter.

ASPLM662-2

e. ADDITIONAL DEBUG CODE (PRODUCES NO OUTPUT)

The following sections of code, while not producing output, are required for use in debug mode, and are conditional on &$DEBUG.

```
CSECT          PURPOSE/LOCATION/DESCRIPTION

EXECUT         Test for Diagnose instruction.
                 just after label EX0C1, which is label taken when an
               illegal operation code is discovered.
                 This checks illegal opcode for being X'83'. If it is,
               control passes to EXDIAG, which executes the instruction.

               Execute Diagnose instruction.
                 after section labeled EXSI, i.e., after other SI
               instruction interpretation code.
                 Moves immediate field of the instruction to ECFLAG2.
               Note that this and the code just described make up the
               only way to set ECFLAG2 at the current time, and so are
               required for debugging usage.
```

4. THE REPLACE MONITOR

The Replace Monitor currently contains no embedded debug code.
>< NUMBER SEQ1=67000000,NEW1=67000100,INCR=100,INSERT=YES

ASPLM670-1

APPENDIX VII. SYSTEM RESOURCE REQUIREMENTS, JOB CONTROL LANGUAGE

A. SYSTEM RESOURCE REQUIREMENTS

This section describes the system resource requirements needed to run ASSIST 1.1. Certain of these requirements may be necessary only to supply particular facilities inside ASSIST.

1. MEMORY

a. PROGRAM CODE AND PREALLOCATED DATA AREAS

A full ASSIST supporting all possible options allowed requires less than 64K bytes, and generally would require less, since many options are provided which would not be used at a particular installation. A reasonably useful version can be generated to use 28K bytes or less, if the large options are omitted and &$OPTMS=0 for instance. Under OS/360, it is possible to use OVERLAY techniques to reduce the size to approx 20-22K, although this is not recommended. A list of the preallocated storage and data required by various facilities is listed in part C.

B. I/O MODULES

ASSIST uses GET/PUT locate for unit record devices see Appendix II. &$IOUNIT.

c. DYNAMIC STORAGE AREA

During initialization, ASSIST acquires the largest single block of storage remaining in its region, up to a given limit (512K), and then returns the FREE= value to the operating system. The size of this block determines the size of the largest program which can be run under ASSIST. The limiting factor is the quantity of storage used in this dynamic work area during assembly. For the symbol table, literal constant table, and temporary storage of record blocks between Pass I and Pass II of assembly, an average of 50-90 bytes per statement is required.

d. TOTAL MEMORY

Reasonable programs of several hundred statements should easily run in a total memory space of 64K. For most beginning student programs, 44K should suffice, even with a 30K ASSIST system.

2. INPUT/OUTPUT DEVICES

a. CARD READER or equivalent device.

b. LINE PRINTER or equivalent device. A width of 133 characters (including carriage control) is desirable, but a 121 character width is sufficient for all output except a few headings and messages. Both assembly output and completion dumps require no more than 121 bytes.

c. CARD PUNCH or equivalent device. (optional)

d. INTERMEDIATE DISK UTILITY I/O. Allows the assembly of much larger programs. Set &$BUFNO to 3 or 4 and &$BLEN to 3524 or 7044 (half or full track buffers +4) for most efficient operation.

3. OPERATING SYSTEM SERVICES

ASSIST has been written mainly for use on IBM S/360 computer systems under OS/360. However, the system services dependent on OS/360 have in general been utilized only in a few sections of code inside ASSIST, so that only a few sections of code need be changed in order to run ASSIST under a different operating system. The following sections describe the system services used and what substitutes could be used instead, if necessary.

a. STORAGE ALLOCATION/DEALLOCATION

ASSIST acquires storage using the GETMAIN macro and deallocates it using the FREEMAIN macro. If these or their equivalents are not available, a large workarea on a doubleword boundary may be added at the end of the csect ASSIST, and code in two places in that csect modified to just use this static workarea instead of acquiring it dynamically. (Only 1 each of FREEMAIN and GETMAIN are used).

b. INPUT/OUTPUT SERVICES

ASSIST uses QSAM GET/PUT logic, with the following macros: OPEN, CLOSE, GET, PUT, DCB. All input/output code is contained in the control section XXXXIOCO, so any modifications required can be made there without affecting any other code.

c. TIMING SERVICES

ASSIST may use the STIMER and TTIMER macros, both for computing time interval statistics, and for controlling user program execution time. All such uses are in one section (ASTIMR## - ASTIMER) of the control section ASSIST, and can be changed if needed. If no timing is desired, or if none is available, ASSIST can be generated to use none (i.e., &$TIMER=0). See Appendix VIII for details.

d. PROGRAM INTERRUPT SERVICES

The assembler part of ASSIST (csect MPCON0) uses the SPIE macro to trap certain (rare) interrupts). The interpreter (csect EXECUT) definitely requires the use of a SPIE or equivalent to trap any interrupt occurring during program interpretation, and its program logic depends very much on this service being available. It is a definite requirement that it be possible to trap an interrupt, and be able to change the location where execution is resumed afterwards.

After version 1.2/A1, only 1 SPIE will exist, thus easing conversion.

B. JOB CONTROL LANGUAGE

To use ASSIST as described in the Introductory User's Manual, the following catalogued procedure should be added to the PROCLIB of an OS/360 system:

```
  //DATA EXEC PGM=ASSIST
  //FTO5F001 DD DDNAME=INPUT    (omit if running only BATCH jobs).
  //FT06F001 DD SYSOUT=A
  //FT07F001 DD SYSOUT=B
  //FT08F001  DD  UNIT=SYSDA,SPACE=(CYL,(10,1)),DISP=(NEW,DELETE)
  //STEPLIB  DD DSN=library,UNIT=unit,VOL=SER=vol,DISP=SHR
```

where library, unit, and vol describe the library containing the ASSIST load module, if not located in LINKLIB.

The ddnames above were chosen for compatibility with FORTRAN, and especially with WATFOR. If different ones are desired, the ddnames in the DCB's in the control section XXXXIOCO of ASSIST must be changed. In addition, the BATCH control cards ($JOB, $ENTRY) were chosen for compatibility with WATFIV BATCH cards.

C. OPTIONAL FACILITY STORAGE REQUIREMENTS

The following list gives the approximate storage requirements of a number of the optional facilities provided by ASSIST. If possible, experiments should be done with the distribution version of the ASSIST program to see the effects of the various options allowed, and then any of those not desired can be removed, thus saving time, and probably more important, storage. In some cases, the presence or absence of various options interacts, but the differences in storage are minor.

The options are listed in alphabetical order by the set variable value used to include the given option.

&$CMPRS=1 330 bytes. also requires 4048 bytes in dynamic area if CMPRS option actually used .

&$COMNT>0 300 bytes (approx) (comment counting option COMNT)

&$DATARD=1 250 bytes (allows 2nd card reader for data only).

&$DECSA=1 42 bytes (assembly of decimal instructions)

&$DECK=1 250 bytes (object deck punch - DECK)

&$DISKU 600 bytes (disk utility intermediate storage)

&$FLOTA=1 600 bytes (assembly of floating point instructions)

&$FLOTAX=1 21 bytes (assemble extended floating point instructions, in addition to 600 bytes for &$FLOTA=1)

&$FLOTE=1 150 bytes (execution of floating point instructions)

&$HEXI=1 600 bytes (approx) (allows execution of XHEXI)

&$HEXO=1 160 bytes (approx) (allows exectuion of XHEXO)

&$KP26=1 420 bytes (allows 026 keypunch option - KP=26)

&$MACROS=1 approximately 15-16K(macro processor)

&$OBJIN=1 1100 bytes (object deck loading - OBJIN)

&$PAGE=1	380 bytes (page counting, control)
&$PRIVOP=1	100 bytes + more code to be written (this amount allows privileged operations to be assembled, not executed).
&$PUNCH=1	220 bytes (unless &$DATARD=1, in which case only requires 160 bytes beyond that given above). (allows PUNCH)
&$P370=1	12 bytes (interpret S/370 privileged operation codes, in addition to that for &$PRIVOP)
&$P370A=1	57 bytes (assembly of S/370 privileged operations, in addition to that for &$PRIVOP)
&$RECORD=2	170 bytes beyond other options.
&$RELOC=1	70 bytes (allows RELOC option)
&$REPL=1	3600 bytes (also requires &$RELOC=1)
&$REPL=2	4000 bytes (also requires &$RELOC=1)
&$S370=1	308 bytes (interpret S/370's with 370 hardware)
&$S370=2	788 bytes (interpret S/370's with only 360 instructions)
&$S370A=1	85 bytes (assembly of S/370 non-privileged instructions)
&$TIMER=1	570 bytes (allows T=, TD=, TX= options)
&$TIMER=2	660 bytes (extra flexibility in timing)

NOTE the value of &$OPTMS can also be set, and should be set to 0 if there is a lack of storage, or perhaps 3 in medium cases.

D. USING ASSIST EFFECTIVELY IN DIFFERENT ENVIRONMENTS

ASSIST HAS BEEN DESIGNED TO FACILITATE ITS USAGE IN A NUMBER OF QUITE DIFFERENT ENVIRONMENTS, RANGING FROM RUNNING A BATCH OF SMALL STUDENT PROGRAMS AS THE ONLY PROBLEM PROGRAM IN A SMALL COMPUTER, TO RUNNING LARGE NUMBERS OF RUNS AS ONE OF A GROUP OF PROCESSORS OPERATING UNDER A SWAPPING MONITOR IN A DEDICATED REGION OF A LARGE MACHINE. IN GENERAL, ASSIST CAN BE USED EFFECTIVELY USING VERY MINIMAL RESOURCES (SUCH AS 40K OF STORAGE IN A 65K MACHINE WITH ON-LINE CARD READER AND PRINTER), BUT CAN TAKE ADVANTAGE OF EXTRA SPACE AND I/O DEVICES TO OFFER MANY USEFUL OPTIONAL FACILITIES. THIS SECTION BRIEFLY DESCRIBES SOME OF THE VARIOUS WAYS ASSIST CAN BE RUN, IN ORDER OF INCREASING RESOURCE REQUIREMENTS, FINISHING WITH A DESCRIPTION OF ITS USAGE AT PENN STATE UNIVERSITY AND SOME OF OUR EXPERIENCES WITH IT.

MINIMAL SYSTEM (SUCH AS A 360/30 WITH 64K BYTES, UNDER DOS/360, OR A 360/40 OR /50 RUNNING UNDER OS/360 - PCP). (**NOTE** A DOS VERSION OF ASSIST IS IN PREPARATION AS OF 08/23/71, AND SHOULD BE AVAILABLE IN A FEW MONTHS). FOR THIS CASE, ASSIST SHOULD BE STRIPPED DOWN TO A MINIMAL SIZE PROGRAM WHICH CAN ASSEMBLE AND INTERPRET SMALL STUDENT PROGRAMS, USING 1 CARD READER AND 1 PRINTER, WITH $JOB BATCH CONTROL CARDS. TYPICALLY, A NUMBER OF JOBS WOULD BE COLLECTED, THEN FED TO ASSIST IN A BATCH AT DESIRED INTERVALS. ALL OF THE MORE EXOTIC FACILITIES CAN BE OMITTED, THUS SAVING SPACE. IT MAY EVEN BE ADVISABLE TO OMIT FLOATING POINT INSTRUCTIONS FROM THE ASSEMBLER, IF SPACE IS REALLY CRITICAL.

NORMAL JOB ON MEDIUM TO LARGE SYSTEM. ASSIST MAY SIMPLY BE ADDED TO A SYSTEM LIBRARY, AND USED EITHER TO PROCESS SEPARATELY-SUBMITTED INDIVIDUAL JOBS, OR TO PROCESS COLLECTED BATCHS OF RUNS. IF 100-200K OF STORAGE IS AVAILABLE, ALMOST ANY STUDENT-TYPE PROGRAM CAN BE RUN, AND A NUMBER OF THE OPTIONAL FACILITIES CAN BE INCLUDED, THUS EXTENDING THE RANGE OF PROGRAM TYPES WHICH CAN USE ASSIST. EVEN IF RUN AS A PROCESSOR FOR SINGLE JOBS, ASSIST'S LOW OVERHEAD COMPARED WITH THE SYSTEM ASSEMBLERS CAN SAVE PROCESSING TIME, BESIDES PRINTING MUCH LESS OUTPUT AND GIVING BETTER DIAGNOSTICS.

LARGE SYSTEM: FAST-TURNAROUND PROCESSOR UNDER A SWAPPING MONITOR.

ASSIST'S SMALL SIZE AND HIGH SPEED MAKE IT USEFUL FOR PROCESSING LARGE NUMBERS OF JOBS AS ONE OF SEVERAL PROCESSORS WHICH ARE SWAPPED IN AND OUT OF A DEDICATED REGION IN A LARGE COMPUTER, THUS GIVING STUDENT PROGRAMMERS VERY FAST TURNAROUND AT A LOW COST/RUN. THIS TYPE OF USAGE ACCOUNTS FOR THE BULK OF ASSIST UTILIZATION AT PENN STATE. WE GIVE AN OVERVIEW OF THE SYSTEM AT PENN STATE, AS OF 08/23/71.

THE MAIN COMPUTER SYSTEM USED IS A 360/67, POSSESSING 1 MEGABYTE OF PROCESSOR STORAGE, WITH 2 MEGABYTES OF 8 MICROSECOND CYCLE LCS, WITH 2 2301 DRUMS FOR RESIDENCE OF HEAVILY-USED PROGRAMS (INCLUDING ASSIST), PLUS ADDITIONAL PERIPHERAL GEAR. A SEPARATE 360/50 (512K PROCESSOR CORE, 1 MEGABYTE LCS, SYSTEMS RESIDENCE ON 2319 DISK) IS AVAILABLE AT SOME TIMES FOR THE BATCH TERMINAL HANDLING DESCRIBED BELOW.

BRIEFLY, WHILE RUNS MAY BE SUBMITTED AT THE COMPUTER CENTER ITSELF, THE SYSTEM APPEARS ORIENTED TO THE HANDLING OF REMOTE BATCH TERMINALS. AT LEAST 13 READER/PRINTER/PUNCH TERMINALS (SUCH AS IBM 2780 OR DCS CP-4 TYPE DEVICES) ARE LOCATED EITHER AT THE MAIN CAMPUS OR AT COMMONWEALTH CAMPUSES SPREAD ACROSS THE STATE OF PENNSYLVANIA, IN ADDITION TO SIX 360/20'S OF VARYING CAPACITIES. IN PARTICUALR, FOUR OF THESE ARE DEDICATED TO SHORT-JOB-FAST-TURNAROUND REMOTE STATIONS ON CAMPUS, WHERE STUDENTS SUBMIT THEIR OWN JOBS AND OBTAIN THE OUTPUT ALMOST IMMEDIATELY, GENERALLY BEING HELD UP ONLY BY THE PRINTER SPEED. THE SIZE OF THE JOBS IS LIMITED TO 600 OUTPUT RECORDS AND EITHER 5 SECONDS TOTAL TIME ON THE /67 OR 20 SECONDS ON THE /50.

RUNNING UNDER THE OPERATING SYSTEM (OS-MVT WITH A HIGHLY-MODIFIED VERSION OF HASP) AS ONE LONG JOB IS A SPECIAL MONITOR PROGRAM (THE RPSS MONITOR). THIS PROGRAM, WRITTEN BY ROYCE JONES OF THE PSU COMPUTER CENTER STAFF, ALLOWS EACH JOB SUBMITTED TO BE PROCESSED BY THE APPROPRIATE ONE OF APPROXIMATELY HALF A DOZEN PROGRAMS, WHICH ARE LOADED INTO MEMORY WHEN NEEDED. TYPICALLY 75 TO 80 PER CENT OF THE JOBS ARE RUN BY A PSU-MODIFIED VERSION OF WATERLOO UNIVERSITY'S WATFIV COMPILER, 10 TO 15 PER CENT ARE FOR ASSIST, 5 TO 10 PER CENT FOR CORNELL'S PL/C, AND THE REMAINDER FOR VARIOUS OF THE OTHER PROCESSORS AVAILABLE, WHICH INCLUDE A LARGE STATISTICAL PACKAGE STATPAC, A LISP INTERPRETER, A TURING MACHINE SIMULATOR TUTOR , AND SEVERAL OTHERS. DUE TO THE LOW-OVERHEAD METHOD OF HANDLING PRODUCED, LARGE NUMBERS OF JOBS CAN BE RUN AT RELATIVELY LOW COST PER JOB, AND ENOUGH PROCESSORS ARE OFFERED TO HANDLE A GREAT DEAL OF COMPUTER INSTRUCTION. THE FAST TURNAROUND JOB CATEGORY APPEARS VALUABLE NOT ONLY FOR INSTRUCITON BUT FOR RESEARCH, AND HAS COME TO BE USED QUITE HEAVILY. FOR EXAMPLE, A TOTAL OF 5000-7000 JOBS RUN IN A DAY IS PROBABLY TYPICAL, WITH 8000-9000 OR MORE JOBS OCCURRING DURING BUSY PERIODS. OF THSE, USUALLY ALL BUT ABOUT 2000 PER DAY ARE OF THE FAST-TURNAROUND JOB CATEGORY TYPE.

IN ADDITION TO THE FAST-TURNAROUND MODE OF RUNNING, ASSIST IS OF COURSE AVAILABLE AS A SEPARATE PROGRAM, FOR USE WHEN A PROGRAM EXCEEDS THE LIMITS DESCRIBED ABOVE. IN GENERAL, WE FIND THAT STUDENTS WILL PUT IN MUCH EFFORT TO KEEP USING THE FAST-TURNAROUND FACILITIES, FOR EXAMPLE THEY CANNOT NORMALLY ASSEMBLE AND RUN A 1200 CARD PROGRAM, BUT MANAGE TO DO SO BY ASSEMBLING IT UNDER CMPRS (OR RUNNING TWO ASSEMBLIES, EACH WITH HALF OF THE LISTING OFF), THEN RUNNING THE PROGRAM WITH THE ENTIRE LISTING TURNED OFF, SAVING ALL RECORDS FOR EXECUTION. IT IS THUS STILL POSSIBLE TO RUN A PROGRAM THIS SIZE IN THE GIVEN LIMITS.
IN ADDITION TO ACCOUNTING FOR 95 PER CENT OR MORE OF THE RUNS MADE IN OUR FIRST ASSEMBLER COURSE, ASSIST IS USED FOR A FAIR AMOUNT OF DEBUGGING IN THE SYSTEMS COURSE WHICH FOLLOWS, AND IS ALSO USED IN THE DATA STRUCTURES COURSE WHEN APPROPRIATE. THE SYSTEMS COURSE ALSO MAKES USE OF THE REPLACE MONITOR FACILITY FOR AN ASSIGNMENT OR TWO, AND THE OBJECT DECK LOADER HAS EVEN BEEN USED BY A GRADUATE LEVEL COMPILER COURSE TO EXECUTE OBJECT CODE FROM STUDENT-WRITTEN COMPILERS CREATED USING THE XPL SYSTEM. VARIOUS PROJECTS IN THE CMPSC DEPARTMENT HAVE ALSO MADE USE OF ASSIST, INCLUDING A FAIRLY EXTENSIVE THEOREM-UNEXPECTED) USES HAVE BEEN FOUND FOR ASSIST'S DEBUGGING FEATURES. PROBABLY THE MOST UNUSUAL PROGRAM RUN WAS ONE WHICH REQUIRED COMPLETE ADDRESSING CHECKING, AND WAS THUS RUN UNDER ASSIST, EVEN THOUGH IT THUS RESULTED IN A TOTAL EXECUTION TIME OF 2000 SECONDS ON THE /67.

ASSIST WAS FIRST USED BEGINNING SPRING TERM 1970, AND SINCE THEN HAS PROCESSED AT LEAST 50000 JOBS . THE AVERAGE RUN TIMES FOR THE JOBS FOR THE BEGINNING COMPUTER SCIENCE COURSES HAVE GENERALLY BEEN IN THE RANGE OF 1 TO 2.7 SECONDS OF CPU TIME (67).

APPENDIX VIII. TIME, RECORDS, AND PAGES CONTROL

The ability to set limits on and monitor time usage and output is especially necessary for any program intended for student usage. ASSIST provides a large number of different ways to handle such control, in order to allow system planners the maximum possiblity for tailoring a version of ASSIST to local requirements.

ASSIST can be generated to maintain complete control over the time used in any part of a user's run, over the number of output lines lines printed and cards punched, and over the number of pages printed. Flexibility is provided in three major ways:

SYSTEM GENERATION OPTIONS allow the planner to select only those types of control he desires. Three separate levels of timer control are allowed (&$TIMER=0, 1, 2). In the first case, no timing is done at all, thus saving space and execution time. In this case, the I= option can be used to limit the number of instructions executed by a user program, thus limiting loops at a very low cost in facilities needed. The second option provides complete timer control over each separate phase of a job, and third option adds even more flexibility when running under another control program which is already performing timing checks. In essence, this allows the user to get as much time as he actually has, without requiring him to specify a value.

Two distinct versions of output record control are available, i.e., &$RECORD= 0 or 1, or 2. ASSIST always counts output records and never will exceed the limits given, but the last option allows a function like &$TIMER=2 which permits ASSIST to query the operating system to determine the actual remianing records, again removing a value which must be supplied by the user otherwise.

Page control (&$PAGE=1) permits complete control over the number of pages of output allowed by any phase of a run, and is of course desirable at any installation which performs accounting and control on the number of pages.

MULTIPLE OPTIONS allow flexiblity in specifying the values used to perform control. Each of the three areas has at least 3 parameters associated with it, and are set up in such a way as to allow various effects, depending on the values given.

MULTIPLE OPTION SOURCES permit values to be specified wherever it is most convenient to do so, allowing for both limits and defaults. The parameter analysis routine (APARMS) is completely table-driven, and has many provisions for future modifications and additions. The limit and default values are gathered in one place in the program (ASSIST data area), and can be easily modified to run with different default values or limit values for numerical options (ex: making BATCH the default).

A. TIME, RECORDS, PAGES CONTROL ALGORITHM

(The reader should be familiar with PART III. of the USER'S GUIDE, which describes the effects to the user of the algorithms used).

In addition to the instruction count remaining counter, used by EXECUT while interpreting a user program, up to three other counters are maintained by the program, one for each of the three areas.

A time remaining value is maintained using the hardware timer, and the STIMER and TTIMER macos, or equivalent. The timer is manipulated only in csect ASSIST, in the internal subroutine (INSUB) ASTIMR##, which is composed of a number of processing code sections, one for each point during a job where the timer must be set or tested.

A records remaining count is always kept, regardless of the option used to generate ASSIST. For each phase of the program, it is set by a particular one of several sections of an ASSIST csect INSUB called ASRECL##. It is then decremented by 1 every time a line is printed or a card punched (csect XXXXIOCO - entries XXXXPRNT, XXXXPNCH). If the count becomes negative, the module stops producing output and returns this indication to the calling program (which is sufficient to make any of the existing modules stop their execution at that point).

The page control code actually has two counters of its own. A lines remaining in the current page counter is decremented by XXXXPRNT every time a line is printed, and when it becomes zero, ta pages left counter is decremented. When the latter becomes zero, the same action is taken as when the record count becomes negative. The Number of lines per page is given by the L= parameter, which is set by APARMS. The values of lines remaining and pages remaining are set by sections of ASSIST csect INSUB ASPAGE##.

The actions required at any stage of the program are thus done by appropriate sections of the INSUBs ASTIMR##, ASRECL##, and ASPAGE##, some or all of which may exist, depending on local requirements. In general, at any place in the main control part of csect ASSIST at which a change of status is required, 1 to 3 calls are made to the sections of the three INSUBs having corresponding numbers. The organization lends itself to easy modification, since the value-setting code is definitely isolated and well-marked.

The basic algorithm common to all three areas may now be outlined as follows:

1. COMPUTE VALUES FOR INDIVIDUAL PARAMETERS. This is done as described in PART III of the USER'S GUIDE, and the computation done by APARMS.

2. SET INITIAL LIMIT VALUES CORRESPONDING TO T=, R=, and P=. The limits thus set hold for the entire run (or $JOB run, if BATCH), and cannot be exceeded, regardless of the other values. IF &$TIMER=2, the actual time remaining will be obtained from the operating system, and used if it is less than a user-supplied T= option, or if no T= value was given on either the invoking PARM field or $JOB card. A similar action can be done for record counting if &$RECORD=2. The lines remaining count is set to 0, in order to trigger a new page for the ASSIST header. An STIMER (if available) is set to the given value. **NOTE** these actions are coded as the '04' entries of the INSUBs.

3. PRINT HEADERS AND DO INITIAL PHASE. The ASSIST header is printed, with a PARM field or $JOB card, then either assembly begins or a user object deck is loaded. Any timer runout or output excession quickly terminates processing by the module in control, the main program ASSIST obtains control fairly soon, and either terminates completely or flushes to the next $JOB.

4. SET VALUE FOR USER EXECUTION. Assuming that the user program is permitted to execute, limits are set for it (INSUB entries '16'). First, temporary values are computed, by taking the minimum of the value remaining from the previous setting (T=, etc), and the value specified for execution plus dump (TX=, RX=, PX=). For each existing remaining counter, the value is then set to the temporary value minus the dump value (TD=, RD=, PD=), and execution initiated.

This process in effect allows the user to reserve some portion of his time and output to be saved for a dump, without exceeding either the total limit or limit for execution plus dump together.
If any limit is exceeded during user execution, it is terminated immediately.

5. SET VALUES FOR DUMP LIMITS. After user execution terminates for any reason, the values reserved by the 'D' parameters are added back to the remaining counters. Thus, if a user consumed all of the time up to the temporary limit, he still will have the TD= value saved for a dump. On the other hand, if he still had time left, he will be able to use it for his dump if he needs it. If TD (or RD or PD) =0, he can use all of the 'X' option values for his execution, useful for a debugged program desiring maximum ouput and running time. These actions are accomplished by INSUB sections labeled '20'.

B. RECOMMENDED OPTIONS AND MODIFICATION METHODS

In general, there probably exists some combination of values for the default and limit values for all of the control values which will be satisfactory for a given installation. The following describes a number of possible goals which can be obtained without changing the ASSIST program logic.

TOTAL LIMITS ARE ONLY IMPORTANT VALUES. In this case, set the total limit values to the deisred maximum values, set the default values as desired, then set the 'X' values equal to the corresponding total values (SEE APPENDIX II for the SET variables involved, and the ASSYSGEN writeup for modification procedure). Set the 'D' limit values to the total limit values, and the defaults to appropriate values for the model computer being used. We suggest that reasonable values are those which permit the user to see the first part of a dump (i.e., RD=20-25, PD=1, TD= machine dependent), at least for default values.

ASSEMBLY USAGE UNIMPORTANT, USER EXECUTION TO BE MONITORED. In this case, the total limits and default values should be set to very large values, and only the 'X' and 'D' options given appropriate values. Thus, the total limits are in essence ignored.

INFORMATION AVAILABLE FROM OPERATING SYSTEM (OPTION TYPES 2). In this case, set all the values except the 'D' values very high, since ASSIST will use limits obtained from the operating system.

NO TIMING AVAILABLE (OR DESIRED). In this case, the limit and default settings for I= should be carefully coded, since this will be the only execution limit on the user program (unless it loops while producing output, in which case records limits can stop it).

RUNNING ASSIST UNDER A BATCH MONITOR.   (see section D of APPENDIX VII). In some cases, ASSIST may be desired to be able to run on a system both from normal batch jobs and from specially-submitted runs, usually of limited time and output, and possible running under a swapping control monitor (such as the Penn State Computer Center's RPSS MONITOR or the WATERLOO Computing Centre's XMONITOR). In this case, the limit and default values inside ASSIST should generally be set high, since the particular monitor calling it can pass it whatever additional values are desired. If the extra values are concatenated to any user PARM field, individual user programs cannot escape whatever additional control the monitor chooses to perform, do to the nature of PARM handling module APARMS.

Several of the possible changes are particularly easy to make:

ALLOW USER DUMPS TO BE AS LONG AS DESIRED. Set all the D values to zero, then change the '20' parts of the INSUBs included to place very large values in the remaining counters.

BEGIN USER EXECUTION OR USER DUMP ON NEW PAGE. The entries ASPAGE16 and ASPAGE20 do not reset the lines remaining counter, thus giving the user the benefit of the doubt on partial pages. If these two phases should begin on new pages, the desired section may just reset lines remaining counter (AJOLREM) to zero.