FOREWORD

This User's Guide discusses High Level Assembler Language (HLAL) as an effective programming tool for developing computer software more efficiently. It is intended for all programmers having basic knowledge of OS/360 Assembler Language.

Part I of this guide presents some of the basic ideas of Dr. Harlan Mills on structured programming. However, only those ideas applicable to the RTCC environment have been incorporated in HLAL. Therefore, this section is intended only to provide general background information supportive of the guidelines and detailed formulation of HLAL.

Part II discusses these guidelines in detail and provides a functional description of HLAL components. Included in the discussion are MACRO formats since HLAL makes extensive use of the OS/360 Assembler MACRO facilities and is essentially a MACRO language.

All sections are identified by a one-digit number in the upper righthand corner of the page. In Section 3 of Part II, the appropriate MACRO name has been added. These MACRO definitions are filed alphabetically and are page-numbered within to facilitate updating. Pages in all other sections are numbered consecutively.

Only current documentation is maintained in this book. All previous versions will be deleted as they become obsolete and filed in Records Retention. Any additions or changes to this guide may be directed to Will Taylor at 333-3300, extension 3519.

TABLE OF CONTENTS

PART I. STRUCTURED PROGRAMMING

1. PRECISION PROGRAMMING

1.1 COMPLEXITY AND PRECISION IN PROGRAMMING

The digital computer has introduced a need for highly complex, precisely formulated, logical systems on a scale never before attempted. Systems may be large and highly complex, but if human beings, or even analog components, are intrinsic in them, then various error tolerances are possible, which such components can adjust and compensate for. However, a digital logic system, hardware and software, not only makes the idea of perfect precision possible - - it requires perfect precision for satisfactory operation. This complete intolerance to the slightest error gives programming a new character, unknown previously, in its requirements for precision on a large scale.

The combination of this new requirement for precision, and the commercial demand for computer programming on a broad scale has created many false values and distorted relationships in the past decade. They arise from intense pressure to achieve complex and precision results in a practical way without adequate theoretical foundations. As a result, a great deal of programming today uses people and machines highly inefficiently, as the only means presently known to accomplish a practical end.

It is universally accepted today that programming is an error-prone activity. Any major programming system is presumed to have errors in it. Only the very naive would believe otherwise. The process of debugging programs and systems is a mysterious art. Indeed, more programmer time goes into debugging than into program designing and coding in most large systems. But there is practically no systematic literature on this large undertaking. While a source of constant and deep frustration, such errors are nothing new in programming. They have always been there, from the very first days.

Yet, even though errors in program logic have always been a source of frustration, even for the most careful and meticulous, this may not be necessarily so in the future. Programming is very young as a human activity - - some twenty years old. It has practically no technical foundations yet. Imagine engineering when it was twenty years old. Whether that was in 1620 or 1770, it was not in very good technical shape at that stage either! As technical foundations are developed for programming, its character will undergo radical changes.

Approval	Approval
William M Taylor	S. E. James

We contend here that such a radical change is possible now - - that the techniques and tools are at hand to permit an entirely new level of precision in programming. This new level of precision will be characterized by programs that ordinarily execute properly the very first time they are ever run. But to accomplish that level of precision, programming standards and disciplines will be required of an entirely new scope and depth, as well.

Note, here, the objectives of such precision in programming deal with execution, rather than assembly/compilations. Some improvement may be noticeable in reducing syntax errors, but assemblers/compilers can find syntax errors already. It is the program logic errors at the system level which can be practically eliminated from programming today.

1.2 KEY TECHNICAL PRINCIPLES

There have been, from the beginning of programming activities, certain principles from general systems theory that good programmers have identified and practiced in one way or another. These include developing systems designs from a gross level to more and more detail until the detail of a computer is reached, of dividing a system into modules in such a way that minimal interaction takes place through module interfaces, of creating standard subroutine libraries, and using high level programming languages for the coding process.

Precision in programming will see a reapplication of these classical ideas, such as program modularity and clean interface construction. However, there are also two key principles, which are new in their application to programming, that will play a major role in the implementation and exploitation of these ideas. These principles are based on new mathematical results, one graph-theoretic, one function-theoretic in character.

The first key technical principle is that the control logic of any program can be designed and coded in a highly structured way. In fact, we shall see that arbitrarily large and complex programs can be represented by iterating and nesting a small number of basic and standard control logic structures.

This principle has an analogue in hardware design where it is known that arbitrary logic circuits can be formed out of elementary "and", "or", and "not" gates. This is a standard in engineering so widespread it is taken for granted. But it is based on a theorem in Boolean algebra that arbitrarily complex logic functions can be expressed in terms of "and", "or" and "not" operations. As such, it represents a standard based on a solid theoretical foundation. It does not require add hoc justification, case by case, in actual practice. Rather, it is the burden of a professional engineer to design logic circuits out of these basic components. Otherwise, considerable doubt would arise about his competence as an engineer.

A practical application of this first principle is writing "structured" programs - - e.g. GOTO-free PL/I programs (Dijkstra 1968). In PL/I, the branching control logic can be defined entirely in terms of DO loops, IF-THENELSE and ON statements. The resulting code can be read strictly from top to bottom, typographically, and is much easier understood thereby. It takes more skill and analysis to write such code, but its debugging and maintenance is greatly simplified. Even more importantly, such structured programming can increase a single programmer's span of detailed control and productivity by a large amount. Here as in circuit design, a theoretical result puts the burden on the programmer to produce GOTO-free code, rather than on case by case demonstrations by technical management.

The second key technical principle is that programs can be coded in a schedule that requires no simultaneous interface hypotheses. That is, programs can be coded in such a way that every interface is defined initially and uniquely in the coding process itself, and referred to thereafter only in its previously coded form.

This principle has an analogue in the theory of computable functions. A key point in characterizing a computable function is that its valuation can be accomplished in a sequence of elementary computations, none of which involves solving a simultaneous system of equations. Any program which is to be executed in a computer can be coded in such an execution sequence. And the very fact that the computer evaluates only computable functions means that no interfaces can be defined hypothetically and simultaneously in computation.

In practical application, this second principle leads to "top down" programming where code is generated in an execution precedence form. In this case, programmers write job control code first, then linkage editor code, then source code. The opposite (and typical implementation procedure) is "bottom up" programming, where source modules are written and unit tested to begin with, and later integrated into subsystems and, finally, systems. This latter integration process, in fact, tests the proposed solutions of simultaneous interface problems generated by lower level programming; and the problems of system integration and debugging arise from imperfections of these proposed solutions. In a real sense, the usual system integration and debugging process seeks to solve sets of complex simultaneous interface equations which are created by the very system development process! Top down programming circumvents the integration problem by the coding sequence itself.

1.3 STANDARDS, CREATIVITY AND VARIABILITY

Many reactions to technical standards in programming make a basic confusion between creativity and variability. Programming these days is a highly variable activity. Two programmers may solve the same problem with very different programs. Two engineers asked to design a "half adder" with economical use of gates will be much less variable in their solutions, but, in fact, no less creative than two programmers in a typical programming project. Carried to an extreme, two mathematicians asked to solve a differential equation may use different methods of thinking about problems, but will come up with identical solutions and still be extremely creative in the process.

The present programming process is mostly writing down all the things that have to be done in a given situation. There are many different sequences which can accomplish the same thing in most situations. And this reflects itself in extreme variability. A major problem in programming at the present time is simply not to forget anything - - that is, to handle all possible cases and to invent any intermediate data needed to accomplish the final results. Thus, as long as programming is primarily the job of writing everything down in some order, it is, in fact, highly variable - - but that, in itself, is not creative.

It is possible to be creative in programming and that deals with far more ill-defined questions, such as minimizing the amount of intermediate data required, or the amount of program storage, or the amount of execution time, etc. Finding the deep simplicities in a complicated collection of things to be done is the creativity in programming. Getting a program to run correctly, handle all error conditions, etc., is like getting the ball in all 18 holes on a golf course. If you debug long enough, or hit the ball often enough, you get done. Only nobody asks in the clubhouse, "Did you get the ball in all 18 holes today?"

1.4 CONTROLLING COMPLEXITY THROUGH TECHNICAL STANDARDS

A major purpose in creating new technical standards in programming is to control complexity. Complexity in programming seems sometimes to be a "free commodity". It does not show up in storage or in throughput time, and it always seems to be something that can be dealt with indefinitely at the local level.

In this connection, it is an illuminating digression to recall that 500 years ago, no one knew that air had weight. Just imagine, for example, the frustrations of a water pump manufacturer then, building pumps to draw water out of wells on the "theory" that "nature abhors a vacuum". By tightening up seals, one can raise water higher and higher - - five feet, ten feet, then 15 feet, and so on, until one gets to 28 feet. But then, mysteriously and without seeming reason, no amount of effort avails to go higher. As soon as it is known that air has weight and it is, in fact, the weight of a column of some 28 feet of water, then the frustration clears up right away. Knowing the weight of air allows a better pump design, for example, in multiple stage pumps, if water has to be raised more than 28 feet.

We have a similar situation in programming today. Complexity has a "weight" of some kind, but we do not know what it is. We know more and more from practical experience that complexity will exact its price in a qualitative way, but we cannot yet measure that complexity in operational terms. For example, we are soldom able to intelligently reject a program module because it has "too many units of complexity in it". These units of measure will, in all probability, be in "bits of information". But just how to effect the measurements still requires development and refinement.

Nevertheless, we have qualitative notions of complexity, and standards can be used to control complexity in a qualitative way, whether we can measure them precisely or not. One kind of standard we can use to control complexity is structural, as in the first principle noted above. Then we can require that programs be written in certain structural forms rather than simply arbitrary complex control graphs generated at a programmer's fancy. The technical basis for the standard is to show that arbitrarily complex flowcharts can be reformulated in equivalent terms as highly structured flowcharts which satisfy certain standards.

2. STRUCTURED PROGRAMS

2.1 THE IDEA OF STRUCTURED PROGRAMS

We are interested in writing programs which are highly readable, whose major structural characteristics are given in hierarchical form. In fact, we are interested in writing programs which can be read sequentially in small segments, usually under a page in length, such that each segment can be literally read from top to bottom, with complete assurance that all control paths are visible in the segment under consideration.

There are two main requirements through which we can achieve this goal. The first requirement is GOTO-free code, i.e., the formulation of programs in terms of a few standard and basic control structures, such as IF-THEN-ELSE statements, DO loops, CASE statements, DECISION tables, etc., with no arbitrary jumps between these standard structures. The second requirement is library and macro substitution facilities, so that the segments themselves can be stored under symbolic names in a library and the programming language permits the substitution of any given segment at any point in the program by a macro-like call.

PL/I in OS/360 has both the control logic structures, and the library and macro facilities necessary. Assembler Language in OS/360 has the library and macro facilities available and a few standard macros can furnish the control logic structures required.

We will develop later a theoretical basis for programming without arbitrary jumps (i.e., without GOTO or RETURN statements) using only a set of standard programming figures, such as mentioned above. At the present time, we take such a possibility for granted, and note that any program, whether it be one page or a hundred pages, can be written using only IF-THEN-ELSE and DO loop statements for control logic.

The control logic of a program in a free form language, such as PL/I or PL360, can be displayed typographically, by line formation and indentation conventions. A Syntax-Directed Program Listing - - a formal description for such a set of conventions - - is given in (Mills 1970). Conventions often used are to indent the body of a DO-END block, such as

```
DO  I=J TO K;

    statement 1

    statement 2
      . . .
    statement n

END;
```

and the clauses of IF-THEN-ELSE statements, such as

```
IF  X  >  1  THEN

    statement 1

ELSE

    statement 2.
```

In the latter case, if the statements are themselves DO-END blocks, the DO, END are indented one level, and the statements inside them indented further, such as

```
IF  X  >  1  THEN

    DO;

        statement 1

        statement 2
          . . .
        statement k

    END;
```

```
ELSE
    DO;
        statement k + 1
            . . .
        statement n
    END;
```

In general, DO-END and IF-THEN-ELSE can be nested in each other indefinitely in this way.

2.2 SEGMENT STRUCTURED PROGRAMS

Imagine a hundred page PL/I program written in GOTO-free code. Although it is highly structured, such a program is still not very readable. The extent of a major DO loop may be 50 or 60 pages, or an IF-THEN-ELSE statement take up ten or fifteen pages. There is simply more than the eye can comfortably take in or the mind retain for the purpose of programming.

However, with our imaginary program in this structured form, we can begin a process, which we can repeat over and over until we get the whole program defined. This process is to formulate a one-page skeleton program which represents that hundred page program. We do this by selecting some of the most important lines of code in the original program and then filling in what lies between those lines by names. Each new name will refer to a new segment to be stored in a library and called by a macro facility. In this way, we produce a program segment with something under 50 lines, so that it will fit on one page. This program segment will be a mixture of control statements and macro calls with possibly a few initializing, file, or assignment statements as well.

The programmer must use a sense of proportion and importance in identifying what is the forest and what are the trees out of this hundred page program. It corresponds to writing the "high level flow chart" for the whole program, except that a completely rigorous program segment is written here.

A key aspect of any segment referred to by name is that its control should enter at the top and exit at the bottom, and have no other means of entry or exit from other parts of the program. Thus, when reading a segment name, at any point, the reader can be assured that control will pass through that segment and not otherwise affect the control logic on the page he is reading.

In order to satisfy the segment entry/exit requirement, we need only be sure to include all matching control logic statements on a page. For example, the END to any DO, and the ELSE to any IF ... THEN should be put in the same segment.

For the sake of illustration, this first segment may consist of some 30 control logic statements, such as DO-WHILE's, IF-THEN-ELSE's, perhaps another 10 key initializing statements, and some 10 macro calls. These 10 macro calls may involve something like 10 pages of programming each, although there may be considerable variety among their sizes.

Now we can repeat this process for each of these 10 segments. Again, we want to pick out some 50 control statements, segment names, etc., which best describe the overall character of that program segment and relegate further details to the next level of segments. We continue to repeat the process until we have accounted for all the code in the original program. Our end result is a program, of any size whatsoever, which has been organized into a set of named member segments, each of which can be read from top to bottom without any side effects in control logic, other than what is on that particular page. A programmer can access any level of information about the program, from highly summarized at the upper level segments to complete details in the lower levels.

In our illustration, this one hundred page program may expand into some hundred and fifty separate segments, because (1) the segment names take up a certain amount of space, and (2) the segments, if kept to a page maximum, may average only some two-thirds full on each page. Each page should represent some natural unit of the program, and it may be natural to only fill up half a page in some instances.

In the theoretical development carried out below, it will be apparent that it is possible to structure any given program much more deeply than that called for in maintaining segments to page sizes or less. The additional latitude in expanding a necessary half-dozen lines or so into some fifty, requires programmer creativity and perspective. It formalizes a process that good programmers do well instinctively and poor programmers do not so well. But it also standardizes this process of the selection of major from minor aspects of a program and allows all programmers to operate on a common base.

2.3 CREATING STRUCTURED PROGRAMS

In the preceding section, we assumed that a large size program somehow existed, already written with structured control logic, and discussed how we could conceptually reorganize the identical program in a set of more readable segments. In this section, we observe how we can create such structured programs a segment at a time in a natural way.

We suppose that a program has been well designed and that we are ready to begin coding. We also note a common pitfall in programming is to "lose our cool" - - i.e., begin coding before the design problems have been thought through well enough. In this case, it is easy to compromise a design because code already exists which is not quite right, but "seems to be running correctly"; the result is that the program gets warped around code produced ad hoc. We assume that has not happened here.

Our main point is to observe that the process of coding can take place in practically the same order as the process of extracting code from our imaginary large program in the previous section. That is, armed with a program design, one can write the first segment which serves as a skeleton for the whole program, using segment names, where appropriate, to refer to code that will be written later. In fact, by simply taking the precaution of inserting dummy members into a library with those segment names, one can compile or assemble, and even possibly execute this skeleton program, while the remaining coding is continued. Very often, it makes sense to put a temporary write statement "got to here OK" as a single executable statement in such a dummy member.

Now, the segments at the next level can be written in the same way, referring as appropriate to segments to be later written (also setting up dummy segments as they are named in the library). As each dummy segment becomes filled in with its code in the library, the recompilation of the segment that includes it will automatically produce new updated, expanded versions of the developing program. Problems of syntax and control logic will usually be isolated within the new segments so that debugging and checkout goes correspondingly well with such problems so isolated.

It is clear that the programmer's creativity and sense of proportion can play a large factor in the efficiency of this programming process. The code that goes into earlier sections should be dictated, to some extent, not only by general matters of importance, but also questions of getting executable segments reasonably early in the coding process. For example, if the control logic of a skeleton module depends on certain control variables, their declarations and manipulations may want to be created at fairly high levels in the hierarchy. In this way, the control logic of the skeleton can be executed and debugged, even in the still skeleton program.

Note that several programmers may be engaged in the foregoing activity concurrently. Once the initial skeleton program is written, each programmer could take on a separate segment and work somewhat independently within the structure of an overall program design. The hierarchical structure of the programs contribute to a clean interface between programmers. At any point in the programming, the segments already in existence give a precise and concise framework for fitting in the rest of the work to be done.

2.4 READING STRUCTURED PROGRAMS

Reading programs is as much an art today as writing them. There are as many ways of reading programs as there are programmers. Our objective is to develop a systematic basis for reading, so that the process is as nearly repeatable as possible; that is, so that two programmers would go through nearly the same activity in reading a given program and record the same set of observations about it.

As long as programs are the proverbial "bowls of spaghetti", there is little systematic that can be introduced into reading. It is simply a question of following threads of control and an a priori enumeration of that control is usually not practical. But when programs are structured as described above, then it is, indeed, possible to give a systematic sequence in which reading can be done. This sequence within each segment is strictly from top to bottom, noting, of course, the programming effect of the various figures encountered which cause branching and looping. The sequence between segments has more possible variety. These sequences correspond to alternatives available in conducting a tour through a tree. Systematic tree tours can be easily imagined in top down, bottom up, left to right forms, etc. For example, in a top down tour, one examines first the top node, then the nodes connected to that top node, then each of the nodes connected to the latter nodes, etc., until one has found all the nodes of the tree.

It is likely that both top down and bottom up tours will be useful in reading structured programs. When a programmer is trying to get acquainted with a program it seems that a top down reading sequence will be most instructive, so that the program unfolds much as it does in the writing process. However, when a programmer, or set of programmers, wants to do a thorough job of validating a program through reading, then it appears that a bottom up tour may be an effective way of proceeding. Each segment so read in the bottom up tour can be characterized as a checkpoint in the reading process so that the segments above which call on it will then be verifiable by using checkpoint information on the segments they name.

In this connection, it is important to observe that because of its one-in, one-out control character, a segment induces a change of state in the programming system and transfers control to the next line in the segment naming it. This change of state will be represented in changed data values in two categories of data; those internal to the segment (and therefore of no interest to the segment naming it) and data external to the segment. It is this external data that, when characterized, permits the segment to be read and its effects noted simply by name.

It is also evident that program segments, as we have defined them, are natural units of documentation and specification. In fact, the specification of a segment is the best means of accessing its function at higher levels in the programming system. In this case, a reading checkpoint should contain the assertion that the segment carries out its specification correctly, subject, of course, to its named segments carrying out their specifications correctly as well. Now, if one begins at the bottom, verifies each segment carries out its specification and progresses upward, one can finally arrive at the full program as it has been checkpointed, and an opinion about its correctness.

Note again, as in the programming process, that this reading process can involve several programmers concurrently with the joint results being aggregated at higher levels into a final opinion about the program's correctness. Note also, unlike writing programs which seemingly have to be done by a single programmer, several programmers can be reading the same segments simultaneously to arrive at independent conclusions about their validity.

3. THE STRUCTURED PROGRAMMING PROCESS

3.1 FUNCTIONAL SPECIFICATIONS

We define a functional specification to correspond to the mathematical idea of a function, namely, a mapping of inputs into outputs, without regard to how that mapping is to be accomplished. In practical terms, of course, one has to have some underlying ideas on techniques and algorithms that are possible, in order to write a feasible functional specification. For example, we simply cannot formulate impossible computing processes as functional specifications without any hope of implementing them.

However, the general situation in programming system development is that the functional specifications are rather large and complex, simply to write them down. In illustration, the input and output messages and codes of a large information retrieval system may run to hundreds, or even thousands of pages. Because of this, functional specifications are seldom complete as mathematical descriptions, but nevertheless, the mathematical model is an ideal that we have in mind when we speak of functional specifications.

There is an additional advantage in defining a functional specification to correspond to the idea of a mathematical function. It represents a platform from which several independent alternative algorithmic approaches might be explored, even by different groups for later comparison and selection. It permits parallel efforts to an objective that is independent of the means.

Ordinarily, the development of functional specifications interact with the process of program design to achieve those specifications. In unique, highly specialized systems, program design may have a significant feedback to functional specifications to reflect certain opportunities available in hardware architecture or in a programming technique which the ultimate user can adapt to his needs in the programming system. For example, ultimate users can often view information systems in various, almost equivalent ways. In such cases, a particular indexing system already available may well affect the functional specifications for that user system.

3.2 FUNCTION EXPANSIONS

We have noted above that the top down programming process represents a step by step expansion of a mathematical function into simpler mathematical functions, using BLOCK, IF-THEN-ELSE, DO-WHILE, CASE, or DECISION statements as elementary structural devices. Such a programming process is easy to visualize with these constructs. Given a functional specification to be expanded by one step, we ask the question, "What elementary program statement can be used to expand the function?" The expansion chosen will imply one or more subsequent functional specifications, which arise out of the original specification. These new functional specifications can each be treated exactly as the original functional specification and the same questions posed about them.

As a result, the top down programming process is an expansion of functional specifications to simpler and simpler functions until, finally, statements of the programming language are reached. The beginnings of such a process is shown below, expanding the functional specification "Add member to library". Such a functional specification will require more description, but the breakout into subfunctions by means of programming statements can be accomplished as indicated here.

f = "Add member to library"	(specification)
f = (BLOCK, g, h)	(expansion)
g = "Update library index"	(subspecification)
h = "Add member text to library text"	(subspecification)
g = (IF-THEN-ELSE, p, i, j)	(expansion)
p = "Member name is in index"	(subspecification)
i = "update text pointer"	(subspecification)
j = "Add name and text pointer to index"	(subspecification)

etc.

```
f = IF "Member name is in index" THEN          (restatement of two
        "Update text pointer"                  levels of expansion)
    ELSE
        "Add name and text pointer to index"
        "Add member text to library text"
```

3.3 PROGRAM DESIGN

Good programmers have always organized large programming systems into a succession of subsystems of increasing detail with minimal interconnections between the subsystems. They also identify common subprocessing activities, if present, and formulate these as subroutines to be called throughout the programming system. We follow these ideas, sharpen them in some ways, because of the structured programs we intend to create.

First we make a distinction between subprograms which are created for structuring the system, and subprograms which carry out common low-level processing functions in many places in the system. The latter set of subprograms we isolate first, and append to the programming language itself, just as sine or exponential routines are regarded as part of PL/I or Fortran. These subprograms are documented and considered as part of the language description in which programmers write the programming system. It is natural to make these subprograms completely self-sufficient with respect to data, that is, to use no data from their environment except that passed explicitly in the arguments of their calls, Such subprograms may, in fact, be extensive and have their own private environment, e.g., it is conceivable that a subprogram accessed only by calls with explicit arguments may still access large masses of data in their execution, and that even large masses of data be identified in their argument list. But, nevertheless, the concept of data independence from the rest of the programming system is held.

The other type of subprogram which is used to help structure a system will ordinarily appear only once as a call from some other program. In this case, we use no arguments for the subprogram call, but let the communication between programs be based entirely on data structures that both programs are aware of. Ordinarily, these data structures will be nested to correspond to the nesting structure of the programs themselves, and data scopes will be made as low as possible to localize their range of validity.

The process of program design is much influenced by the structured programs that are to result. For example, in defining a subsystem and the immediate constituents of that subsystem as smaller subsystems, one seeks enough control logic to fill up a page of conventional code, but not so much as to overflow pages. It takes some practice to accomplish this, but after some practice, it becomes easier than it might look to organize an entire programming system into a hierarchy of subsystems which are page-like segments in their final code. If, in the coding process, the coding estimates are greatly missed, some rethinking on the program design should be done and recoding carried out accordingly.

3.4 PROGRAM CODING

At the point in time when one is coding a segment, one has, in top down programming, sufficient information to write that segment correctly from code in higher levels which have already been written in order to reach this point of the coding. It is good practice to verify the code as it is written, for logical consistency, with previous code in terms of definitions, exact names, etc., line by line. Ordinarily, programmers do not imagine this kind of verification is really necessary, and rely on their short-term memories to put together and integrate sections of code written in a non-time-structured way. But there is nothing so sobering as programming in this way, discovering how often the short-term memory fails, and reflecting on how much additional debugging would have been necessary because of these failures. Programming today takes such additional debugging for granted, but it is not a necessary activity.

PART II. HIGH LEVEL ASSEMBLER LANGUAGE

1. CONCEPTS

All frequently used segments of code are generated by MACROs. These include those that are common to all applications and those that fulfill individual requirements of each application. All MACROs are coded such that:

(a) They are self-documenting

(b) They are written to process higher level language type statements

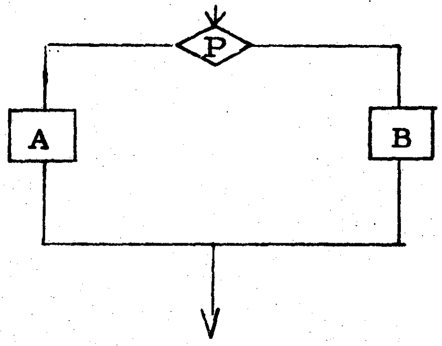
(c) The code that is generated to perform a given function is optimized and debugged when the MACRO is originally written, such that coding errors are reduced, resultant code is more efficient and the function does not have to be redesigned and rewritten each time it is used.

```
IF             BIT,X,IS,ON,THEN
 ~               ~
ELSE
 ~               ~
ENDIF
```

The common set of MACROs contains MACROs that define the beginning and ending block segments used for programming in the structured form.

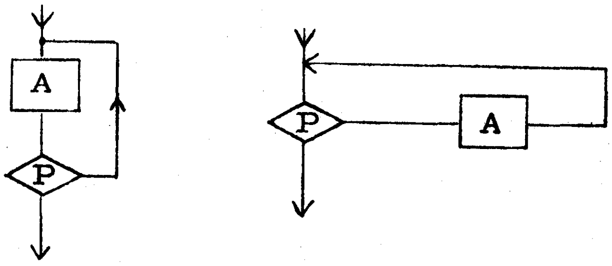
```
IF     P    THEN
       A
ELSE
       B
ENDIF
```

type: dual path decision logic



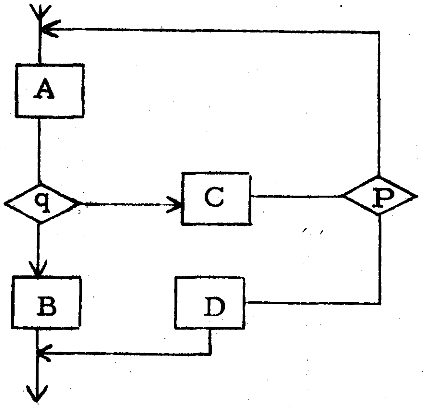
```
{UNTIL}
{WHILE}     P       DO
   A
ENDDO
```

type: looping logic



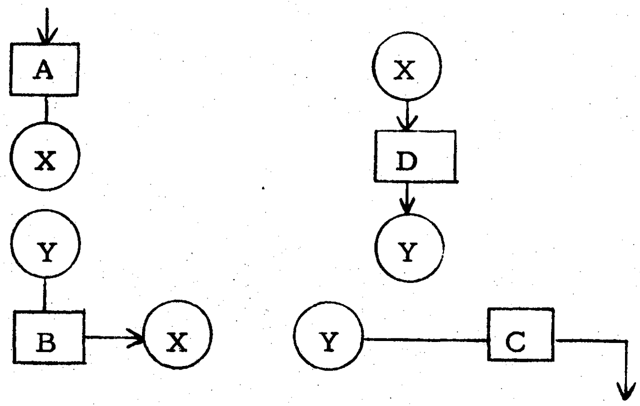
```
STRTSRCH   (UNTIL)     P      DO
           (WHILE)
   A
EXITIF   q
   B
ORELSE
   C
ENDLOOP
   D
ENDSRCH
```

type: table search logic

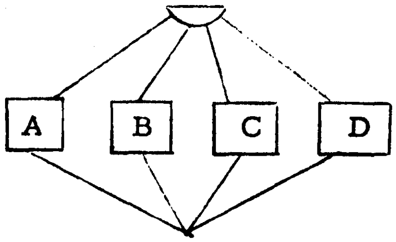


```
A
DO   X
B
DO   X
C
~
BGNSEG   X
D
ENDSEG
```

type: common code



```
CASE      $5,AT=(A,B,C,D)        type:  multiple path decision logic
```



The common set of MACROs also contain MACROs that will perform both the standard logical and mathematical operations.

```
OIBIT      X                          - NIBIT
  }                                   - XIBIT
                                      - TMBIT
X   BIT        0, ON
Y   BYTE
```

After data base is defined, bit manipulation is done without the need of byte masks (X '80')

```
MATH  '((A - B)  *  (C - D))/E = F'
```

mathematical operations.

The individual application set of MACROs will include MACROs which interface with supervisor services.

```
GWORK
RTWRITE
OPEN
```

All application MACROs are tailored to the formats, acronyms and language of that application.

```
GMCECNTL NAME = SING, INTERVL = 5,  CHAIN = LAST
```

This is a specialized GSSC Skylab MACRO for resetting execution interval of a load module.

Except for the guidelines imposed by HLAL any code that can be written in basic Assembler Language can be generated with the block structured MACROs.

IF F, ($5), EQ, ($6), THEN

Register notation in IF MACRO

IF *,, IS, ZERO, THEN

Condition code has already been set.

All frequently used functions too large to expand directly into MACROs are designed and programmed as re-entrant routines which are invoked through tailored interface MACROs.

The set of MACROs needed to program a given area of an application are of such number that the learning time is relatively short.

Some form of block structured listing will be automatically produced each time a program is updated. (Pre- and post-Assembler Processors)

The use of HLAL requires as initial investment effort:

a. Generate the application oriented subset of MACROs (the common set are operational)

b. Educate all application programmers in their use and

c. Define the user data base and all interfaces with DSECTs and labels.

Experienced programmers (2 or 3) with extensive knowledge in the basic Assembler MACRO Language are needed to perform item a.

2. GUIDELINES

These following guidelines will be followed unless they result in gross inefficiencies in code. Any deviations will be discussed with and approved by the designated HLAL coordinator.

a. Should not modify executable code, except for moving a length field into a storage to storage instruction.

b. No conditional or unconditional branching. (The block structured MACROs generate all branching instructions.)

c. No programmer generated labels should be used for branching (the block structured MACROs generate all branching labels).

d. Code in straight forward, readable manner. (Do not get tricky.)

e. Do not use relative addressing (*+8). Do not use absolute displacements 28($5, $6) , use symbolic expressions X - Y ($5, $6) or Y($6) .

f. Reference registers by labels EQUed by HEADC or EQUATE MACROs: $0 - $15 for general purpose registers and FPR0 - FPR6 for floating point registers.

g. Data base and interfaces are referenced by labels defined in DSECTs.

These restrictions cause a programmer to generate straight forward code and avoid some features of basic Assembler Language that usually cause more trouble (excess debugging and non-readability) than they are worth in increased execution time efficiency.

3. MACRO FORMATS, DEFINITIONS AND EXAMPLES

The common HLAL MACROs are sub-divided into ten function groups:

a. Dual-path decision logic:

- IF
- ELSE
- ENDIF

b. Looping logic:

- UNTIL
- WHILE
- BGNWHILE
- ENDDO

c. Error checking logic:

- ERREXIT
- ERRENTER
- ERRMSG
- ERRETURN

d. Table search logic:

- STRTSRCH
- EXITIF
- ORELSE
- ENDLOOP
- ENDSRCH

e. Common code logic:

- DO
- BGNSEG
- ENDSEG

f. Multi-path decision logic:

- CASE

g. Entry, exit logic:

- HEADC
- ENTER
- EQUATE
- GRETURN

h. Bit manipulation:

- NIBIT
- OIBIT
- TMBIT
- XIBIT

i. Mathematical equations:

- MATH
- PRN
- LENGTH
- PARM

$\left.\begin{array}{l}\text{PRN}\\ \text{LENGTH}\\ \text{PARM}\end{array}\right\}$ invoked by MATH

j. Data base definition:

- BIT
- BYTE

The formats, definitions, and examples of the MACROs follow, the MACROs ordered alphabetically. At the end of the MACRO definitions is a one page coding reference sheet for quick referral once a basic understanding of the MACROs is attained.

NAME - BGNSEG

DESCRIPTION

The BGNSEG MACRO generates a label for a section of code to be branched to by the DO MACRO.

The format is:

```
          BGNSEG    SEGMENT,REG
+SEGMENT     DS    OH
```

where SEGMENT is the name of the label to be generated and REG is the register to be used in returning from this segment of code.

When the BGNSEG MACRO follows a DO MACRO which references it, it will use the register specified in the DO macro. If the register is specified in the BGNSEG MACRO and it does not agree with the register specified in the previous DO MACRO, an error message will be written.

When the BGNSEG MACRO precedes any DO MACRO reference to it, the register will default to $14 unless a register is specified.

A maximum of 50 segments may appear in an assembly. Registers need to be expressed in notation $1, $2 etc.

EXAMPLES

Example 1

```
                   BGNSEG        COMPUTE
  +COMPUTE         DS            OH
                   ∫
                   ENDSEG        COMPUTE
  +                BR            $14
```

Example 2

```
                        DO              CODE,$6
    +                   BAL             $6,CODE
                        }
                        BGNSEG          CODE
    +CODE               DS              OH
                        }
                        ENDSEG          CODE
    +                   BR              $6
```

NAME - BGNWHILE

DESCRIPTION

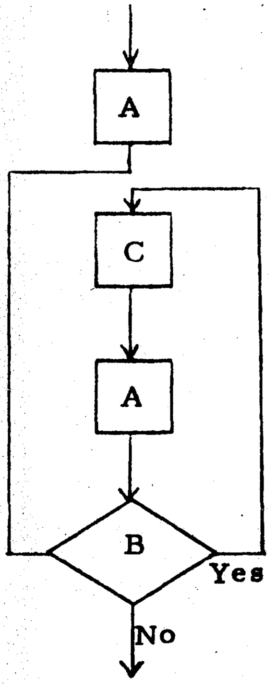
(BGNWHILE (no operands))

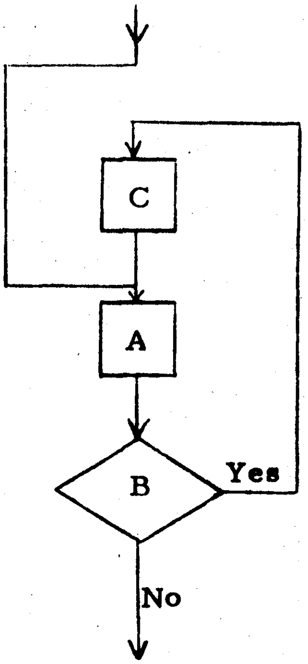
The BGNWHILE macro will cause execution of a WHILE loop to begin at the instruction immediately following the BGNWHILE macro. This macro should be preceded by a WHILE macro and succeeded by an ENDDO macro. Normally, a WHILE loop begins at the ENDDO macro by checking the condition specified in the WHILE macro.

The following example illustrates how a BGNWHILE would be used to start execution of a loop between the WHILE and ENDDO macros.

Instruction Sequence

Without BGNWHILE	With BGNWHILE
$\boxed{A}$	
WHILE ($\boxed{B}$), DO	WHILE ($\boxed{B}$), DO
$\boxed{C}$	$\boxed{C}$
$\boxed{A}$	BGNWHILE
ENDDO	$\boxed{A}$
	ENDDO


A
C
A
B
Yes
No


C
A
B
Yes
No

NAME - BIT

DESCRIPTION

The purpose of the BIT macro is to generate a data base definition whose length can be used as a key to test or manipulate a specific bit in a byte.

DEFINITION

symbol	**BIT**	{**Bit number, or list of bit numbers, or binary 8-bit** configuration} [,ON]

where

- **symbol --** **any valid non-blank label. If omitted, an error condition will be raised with a condition code of 12.**
- **bit number --** **an unsigned decimal integer, 0 through 7, representing standard bit notation.**
- **list of bit numbers -- a list of bit numbers separated by commas. The entire list must be enclosed by parenthesis.**
- **binary 8-bit configuration** -- notation of the form B'XXXXXXXX', **where X is 1 if the corresponding bit is to be** represented by this label and X is 0 if the corresponding bit is not to be represented by this label.
- **ON --** **indicates the bit or bits indicated in the first operand are to set to 1 in a global variable which is passed to the BYTE macro.**

FUNCTION

The BIT macro performs its operations as follows:

- checks to see if there is a valid non-blank label attached to the macro.
- processes the information passed by the first operand, checking each time for an invalid bit number or binary character.
- generates a DS and ORG statement to establish a length which can be used to test or manipulate bit(s), and reset the location counter setting. (There is an exception to this -- if the name of the CSECT currently being processed starts with SCDB, the DS and ORG statement will not be generated.

EXAMPLES OF THE USE

The following are included to give the user a feeling of what can and cannot be done with the BIT macro:

Example 1

NAME	OPERATION	OPERANDS
FIRST	BIT	0
+FIRST	DS	XL(B'10000000')
+	ORG	*-B'10000000'

Example 2

NAME	OPERATION	OPERANDS
SECOND	BIT	(0, 1, 5, 7), ON
+SECOND	DS	XL(B'11000101')
+	ORG	*-B'11000101'

Note: In the above example, specifying 'ON' had no effect upon the expansion of the macro.

Example 3

NAME	OPERATION	OPERAND
THIRD	BIT	B'00111100'
+THIRD	DS	XL(B'00111100')
	ORG	*-B'00111100'

The following examples would raise error conditions:

CODING			CAUSE OF ERROR
NAME	OPERATION	OPERAND	
	BIT	0	name field blank
ONE	BIT	0, 1, 2	operand not enclosed in parentheses
TWO	BIT	8	operand is greater than 7, does not satisfy standard bit notation
THREE	BIT	'01010101'	improper binary notation, should be B'01010101'
FOUR	BIT	,ON	first operand missing

GENERAL NOTES

- All errors detected by the BIT macro will raise a condition code of 12 and result in the termination of processing by the macro. No DS and ORG will be generated unless the operand(s) are valid.
- Specifying 'ON' is used only in conjunction with the BYTE macro. Nothing is gained by the user in using this if the BYTE macro is not also included in his program.

NAME - BYTE

DESCRIPTION

The purpose of the BYTE macro is to generate a data base definition using either information passed from previous calls of the BIT macro or a parameter on the BYTE macro.

DEFINITION

symbol	BYTE	one byte hex value

where

- the operand may be blank, or
- the operand is a value hexadecimal number (range is from 0_{10} to 255_{10}) i.e., X'FF'.

FUNCTION

The BYTE macro performs its operations as follows:

- examines the operand to determine whether or not it is null.
- if the operand is null, the BYTE macro builds a DC using information passed from previous calls of the BIT macro.
- if the operand is present, BYTE generates a DC statement using this parameter.

EXAMPLES OF THE USE

Use with a non-blank parameter.

NAME	OPERATION	OPERAND
FIRST	BYTE	X'CF'
+FIRST	DC	X'CF'

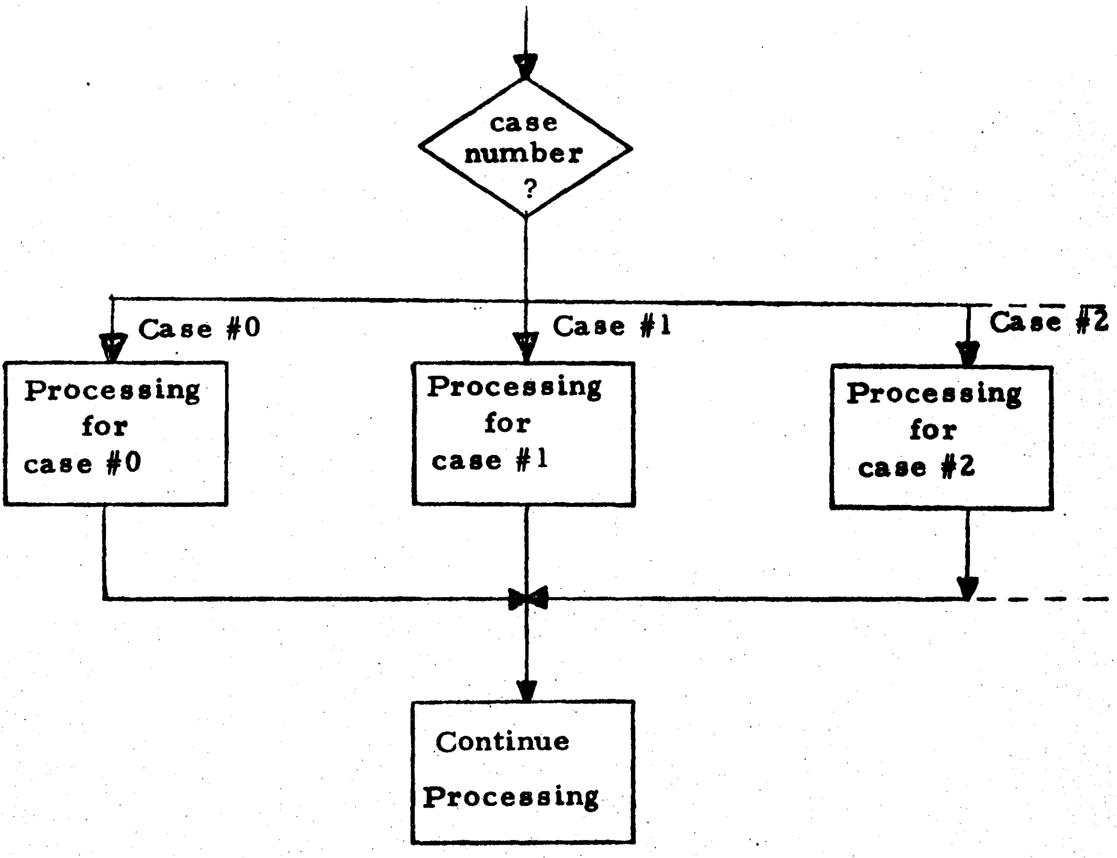
Use in conjunction with the BIT macro

	NAME	OPERATION	OPERAND
	BIT1	BIT	0
+	BIT1	DS	XL(B'10000000')
+		ORG	*-B'10000000'
	BIT3	BIT	2,ON
+	BIT3	DS	XL(B'00100000')
+		ORG	*-B'00100000'
	BIT5	BIT	B'00001000',ON
+	BIT5	DS	XL(B'00001000')
+		ORG	*-B'00001000')
	BIT78	BIT	(6,7),ON
+	BIT78	DS	XL(B'00000011')
+		ORG	*-B'00000011'
	ALL	BYTE	
+	ALL	DC	B'00101011'

NAME - CASE

DESCRIPTION

The purpose of this macro is to generate the code necessary for certain, frequently encountered, decision table type processing logic. In this type of processing one usually has a case (index) number in some GPR and desires to execute one of a list of options (cases) based upon the value of the case number in the GPR. The following block diagram shows the basic flow of this type of logic:



In this macro it is assumed that the increment between the case numbers is a power of two (i. e., 1, 2, 4, 8, . . .) and that the cases are numbered starting with zero. It should be noted that CASE loads the specified RETREG with the address of the instruction following the macro before branching to the determined case; and, it is the responsibility of each case to return to the address specified in the RETREG (if the requirements of structured coding are to be fulfilled). The following shows the formats of the CASE macro:

[symbol]	CASE	case register,	AT = (address list) BT = (address list) (R) LAT = addr. (R) LBT = addr.	[,INDX=number] [,RETREG=register]

case register -
is the register number (or symbol equated to the register number) of the GPR that contains the desired case number. This must not be the same register that is used as the RETREG.

AT = (address list) -
is a list of up to 255 case labels. This list of case labels is used to generate a corresponding list of address constants. When this form of the CASE macro expands the case register will be used to index into this list of ADCONS, inorder to determine which case is to be branched to. There is a one-to-one correspondence between a labels position in the list and its associated case number (i. e., the first label in the list is the name of the case which is to receive control when the case register contains a zero. If a label is left null an address of zero will be generated for the associated case number. (This should be used for any embedded cases numbers, which are not expected to occur and which a program check is desired if it ever does occur). An * may be coded in place of any of the labels to signify that processing is just to continue at the instruction following the macro when the associated case(s) occurs. It should be noted that by specifying one or more of the labels (used in an AT type expansion) in an EXTRN statement, the CASE macro becomes effectively an indexed CALL macro.

BT = (address list) -
is a list of up to 255 case labels, as defined for the AT type expansion. The only difference between the AT and the BT type expansions is that BT generates a branch table instead of an address table for the labels specified. This permits the use of case labels that are not in the same CSECT nor callable, but for which a base register is set up.

(R)
LAT = addr. -
is the address of a remote list address table to be used by CASE in determining where to branch for each value that can be placed in the case register. This address may be specified in a register as (R) where R is some register number (not being used as a case register or a RETREG).

(R)
LBT = addr. -
is the address of a remote list of branch instructions to be used by the CASE in branching to the case designated by the value in the case register. As in LAT this address may also be specified in a register form.

INDX = number
is used to specify the increment used in counting the cases. This must be some power of 2 (i.e., 1, 2, 4, 8, 16, 32, . . .). The default for INDX is 4. (This says that the cases are numbered 0, 4, 8, 12, 16, . . .).

RETREG = register -
is used to specify the register to be setup as the linkage register on the branch. This is specified as any register number or symbol equated to a register number. The default for RETREG is 14.

EXAMPLES OF USE

In the following examples NUM is equated to a GPR that contains the case number.

```
 XXX          CASE      NUM,AT=(*,MUD,,GARB)
+             CNOP      0,4
+XXX          BAL       14,*+20
+             DC        A(*+10+4*(4-1))
+             DC        A(MUD)
+             DC        A(0)
+             DC        A(GARB)
+             L         15, 0 (14,NUM)
+             BALR      14,15
```

```
 XXX          CASE      NUM,BT=(*;MUD,,GARB)
+XXX          LA        14,*+4+20
+             B         *+4(NUM)
+             B         *+4+4*(4-1)
+             B         MUD
+             DC        A (0)
+             B         GARB
```

```
 XXX          CASE      NUM,LAT=($10),RETREG=$8,INDX=1
+XXX          SLL       NUM,2
+             L         15,0(NUM,$10)
+             BALR      $8,15
```

```
 XXX          CASE      NUM,LBT=MUD,INDX=32,RETREG=$9
+XXX          SRL       NUM,3
+             BAL       $9,MUD(NUM)
```

NAME - DO

DESCRIPTION

The DO MACRO generates a branch-and-link to a segment of code, defined by the BGNSEG and ENDSEG MACROs.

The format of the DO MACRO is:

```
    DO     SEGMENT,REG
+   BAL    REG,SEGMENT
```

where SEGMENT is the label of the section of code to be branched to and REG is the register to be used. If the register is not specified, register 14 will be used.

If the register to be used in branching to and from a segment has already been defined by a previous DO or BGNSEG MACRO, issuing a different register will cause an error message to be printed. Registers need to be expressed in notation $1, $2 etc. A maximum of 50 segments may appear in an assembly.

EXAMPLES

Example 1

```
                     DO         COMPUTE
  +                  BAL        $14,COMPUTE
                      ~
                     BGNSEG     COMPUTE
  +COMPUTE           DS         OH
                      ~
                     ENDSEG     COMPUTE
  +                  BR         $14
```

Example 2

```
                          DO           CODE,$6
   +                      BAL          $6,CODE
                          ⌇
                          BGNSEG       CODE
   +CODE                  DS           OH
                          ⌇
                          ENDSEG       CODE
   +                      BR           $6
                          ⌇
                          DO           CODE,$7
   +      4,****          WRONG REGISTER HAS BEEN SPECIFIED
```

NAME - ELSE

DESCRIPTION

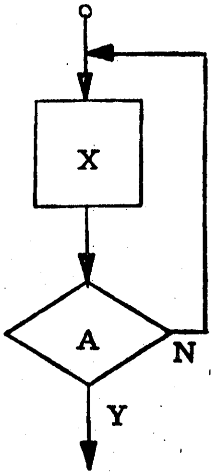
The function of the ELSE macro is to generate the branch and labels that correspond with the branch instructions generated by the IF macro and the labels generated by the ENDIF macro. See the IF macro.

NAME - ENDDO

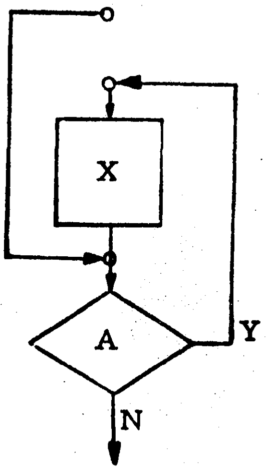
DESCRIPTION

The function of the ENDDO macro is to generate the labels that correspond to the labels and instructions generated by the WHILE/UNTIL macros. See the WHILE or UNTIL macros.

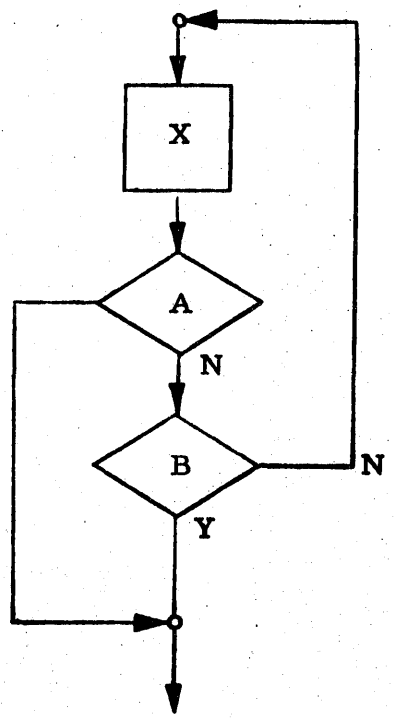
```
UNTIL  A,DO

   X

ENDDO
```



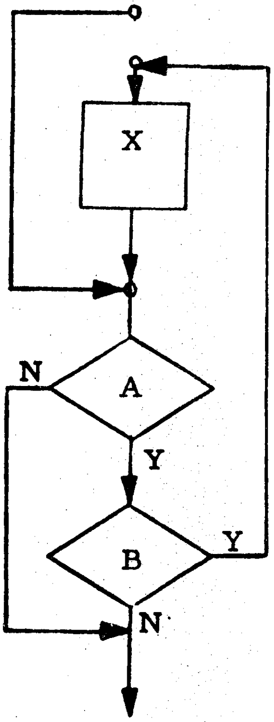
```
WHILE  A,DO

   X

ENDDO
```



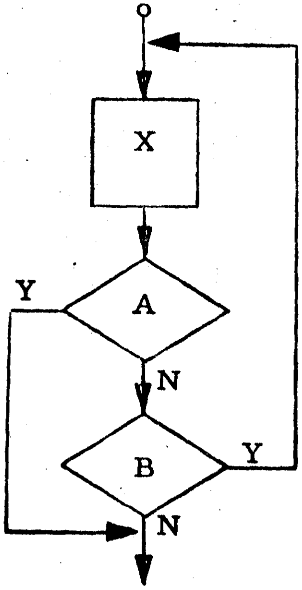
```
UNTIL  A,OR

UNTIL  B,DO

   X

ENDDO
```



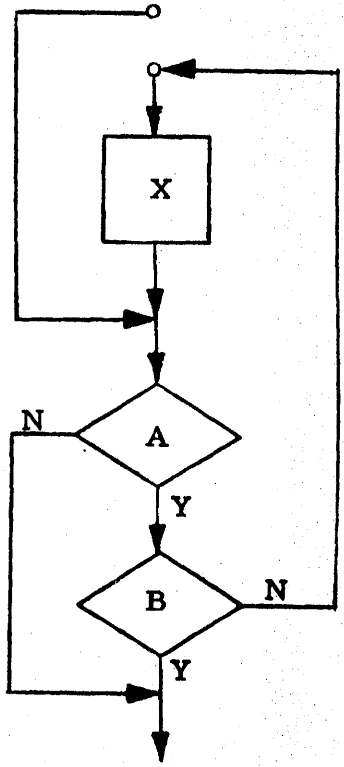
```
WHILE  A,AND

WHILE  B,DO

   X

ENDDO
```



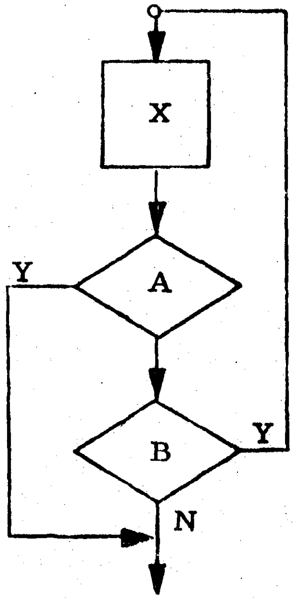
```
UNTIL  A,AND
WHILE  B,DO
   X
ENDDO
```



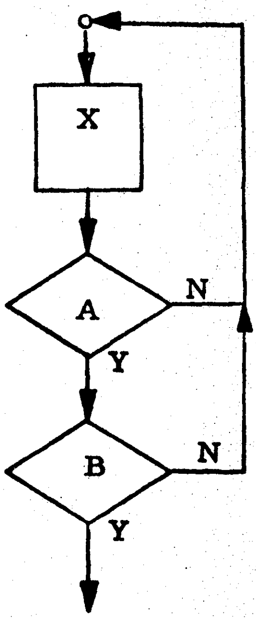
```
WHILE  A,AND
UNTIL  B,DO
   X
ENDDO
```



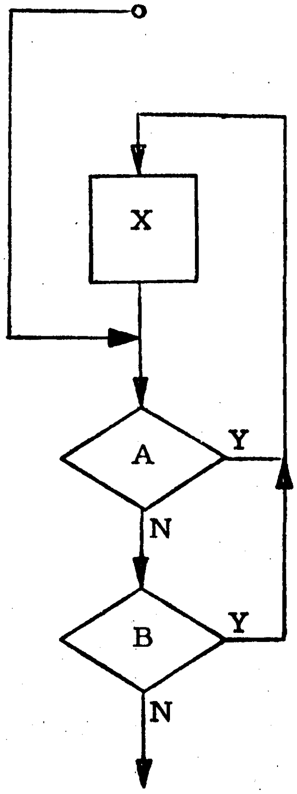
```
UNTIL  A,OR
WHILE  B,DO
   X
ENDDO
```



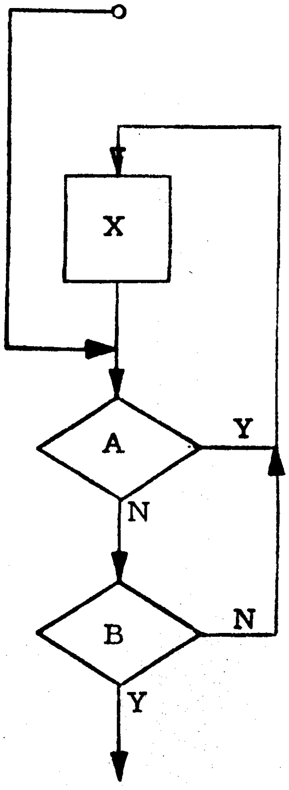
```
UNTIL  A,AND
UNTIL  B,DO
   X
ENDDO
```



```
WHILE  A,OR

WHILE  B,DO

   X

ENDDO
```



```
WHILE  A,OR

UNTIL  B,DO

   X

ENDDO
```



NOTES: In an UNTIL a BCT = yes when the register = 0 after execution of BCT.
In a WHILE a BCT = no when the register = 0 after execution of BCT.

NAME - ENDIF

DESCRIPTION

The function of the ENDIF macro is to generate the labels that correspond with the branch instructions generated by the IF macro. See the IF macro.

NAME - ENDLOOP

DESCRIPTION

The function of the ENDLOOP macro is to define the end of the loop. See the STRTSRCH macro.

NAME - ENDSEG

DESCRIPTION

The ENDSEG MACRO generates a BR instruction. It is used to return from a segment of code that has been branch-and-linked to by the DO MACRO.

The format is:

```
                      ENDSEG        SEGMENT
   +                  BR            REG
```

where SEGMENT is the name of the segment to be terminated and REG is the register to be used. The register is determined by either a previous DO or BGNSEG MACRO.

EXAMPLES

Example 1

```
                      BGNSEG        COMPUTE,$6
   +COMPUTE           DS            OH

                      ENDSEG        COMPUTE
   +                  BR            $6
```

Example 2

```
                      DO            CODE,$7
   +                  BAL           $7,CODE

                      BGNSEG        CODE
   +CODE              DS            OH

                      ENDSEG        CODE
   +                  BR            $7
```

NAME - ENDSRCH

DESCRIPTION

The function of the ENDSRCH macro is to indicate the end of the complete macro set. See the STRTSRCH macro.

NAME - ENTER

DESCRIPTION

The 'ENTER' macro is used to generate multiple - entry point code. The macro generates:

1. One CSECT card (CSECT name = 1st subparameter of the first operand.)
2. An "ENTRY" card for each entry point
3. One save area (22wds if 'INTP' appears in col. 1-4)* and 'SAVE' code which establishes R13 as a base register.
4. A label to branch to 'RETURN' (label = an 'R' concatenated with the CSECT name)
5. '$0 EQU 0', . . . , '$15 EQU 15' so that an XREF is given of Register usage if the $XX symbols are used to specify registers. (The 'EQU's are generated only once per assembly even though more than one 'ENTER' is coded.)
6. Register 15 is loaded with the address of the code associated with the resp. entry point (i.e., the resp. name specified in the second operand sublist - if left blank, '$' is concatenated with the resp. entry point specified in the first operand sublist) so that one executes 'BR $15' after executing code which is common for all entry points.

EXAMPLE OF USE

```
INTP    ENTER  (X, Y, Z), (, ZINTRNAL)
        SR $7, $7          ** FOR LATER USE
        L $12, =V(MGLBAT)
        L $12, $12, 0($12)
        L $10, 4*X'2D'  ($12)
        USING SBL2DA, $12
        TC INTP
        BR $15
```

---

* See Reference 4, for discussion of INTP.

```
$X          DS        OH
            GSIN      A
            STE       O, B
            B         GOTIT

$Y          GSIN      AA
            STE       O, B
            ST        $7, Q       Q=0
            B         GOTIT

ZINTRNAL    GSIN      AA
            STE       O, B

GOTIT       EQU       *
```

NAME - EQUATE

DESCRIPTION

The 'EQUATE' macro is used to generate '$0 EQU 0' . . . '$15 EQU 15' and 'FPRO EQU 0' ... 'FPR6 EQU 6' statements by both the 'HEADC' and 'ENTER' macros. In a CSECT which does not require a save area (and hence wouldn't use' HEADC' or 'ENTER'). one may use 'EQUATE' itself to get the EQU's generated.

EXAMPLE OF USE

```
X           CSECT
            EQUATE
+$0         EQU 0
+$1         EQU 1
             {
+$15        EQU 15
+FPRO       EQU 0
+FPR2       EQU 2
+FPR4       EQU 4
+FPR6       EQU 6
            STM $14, $12, 12 ($13)
            USING X, $15
             {
            END
```

NAME - ERRENTER

DESCRIPTION

ERRENTER &A

&A = a symbol not greater than four characters in length.

The ERRENTER macro should be used to begin a segment of special error processing for a particular error designated by &A, which should have been specified in an ERREXIT macro. The segment should end with (1) an ERRMSG macro for an error message if one is required, (2) another ERRENTER macro for a different error condition, or (3) the ERRETURN macro.

If the ERRENTER macro is preceded by another ERRENTER macro (with no ERRMSG macro between the two), it will expand to a branch to ERRETURN prior to defining the error symbol. Otherwise, it will merely expand to a definition of the error symbol.

The following example shows how ERRENTER would be used to process special error conditions.

Suppose there are three error conditions (ER1, ER2, ER3), one which requires an error message only, one which requires special processing only, and one which requires special processing and an error message. The following code demonstrates the use of ERRENTER in conjunction with the other ERROR MACROS to accomplish these results:

```
                 body of csect with ERREXIT macros
                   to ER1, ER2, ER3)

GRETURN
ERRMSG ER1
DC   C'(error message for er1)'
ERRENTER ER2
                (special processing for er2)
ERRENTER ER3
                (special processing for er3)
ERRMSG
 DC   C'(error message for er3)'
ERRETURN
                (common error processing)
GRETURN
```

This code would expand as follows:

```
                ≀  (body of csect with ERREXIT macros to ER1, ER2, ER3)
           GRETURN
+          B      R&SYSECT
           ERRMSG  ER1
+ERXTER1   BAL  0,ERREXIT$
           DC     C' (error message for er1)'
           ERRENTER ER2
+ERXTER2   DS     0H
            ≀       (special processing for er2)
           ERRENTER ER3
+          B      ERREXIT$
+ERXTER3   DS     0H
            ≀       (special processing for er3)
           ERRMSG
+          BAL        0,ERREXIT$
           DC     C' (error message for er3)'
           ERRETURN
+ERREXIT$  DS     0H
            ≀        (common error processing)
           GRETURN
+          B      R&SYSECT
```

NAME - ERRETURN

DESCRIPTION

ERRETURN

The ERRETURN macro expands to a definition of the symbol ERREXIT$. The ERRETURN macro should be used to begin common error processing. See the ERRENTER macro for examples of its use.

NAME - ERREXIT

DESCRIPTION

ERREXIT &A, &B, &C, &D, &E, &F, &G

&A = 'IF'
SYMBOL

If a symbol is coded for &A, it must be not greater than 4 characters long and it should be the operand of an ERRENTER or ERRMSG macro elsewhere in the CSECT. &B - &G will be ignored, and the macro will generate a BC 15, ERXT (symbol).

If &A = IF, then the operands &B - &G should be coded exactly as they were operands of an IF macro with the exception of &F. &F is normally 'THEN', 'AND', or 'OR' in the IF macro, but it should be a symbol not greater than four characters long in the ERREXIT macro and the same symbol should be the operand of an ERRENTER or ERRMSG macro elsewhere in the program.

Using ERREXIT in case 2 will expand into the same code that the IF macro does except for the BRANCH instruction generated by IF. Instead it will generate a BRANCH to the symbol ERXT(symbol)on the condition specified by the operands &B - &E. (No ENDIF should be associated with an ERREXIT macro).

Example 1:

```
             ERREXIT  IF,F,($3),IS,ZERO,REGZ
+            LTR      $3,$3
+            BC       8,ERXT REGZ

             ERRMSG   REGZ
+ERXTREGZ    BAL      0,ERREXIT$
             DC       C'cannot specify zero reg' (error message)
```

```
             ERRETURN
+ERREXIT$    DS    0H
             PUT   ERDCB,(0)
             GRETURN
+            B     R&SYSECT
```

Example 2:

```
             ERREXIT  ERR2
+            BC    15,ERXTERR2

             ERRENTER  ERR2
+ERXTERR2    DS    0H
                    do special error processing
             ERRMSG
+            BAL   O,ERREXIT$
             DC    C'(error message)'
             ERRETURN
+ERREXIT$    DS    0H
                    do common error processing

             GRETURN
+            B     R&SYSECT
```

NAME - ERRMSG

DESCRIPTION

ERRMSG &A [, &B]

&A = a symbol not greater than 4 characters in length
&B = a register number (defaults to 0)

The ERRMSG macro should be used to define an error message for the error condition designated by &A. &A should be left blank if the error condition was designated by an ERRENTER macro (with the associated special error processing) immediately preceding the ERRMSG macro.

The error link register is specified by &B and should be specified only by the first ERRMSG macro in the CSECT. &B will then default to that of the first ERRMSG macro for subsequent ERRMSG macros and will default to 0 on the first ERRMSG macro if not specified.

The ERRMSG macro expands to a BAL off the error link register to ERRETURN, defining the BAL instruction with the error symbol, if one is specified.

See the ERRENTER macro for examples.

NAME - EXITIF

DESCRIPTION

The function of the EXITIF macro is to test a condition to see whether to continue the loop or exit out of the loop. See the STRTSRCH macro. The following shows the format of the EXITIF macro.

$$\text{EXITIF} \quad \left[\text{condition}\right] \quad , \quad \left\{\begin{array}{c}\text{OR}\\ \text{AND}\\ \text{THEN}\end{array}\right\} \quad \left[\text{, REG=}\right]$$

The condition format is the same as the IF macro except that the label IF is not specified.

NAME - GRETURN

DESCRIPTION

The GRETURN macro expands to a B R&SYSECT. It should be used in conjunction with the HEADC and ENTER macros.

NAME - HEADC

DESCRIPTION

HEADC will generate the CSECT card, save area, entry coding, and return coding for a single entry point Assembler Language program. It will also invoke the EQUATE macro.

The HEADC macro is written as follows:

CSECT name HEADC [INTP = YES]* [,RET = YES]

"CSECT name" will be the name on the generated CSECT card and the entry point for the program. If INTP = YES is coded, a 22-word save area will be generated in place of the standard 18-word save area. A 22-word save area is needed if the program INTP is used

If RET = YES is coded, register 15 will not be restored as part of the return logic, allowing the programmer to store a return code in that register. INTP = YES and RET = YES are not positional parameters; they are keyword parameters.

HEADC will point GPR 13 to the save area and do a 'USING' on GPR 13 so that it will serve as the base register for the program. The return logic can be reached by branching to the label RCSECT name. If this label is more than eight characters, the right-most character is truncated in the generated macro label, and an assembly error is flagged in the 'B RCSECT' statement. To avoid the error message, the programmer should truncate RCSECT to eight characters in coding the 'B RCSECT' statement.

EXAMPLE OF USE

Example 1

```
MUD         HEADC
            {
            {  other code
            {
            B  RMUD
```

---

* See Reference 4, for discussion of INTP.

Example 2

```
MUDAGARB    HEADC       INTP = YES
                         ≀
            TC             INTP
                         ≀
            B .         RMUDAGAR
```

Example 3

```
MUD3        HEADC       RET = YES
            ≀
            L           $15,=F'2'
            B             RMUD3
```

NAME - IF

DESCRIPTION

The function of the IF macro is to generate the labels and instructions that branch to these labels to accomplish the IF-THEN, IF-AND-THEN, IF-OR-THEN, IF-THEN-ELSE, IF-AND-THEN-ELSE, and IF-OR-THEN-ELSE programming functions.

THE IF MACRO SPECIFICATIONS

There are six different IF statements. They are: IF-THEN, IF-AND-THEN, IF-OR-THEN, IF-THEN-ELSE, IF-AND-THEN-ELSE, and IF-OR-THEN-ELSE.

The format for the IF-THEN is:

```
  IF condition

    code - body
  ENDIF
```

which reads "IF the tested condition is true, then execute the code-body."

The format for the IF-AND-THEN is:

```
  IF condition, AND
  IF condition, THEN

    code - body
  ENDIF
```

which reads "IF both conditions are satisfied, then execute the code-body."

The format for the IF-OR-THEN is:

```
  IF condition, OR
  IF condition, THEN

    code - body

  ENDIF
```

which reads "IF either condition is satisfied, then execute the code-body."

The format for the IF-THEN-ELSE is:

```
   IF condition, THEN

     code - body1

   ELSE

     code - body2

   ENDIF
```

which reads "IF the condition is true, THEN execute code-body1, ELSE execute code-body2.

The format for the IF-AND-THEN-ELSE is:

```
   IF condition, AND
   IF condition, THEN

     code - body1

   ELSE

     code - body2

   ENDIF
```

whech reads "IF both conditions are satisfied, THEN execute code-body1, ELSE execute code-body2.

The format for the IF-OR-THEN-ELSE is:

```
   IF condition, OR
   IF condition, THEN

     code-body1

   ELSE

     code-body2
```

ENDIF

which reads "IF either condition is satisifed, THEN execute code-body1, ELSE execute code-body2.

THE IF MACRO FLOWCHARTS

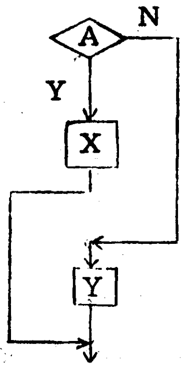
IF A, THEN

X

ELSE

Y

ENDIF



IF A, AND

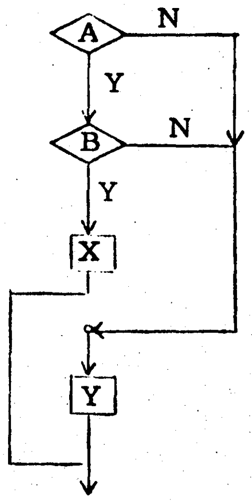
IF B, THEN

X

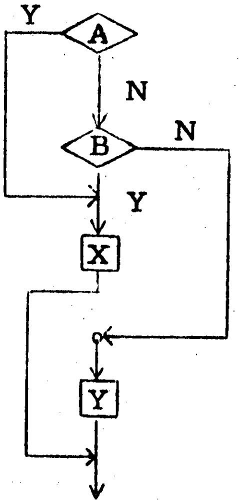
ELSE

Y

ENDIF



```
IF A, OR

IF B, THEN

X

ELSE

Y

ENDIF
```



The following shows the format of IF.

```
                                         (R2)
                                         LABEL2
                                         ONE(S)
                                         OVERFLOW
           [ *  ]             IS       { PLUS     }
           [ BIT]           { GT }     { MINUS    }      {AND }
           [ B  ]   (R1)    { LT }     { MIXED    }      {OR  }
IF         [ H  ] ,LABEL1,  { GE }  ,  { ZERO(S)  }   ,  {THEN}      [,REG=]
           [ F  ]           { LE }     { NMINUS   }
           [ E  ]           { EQ }     { NPLUS    }
           [ D  ]             NE       { NONE(S)  }
           [ T  ]                      { NZERO(S) }
           [ C  ]                      { OFF      }
                                         ON

                                                      { AND }
IF  [TYPE]  ,LABEL, OPERATION, CONDITION,             { OR  }        [,REG=]
                                                      { THEN}
```

The different types are:

1. An * in the type field stands for the condition is already set. When using the * type, the Operation and Condition Fields cannot be omitted.

   Examples:

```
     IF *,,IS,PLUS,THEN
+    BC  13,LABEL
     CODE-BODY
     ENDIF
+LABEL  EQU  *
```

```
     IF    *,,EQ,LABEL,THEN
+    BC    7,LABEL1
     CODE-BODY1
     ELSE
+    B     LABEL2
+LABEL1  EQU *
     CODE-BODY2
     ENDIF
+LABEL2  EQU *
```

2. Bit type: will generate a test under mask. The only valid operation parameter is (IS) and the only valid condition parameters are: ZERO, ONE, ON, OFF, MIXED, NONES, NMIXED, and NZERO.

$$\text{IF BIT, LABEL, IS, }\left\{\begin{array}{l}\text{ZERO}\\ \text{ONES}\\ \text{ON}\\ \text{OFF}\\ \text{MIXED}\\ \text{NONES}\\ \text{NMIXED}\\ \text{NZERO}\end{array}\right\},\left\{\begin{array}{l}\text{AND}\\ \text{OR}\\ \text{THEN}\end{array}\right\}$$

Examples:

```
     IF    BIT,A,IS,ZERO,THEN
+    TM    A,L'A
+    BC    7,LABEL
     CODE-BODY
     ENDIF
+LABEL  EQU *
```

3. B type:

$$\text{IF B,LABEL1,IS, }\left\{\begin{array}{l}\text{ZERO(S)}\\ \text{PLUS}\\ \text{MINUS}\\ \text{NPLUS}\\ \text{NMINUS}\\ \text{NZERO(S)}\end{array}\right\},\left\{\begin{array}{l}\text{AND}\\ \text{OR}\\ \text{THEN}\end{array}\right\}$$

$$\text{IF B, LABEL1, } \left\{\begin{matrix}\text{GT}\\ \text{LT}\\ \text{EQ}\\ \text{NE}\\ \text{GE}\\ \text{LE}\end{matrix}\right\} , \left\{\begin{matrix}\text{T '}\\ \text{L '}\\ \text{X '4F'}\\ \text{C 'FF'}\\ \text{B '01'}\\ \text{LABEL2}\\ \text{(R2)}\\ \text{NUMBER}\end{matrix}\right\} , \left\{\begin{matrix}\text{AND}\\ \text{OR}\\ \text{THEN}\end{matrix}\right\} \quad \left[\text{,REG=}\right]$$

REG = DEFAULTS TO $0

Examples:

```
      IF  B,A,IS,ZERO,THEN
+     CLI  A,X'00'
+     BC   7,LABEL
      CODE-BODY
      ENDIF
+LABEL  EQU *
```

```
      IF  B,A,EQ,BBB,THEN,REG=$1
+     IC  $1,BBB
+     STC $1,*+5
+     CLI  A,X'00'
+     BC   7,LABEL
      CODE-BODY
      ENDIF
+LABEL  EQU *

      IF  B,A,EQ,($1),THEN
+     STC $1,*+5
+     CLI  A,X'00'
+     BC  7,LABEL
      CODE-BODY
      ENDIF
+LABEL  EQU *
```

These B forms of the IF statement alters executable code and are not usable if the program is to be reentrant.

```
     IF  B,A,EQ,B,THEN
+    CLC  0+A,B
+    BC   7,L1

     CODE-BODY

     ENDIF
L1   DS  0H
                                       Reentrant B Type
     IF  B,B,EQ,($1),THEN
+    EX   $1,*+8
+    B    *+8
+    CLI  B,0
+    BC   7,L1

     CODE-BODY

     ENDIF
L1   DS  0H

     IF  B,A,GT,X'4F',THEN
+    CLI  A,X'4F'
+    BC  13,LABEL
     CODE-BODY
     ENDIF
+LABEL  EQU *
```

```
         IF      B,A,GT,138,THEN
+        CLI     A,138
+        BC      13,LABEL

         CODE-BODY

         ENDIF
+LABEL    EQU  *

         IF      B,A,GT, 0+MUD,THEN
+        CLI     A,0+MUD
+        BC      13,LABEL

         CODE-BODY

         ENDIF
+LABEL   EQU    *


MUD      EQU   186
```

4. **Fixed-Point (H or F):**

```
                                     ┌ONE(S)   ┐
                                     │PLUS     │
                                     │MINUS    │
IF  {H} ,  (R1)     ,  IS  ,         │ZERO(S)  │ ,  {AND  }    [, REG=]
    {F}    LABEL1                   {│NZERO(S) │}   {OR   }
                                     │NMINUS   │    {THEN }
                                     │NPLUS    │
                                     └NONE(S)  ┘
```

```
                         ┌GT┐      ┌LABEL2 ┐
                         │LT│      │(R2)   │
IF   {H} ,  (R1)        {│GE│} ,  {│=F' '  │} ,  {AND  }    [, REG=]
     {F}    LABEL1,      │EQ│      │=H' '  │     {OR   }
                         │NE│      │=X' '  │     {THEN }
                         └LE┘      └=C' '  ┘
```

[REG=]   Defaults to $0

```
    IF  H,A,IS,PLUS,THEN
+   LH  $0,A
+   LTR  $0,$0
+   BC  13,LABEL
    BODY-CODE
    ENDIF
+LABEL  EQU  *

    IF  H,($1),IS,ZERO,THEN,REG=($1)
+   LTR   $1,$1
+   BC  7,LABEL
    BODY-CODE
    ENDIF
+LABEL  EQU  *
```

```
     IF  H,A,GT,B,THEN,REG=$5
+    LH  $5,A
+    CH  $5,B
+    BC  13,LABEL
     BODY-CODE
     ENDIF
+LABEL  EQU  *

     IF  H,A,EQ,($1),THEN

+    CH  $1,A
+    BC  7,LABEL
     BODY-CODE
     ENDIF
+LABEL  EQU  *

     IF  H,($1), EQ,($2),THEN,REG=($1)
+    CR  $1,$2
+    BC  7,LABEL
     BODY-CODE
     ENDIF
+LABEL  EQU  *

     IF  F,A,IS,PLUS,THEN
+    L  $0, A
+    LTR  $0,$0
+    BC  13,LABEL
     BODY-CODE
     ENDIF
+LABEL  EQU  *

     IF  F,($1), IS,ZERO,THEN,REG=($1)
+    LTR  $1,$1
+    BC  7,LABEL
     BODY-CODE
     ENDIF
+LABEL  EQU  *
```

```
    IF  F,A,GT,B,THEN,REG=($5)
+   L  $5,A
+   C  $5,B
+   BC  13,LABEL
    BODY-CODE
    ENDIF
+LABEL  EQU  *

    IF  F, ($1),GT,B,THEN,REG=($1)
+   C  $1,B
+   BC  13,LABEL
    BODY-CODE
    ENDIF
+LABEL  EQU  *

    IF  F,A,EQ, ($1),THEN

+   C  $1,A
+   BC 7,LABEL
    BODY-CODE
    ENDIF
+LABEL  EQU  *

    IF  F, ($1), EQ, ($2), THEN,REG=($1)
+   CR  $1,$2
+   BC  7, LABEL
    BODY-CODE
    ENDIF
+LABEL  EQU  *
```

5. Floating-Point (E or D):

```
                           ┌ONE(S)  ┐
                           │PLUS    │
                           │MINUS   │
     ┌E┐     (R1)          │ZERO(S) │     ┌AND ┐   ┌      ┐
IF   │ │ ,  LABEL1,IS,    <│NZERO(S)│> ,  │OR  │   │,REG= │
     └D┘                   │NMINUS  │     └THEN┘   └      ┘
                           │NPLUS   │
                           └NONE(S) ┘
```

```
                          { GT }    { LABEL2 }
        { E }   (R1)      { LT }    { (R2)   }    { AND  }
    IF  {   } , LABEL1,   { GE } ,  { =E' '  }  , { OR   }  [, REG=]
        { D }             { EQ }    { =D' '  }    { THEN }
                          { NE }
                          { LE }
```

REG= Defaults to FPRO

```
     IF  E,A,IS,PLUS,THEN
+    LE  FPRO,A
+    LTER  FPRO,FPRO
+    BC    13,LABEL
     BODY-CODE
     ENDIF
+LABEL  EQU  *

     IF  D,(FPRO),IS,ZERO,THEN,REG=(FPRO)
+    LTDR  FPRO,FPRO
+    BC  7,LABEL
     BODY-CODE
     ENDIF
+LABEL  EQU  *

     IF  E,A,GT,B,THEN,REG=(FPR4)
+    LE  FPR4,A
+    CE  FPR4,B
+    BC  13,LABEL
     BODY-CODE
     ENDIF
+LABEL  EQU   *

     IF  D,(FPRO),GT,B,THEN,REG=(FPRO)
+    CD  FPRO,B
+    BC  13,LABEL
     BODY-CODE
+LABEL  EQU  *
```

```
     IF  E,A,EQ,(FPR2),THEN

+    CE  FPR2,A
+    BC  7,LABEL
     BODY-CODE
     ENDIF
+LABEL  EQU  *

     IF  D,(FPRO),EQ,(FPR2),THEN,REG=(FPRO)
+    CDR  FPRO,FPR2
+    BC  7,LABEL
     BODY-CODE
     ENDIF
+LABEL  EQU  *
```

6. Type Field Omitted:

```
     IF  ,LABEL1,IS,ZERO,THEN,REG= (FPRO)
+    LE  FPRO,LABEL1
+    LTER  FPRO,FPRO
+    BC  7,LABEL2
     BODY-CODE
     ENDIF
+LABEL2    EQU  *

LABEL1   DC  E'0'
```

7. Character (C):

$$\text{IF} \quad \text{C, LABEL1,} \left\{\begin{matrix}\text{LT}\\ \text{GT}\\ \text{GE}\\ \text{EQ}\\ \text{LE}\\ \text{NE}\end{matrix}\right\}\text{, LABEL2,} \left\{\begin{matrix}\text{AND}\\ \text{OR}\\ \text{THEN}\end{matrix}\right\} \left[\text{REG=}\right]$$

Example:

```
     IF       C,ABLE,EQ,BETA,THEN
+    CLC      ABLE,BETA
+    BC       7,LABEL

     CODE-BODY

     ENDIF
+LABEL  EQU  *
```

8. Test Under Mask (T)

```
                                         ⎧ ZERO   ⎫
                                         ⎪ ONES   ⎪   ⎧AND ⎫
IF        T, LABEL1,     ⎧      ⎫        ⎪ ON     ⎪   ⎨OR  ⎬    ⎡      ⎤
                         ⎨ MASK ⎬   ,    ⎨ OFF    ⎬ , ⎩THEN⎭    ⎣,REG= ⎦
                         ⎩      ⎭        ⎪ MIXED  ⎪
                                         ⎪ NONES  ⎪
                                         ⎩ NMIXED ⎭
```

Example:

```
          IF     T,A,X'11',ZERO,THEN
+         TM     A,X'11'
+         BC     7,LABEL

          CODE-BODY

          ENDIF

+LABEL   EQU   *
```

PROGRAMMING NOTES

1. There can be as many as 20 nested IF statements. Each IF statement has to have a corresponding ENDIF statement.

2. The level of a nested IF statement can be found in the LABELS that are generated.

   Example:

```
    IF  condition,  THEN
+   BC    ,IF  5   0025


    ENDIF
+IF 5   0025   EQU  *
```

   The 5 stands for the level of this nested IF statement.

3. There is no limit on the number of IF-OR/IF-AND statements but after the last IF-OR/IF-AND statement there has to be an IF-THEN statement.

4. Any time a register notation is used in an IF statement the register must be in parentheses. If the parentheses are left off the IF macro would treat the register number as a label.

5. Misspelling and abbreviation of "conditions" mnemonices is not allowed.

6. The default register for fixed-point instructions is $0 and for floating-point instructions is FPRO.

7. In using the structured code macros

```
               GT
   ~~~label,   LT ,  ZERO,~~~       generates more inefficient code than does
               EQ
```

   the equivalent statement using the IS opcode.

```
                          PLUS
   i.e.,   ~~label, IS,   ZERO  ~~~~~~
                          MINUS
```

8. Reentrant programs that use the B TYPE (BYTE) IF statements should set the global flag &$RENT to 1. This flag will assure that the code generated by the IF macro is reentrant. This reentrant code is slower than the none reentrant code and should be used only in reentrant programs. The global flag has to be defined and set before a CSECT statement. See the examples of the B TYPE IF statement.

   Example:

```
       GBLB  &$RENT
&$RENT SETB  1
XXXXXX CSECT
```

NAME - MATH

DESCRIPTION

This macro can be used to save coding time when coding equations in Assembler Language, by translating an equation oriented language into Assembler Language. Basically, the MATH macro is similar to the RTFMT macro in that it translates character strings into Assembler Language instructions.

The following is an attempt to describe how MATH works and how to use it effectively.

INTRODUCTION

The MATH macro can be used to convert a "quoted-character-string", of valid "OPERANDS", separated by valid "OP-CODES", into their corresponding Assembler Language instructions. The main purpose of this macro is to let the user write a floating point equation or expression in a manner similar to that used in FORTRAN. It therefore has been designed around the floating point instruction set; though, by correct choice of options many of the fixed point instructions may be utilized.

Before considering any of MATH's advantages, disadvantages, applications, etc., certain definitions should be presented and a description given of how MATH processes an expression.

DEFINITION

OP-CODE - An OP-CODE is a special character or combination of characters which designates the operation to be performed using the following "OPERAND". In general there exist a one-to-one correspondence between each OP-CODE and some Assembler Language instruction.

All OP-CODES must be immediately preceeded and followed by atleast one blank. The following is a list of the valid OP-CODES and their correspondence machine operation:

OP-CODE	Operation	
+	Add	Mathematical OP-CODES
PLUS	Add	
-	Subtract	
MINUS	Subtract	
/	Divide	
OVER	Divide	
*	Multiply	
TIMES	Multiply	
=	Store	Store OP-CODES
STORE	Store	
STORE-IN	Store	
SAVED-IN	Store	
C	Compare	Compare OP-CODES
WITH	Compare	
COMPARE	Compare	
TO	Compare	
$	Place label on next instruction	Equate OP-CODES
EQU	Place label on next instruction	
HERE=	Place label on next instruction	
LABEL=	Place label on next instruction	
XOR	Exclusive OR	Logical OP-CODES (may only be used in fixed point mode)
OR	Or	
AND	And	
.	Load	Load OP-CODE
LOAD	Load	
RELOAD	Load	

Besides the above OP-CODES there is also a set of OP-CODES which correspond to many of the extended mnemonics for branch on conditions. The list of these OP-CODES and there corresponding branch conditions are listed on the next page:

OP-CODES	Condition	
B	15	Branch-on-condition OP-CODES
BH	2	
BL	4	
BE	8	
BO	1	
BP	2	
BZ	8	
BNH	13	
BNL	11	
BNE	7	
BNP	13	
BNZ	7	

NUMBER - Any combination of characters which begins with a - , . , or a 0 - 9, will be placed in the corrected precision floating point literal. There may be no internal blanks in the character combinations making up the NUMBER.

Note: Further information on valid NUMBER character combination may be found under floating point constants in the Assembler Language Manual (C28-6514).

Note: NUMBERS may not be used in fixed point mode. Instead LITERALS should be used in their place. (see page 3-4 for definition of a LITERAL.)

Examples:

1, -400, .100, 1.0 E-10, .0001E5, 0.100, 0, 1.054, 100, etc.

TERM - A TERM is any combination of characters which begins with a letter (A → Z, $, @). Each TERM is assumed to be a valid Assembler Language operand. There may be no internal blanks in the character combination making up a TERM.

SYMBOL - In writing a symbol the following rules must be conformed to:

1. A symbol must consist of one to eight characters. The first character must be a letter. The other characters may be letters or digits (0 through 9).
2. No special characters or blanks are allowed in a symbol.

REG - Any combination of Characters beginning with a "(" is assumed to specify a Register (REG). The last character in this character string should be a ")". There may be no internal blanks in the character combination making up the REG. The characters between the first and last paren in the string must either be a valid register number or a symbol which has been previously equated to a register number.

Note: The macro will set up the following equates in each assembly in which it is used:

FPR0	EQU	0	These are therefore special symbols and should not be used as statement symbols in an assembly.
FPR2	EQU	2	
FPR4	EQU	4	
FPR6	EQU	6	

LITERAL - A LITERAL has the same definition here as it has in Assembler Language except that as in all character strings all quotes must be replaced by double quotes.

Examples:

Literal as written in Assembler Language	Its Corresponding Literal In Math
=X'4600000'	=X"4600000"
=F'1'	=F"1"
=A(A-B)	=A(A-B)

OPERAND - An OPERAND is any valid SYMBOL, TERM, LITERAL, NUMBER, REG, EXP, or PREFIXED-EXP (see definition below of EXP and PREFIXED-EXP).

EXP - An EXP expression is a combination of OPERANDs separated by the desired OP-CODES. Before the first OPERAND in each EXP must be a "(" followed immediately by at least one blank. After the last OPERAND in each EXP must by a ")", which may be preceded by as many blanks as desired.

EXP-REG - In evaluating each EXP, one register is used to contain all intermediate results such that when the last OP-CODE in the EXP has been processed this register will contain the value of the expression. This register is called the expression's register, "EXP-REG".

Example:

In FPR0 was the EXP-REG for the following EXP, FPR0 would contain a +2 when the last OP-CODE is processed.

```
(1 + 4 - 3)
```

```
LE      FPR0,=E'1'  }
AE      FPR0,=E'4'  }  Code generated by above EXP
SE      FPR0,=E'3'  }
```

PREFIXED-EXP - A PREFIXED-EXP is any EXP which is immediately preceded by a special operation prefix. This prefix will cause the corresponding special operation to be performed on the EXP-REG, of the associate EXP, immediately after the last OP-CODE in the EXP has been processed. All the special operations are register to register operations with the EXP-REG being but the first and second operand. The following is a list of the valid prefixes and the operations they cause to be performed on the EXP-REG.

PREFIX	OPERATION
ABS	Load Positive
NEG	Load Negative
TEST	Load and Test
COMP	Load Compliment
HALF	Halve
DUBL	Add it to itself
SQAR	Multiply it times itself

Example:

If the following EXP were encountered with the EXP-REG = FPR2, then the following code would be generated; if TYP=E

```
    SQAR (A - B)

         LE     FPR2,A     }
         SE     FPR2,B     }   code generated by above EXP
         MER    FPR2,FPR2  }
```

MAIN-EXP - The entire character-string to be converted by MATH is called the MAIN-EXP. It is just like any other EXP, except the beginning card and the ending parenthesis are replaced with single quotes.

Example:

EXP	Corresponding MAIN-EXP
(A * (A - B + C))	'A * (A - B + C)'

INNER-EXP - Any EXP contains characters which are a subset of the characters of another EXP, is an INNER-EXP with respect to this other EXP.

OUTER-EXP - An expression which contains one or more INNER-EXPs is outer to each of them.

REG-LIST - The register symbols specified in the field of the REG parameter is called the REG-LIST (see the macro definition on page 3-13).

SYSPARM-TERM - It is often necessary to reference a number stored as a system parameter in processing an equation. This may be done in the MATH macro in the following manner.

If MXXXXX is the system parameter you wish to use, code:

$$\text{SYSPARM}(\underbrace{\text{MXXXXX}}_{\text{System parameter name}}\underbrace{\text{NN}}_{\text{number (optional)}})$$

MXXXXX = six character system parameter name.

NN = one or two digit number to be used as a displacement of the system parameter in referencing it. This will probably only be needed when referencing a system parameter which is an array such as MHRSYT.

Note: As in all TERMS, there may be no imbedded blanks.

Note: In picking up the address of the system parameter, register 1 will be used.

Examples:

```
MATH    'A * SYSPARM(MCRFMN)', TYP=E
         ↓
         code generated
        LE   FPRO,A
        L    1,=V(MCRFMN)
        ME   FPRO,0(1)

MATH    'A * SYSPARM(MHRSYT8)', TYP=E
         ↓
         code generated
        LE   FPRO,A
        L    1,=V(MHRSYT)
        ME   FPRO,8(1)
```

For more examples, see pages

SPECIAL CAPABILITIES

A special capability exists which lets any valid Assembler Language instruction, which does not contain any quotes, be coded as a Math OP-CODE and OPERAND. This is accomplished by coding a # sign immediately before the Assembler Language mnemonic, skipping at least one blank after the mnemonic, and then coding the OPERAND exactly as it would in the Assembler Language instruction.

Examples:

MATH OPCODE	MATH OPERAND		ASSEMBLY LANGUAGE STATEMENT GENERATED
#ST	3, XYZ	⟶	ST 3, XYZ
#TM	0 (4), 1	⟶	TM 0(4), 1
#SLL	3, 0 (4)	⟶	SLL 3, 0(4)
#ST	ONE, A + 3(5)	⟶	ST ONE, A + 3(5)

Note: This capability lets the user embed special operations within the code generated by the macro without having to break the equation up into several parts.

The ability also exists to raise a floating point number to a floating point power via the MATH Macro. This greatly simplifies the coding needed to accomplish this use of the FRXPR# and FDXPD# "power" routines. Also if the "power" routine must be used more than once per assembly, space will be saved by using MATH rather than the CALL Macro because MATH uses the same argument list each time.

Note: The "power" routine will use all four floating point registers. Therefore, it is not possible to save values in these registers across any MATH expansion in which the "power" facility is used. Also since these registers are used by the "power" routine, the power OP-CODE may not be used except in the MAIN-EXP.

To use this facility one need only use the "**" symbol within the MAIN-EXP. This will cause the value currently contained by this Expression's EXP-REG to be raised to the power stated by the OPERAND immediately following the ** symbol. (See the following examples).

Examples:

The following are examples of a few of the possible uses of the power OP-CODE and the Assembler Language code which will be generated in each case.

```
59          MATH  '  A  **  B  =  A+16'

61+FPRO     EQU   0                ***
62+FPR2     EQU   2                  ***   SET UP EQUATES FOR THE
63+FPR4     EQU   4                  ***    FLOATING POINT REGS
64+FPR6     EQU   6                ***

67+         LE    FPRO,A                  A SYMBOL
68+         B     *+36                    BRANCH PAST PARM LIST
69+PARG0001 DC    D'0'                    FIRST ARG TO POWER ROUTINE
70+PARG0002 DC    D'0'                    SECND ARG TO POWER ROUTINE
71+APARG001 DC    A(PARG0001),X'80',AL3(PARG0002) POWER ARG LIST
72+SVFPROXX DC    D'0'                   WHERE PWR SAVES FPRO
73+         STE   FPRO,PARG0001                  1ST  ARG  TO POWER
74+         LE    FPRO,B                  A SYMBOL
75+         STE   FPRO,PARG0002                  2ND ARG TO POWER
76+         L     15,=V(FRXPR=)            A( REAL*4 POWER ROUTINE )
77+         LA    1,APARG001              A( ARGUMENT LIST )
78+         BALR  14,15        FPRO = ARG1 ** ARG2
79+         STE   FPRO,A+16                    A SYMBOL
80                *,--- REG = FPRO WAS USED IN EVALUATING THE EQUATION
81+*                                                                     *
82+*                                                                     *
83+***** END ***** OF ***** EQUATION ***************************************
```

```
350    MATH  '( A + 1 ) / B  ** ( (  B + 1  ) / A )  = C',TYP=D

352+        LD    FPRO,A                 A SYMBOL
353+        AD    FPRO,=D'1'                NUMBER TYPE
354+        DD    FPRO,B                 A SYMBOL
355+        STD   FPRO,PARG0001                1ST  ARG  TO POWER
356+        LD    FPRO,B                 A SYMBOL
357+        AD    FPRO,=D'1'                NUMBER TYPE
358+        DD    FPRO,A                 A SYMBOL
359+        STD   FPRO,PARG0002                2ND ARG TO POWER
360+        L     15,=V(FDXPD=)           A( DOUBLE PRE  POWER ROUTINE )
361+        LA    1,APARG001             A( ARGUMENT LIST )
362+        BALR  14,15         FPRO = ARG1 ** ARG2
363+        STD   FPRO,C                 A SYMBOL
364               *,--- REG = FPRO WAS USED IN EVALUATING THE EQUATION
365+*                                                                    *
366+*                                                                    *
367+***** END ***** OF ***** EQUATION ***********************************
```

```
199          MATH  '( A + 4.0 )    **   ( B  *  .518 )  =  C  ',TYP=D

201+         LD    FPRO,A                 A SYMBOL
202+         AD    FPRO,=D'4.0'                NUMBER TYPE
203+         STD   FPRO,PARG0001               1ST  ARG  TO POWER
204+         LD    FPRO,B                 A SYMBOL
205+         MD    FPRO,=D'.518'                NUMBER TYPE
206+         STD   FPRO,PARG0002               2ND ARG TO POWER
207+         L     15,=V(FDXPD=)          A( DOUBLE PRE  POWER ROUTINE
208+         LA    1,APARG001            A( ARGUMENT LIST )
209+         BALR  14,15        FPRO = ARG1 ** ARG2
210+         STD   FPRO,C                 A SYMBOL
211                *,--- REG = FPRO WAS USED IN EVALUATING THE EQUATION
212+*                                                                    *
213+*                                                                    *
214+***** END ***** OF ***** EQUATION ***********************************
```

```
216          MATH  '( A + 4.0 )    **   ( B  *  .518 )  =  C  '

218+         LE    FPRO,A                 A SYMBOL
219+         AE    FPRO,=E'4.0'                NUMBER TYPE
220+         STE   FPRO,PARG0001               1ST  ARG  TO POWER
221+         LE    FPRO,B                 A SYMBOL
222+         ME    FPRO,=E'.518'                NUMBER TYPE
223+         STE   FPRO,PARG0002               2ND ARG TO POWER
224+         L     15,=V(FRXPR=)          A( REAL*4 POWER ROUTINE )
225+         LA    1,APARG001            A( ARGUMENT LIST )
226+         BALR  14,15        FPRO = ARG1 ** ARG2
227+         STE   FPRO,C                 A SYMBOL
228                *,--- REG = FPRO WAS USED IN EVALUATING THE EQUATION
229+*                                                                    *
230+*                                                                    *
231+***** END ***** OF ***** EQUATION ***********************************
```

RULES MATH USES IN PROCESSING AN EXPRESSION

1. All processing is performed left to right.

2. Each time an INNER-EXP is encountered, the following steps take place:

   a. An EXP-REG is determined for this INNER-EXP.

   b. The INNER-EXP is evaluated in this EXP-REG.

   c. Any special operation, specified by a prefix on the INNER-EXP, is performed on its EXP-REG.

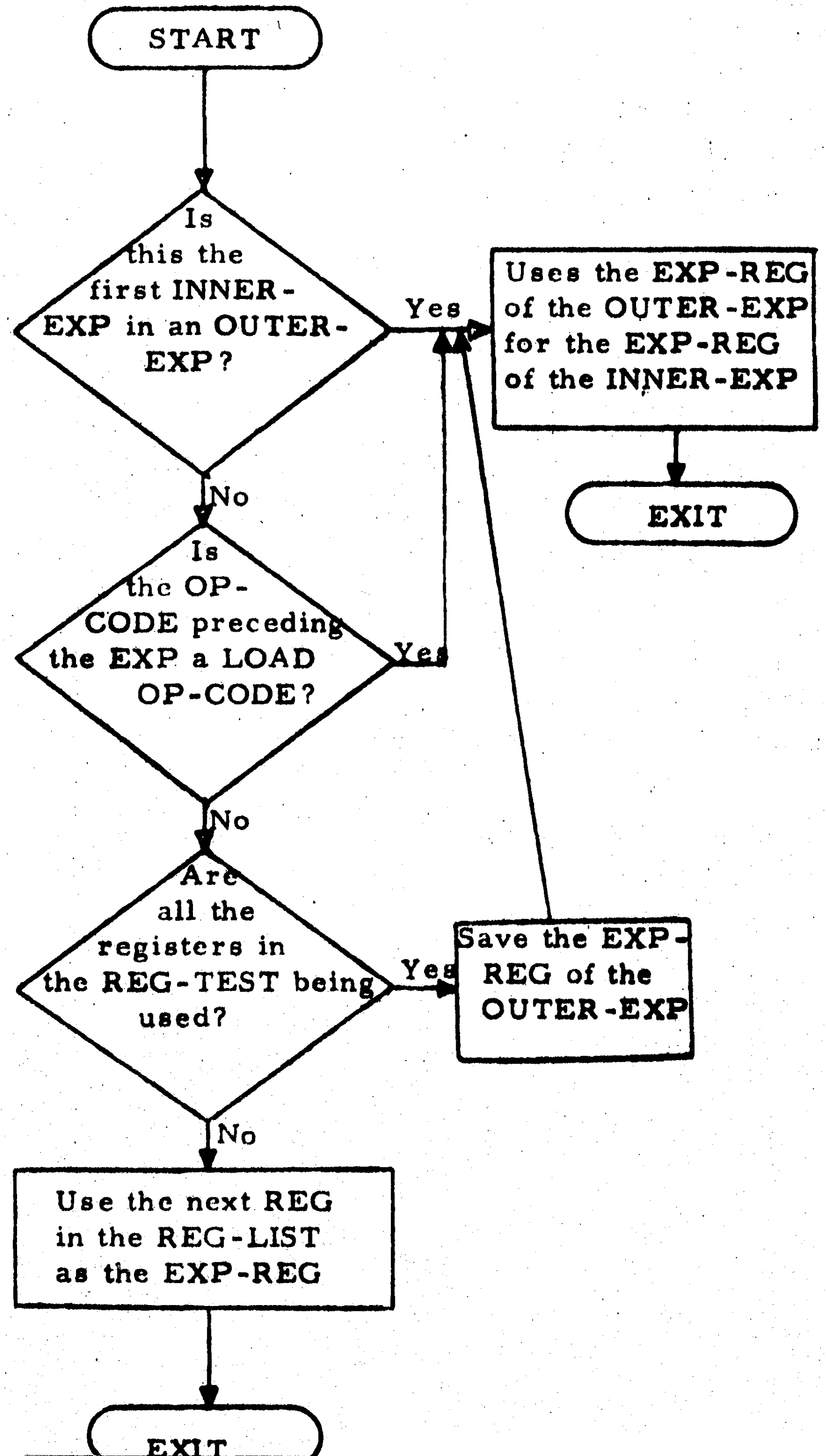
   d. The EXP-REG is used as the operand for the OP-CODE preceding the INNER-EXP.

3. The first OPERAND in each expression is loaded into its EXP-REG unless the first operand is a REG which has the same character structure as the EXP-REG.

   Example: If, in the following expression, the macro was specified as:

```
MATH ' (4) + A * ( (FPR2) + (6) )', REG = (4, FPR2), TYP=D
          |
          | code generated
          v
         AD     4,A
         ADR    FPR2,6
         MDR    4,FPR2
```

4. In determining which register will be the EXP-REG for an expression, it follows the procedure below:


START
Is this the first INNER-EXP in an OUTER-EXP?
Yes
No
Uses the EXP-REG of the OUTER-EXP for the EXP-REG of the INNER-EXP
EXIT
Is the OP-CODE preceding the EXP a LOAD OP-CODE?
Yes
No
Are all the registers in the REG-TEST being used?
Yes
Save the EXP-REG of the OUTER-EXP
No
Use the next REG in the REG-LIST as the EXP-REG
EXIT

5. The character from the TYP parameter (see the macro definition below) is used in forming all instructions.

   Example: If, in the following EXP, the EXP-REG was 0, then the following code would be generated for each TYP.

```
EXP = ( A * B = C )

TYP = E:     LE     0,A
             ME     0,B
             STE    0,C

TYP = H:     LH     0,A
             MH     0,B
             STH    0,C

TYP =        L      0,A
             M      0,B
             ST     0,C
```

MATH MACRO DEFINITION

[Symbol]	MATH	MAIN-EXP [, REG=register list] [, TRACE=ON or OFF] [, ANS = where to put answer] [TYP=character or null]

[ ] - optional

MAIN-EXP - as described on page 3-6 . There may be a maximum of 255 characters in a MAIN-EXP.

TYP - the type of instruction to be generated. (The character to be in each instructions, i.e., E, D, H, null, etc.) Default is TYP = E.

TRACE - On causes the expression and its EXP-REG to be printed when the last OP-CODE in the EXP has been processed. Default is TRACE=OFF. These intermediate expressions can sometimes make following the generated code much easier.

ANS - any valid SYMBOL or REG.

Default is to leave the answer in the first register specified in the REG-LIST.

REG - a single register label or number, or a sublist of one or more register labels or numbers.

Default: REG = (FPRO, FPR2, FPR4, FPR6)

NOTE: The registers specified in this REG-LIST tell the MATH macro which registers it can use to do the calculations in, and what order to use the registers in as it needs new EXP-REGs.

PROGRAMMING NOTES

1. All Equate and Branch OP-CODES must be followed by a valid SYMBOL.

2. The Equate OP-CODE causes the SYMBOL following the OP-CODE to be equated to the address of the next instruction.

3. The Branch-on-condition OP-CODE causes an immediate generation of the same branch on condition to the SYMBOL following the OP-CODE.

4. The HALF and SQAR prefix is invalid in the fixed point mode.

5. The multiply and divide OP-CODES are not valid in the fullword fixed point mode.

6. The XOR, AND, and OR OP-CODES are invalid in the floating point mode.

7. The Divide OP-CODE is invalid in all fixed point modes.

8. Whenever a STORE-OP-CODE appears in an EXP, the value currently in the EXP-REG is stored in the TERM following the OP-CODE for later use in the program.

9. Whenever a LOAD-OP-CODE appears in an EX, it causes the EXP-REG to be loaded with the next OPERAND in the EXP.

10. It should be noted that the hierarchy of operations which exist in fortran does not exist in MATH.

    Example:

    Fortran instruction: C = A - B / D
    MATH equivalent: 'A - (B / D) = C'
    or COMP (B / B) + A = C'

11. Since MATH is only an interpreter and not a compiler, it can only do what it is told in the same order it is told to do it. Therefore, if proper care is taken in arranging the operations in a floating point MATH expression, the floating point operations generated will be as tight as can be generated by coding each instruction separately. The following is an example of how to code tighter in MATH. Both of the following MATH expressions will do the same thing except the second expression requires one register instead of two like the first, and the second expression requires one less instruction.

    Example 1: MATH ' C * (A + B) = D'
    Example 2: MATH ' A + B * C = D'

12. The MATH macro may need storage space whenever it runs out of registers and encounters another level of INNER-EXP. For this reason, MATH will set up, and keep track of, any save areas it needs. There will be special labels on these save areas as follows:

    FRSAVE0N N = 1—> 9 and MATHTSVR

    These labels should not be used in any assembly in which the MATH Macro is used.

13. Though I have tried to list most of the major uses and limitations of MATH, I am sure there still exist several other possible uses and probably still more limitations. However, once the basic mechanics of MATH are fully understood, both its faults and attributes should become almost obvious.

14. MATH will perform special error checking for conditions not checked by the assembler. When it encounters one of the errors, it will flag it with a MNOTE statement having a condition code of 12.

15. The Branch-on condition OP-CODES will accept any combination of eight or less characters as a valid address and let the assembler perform the error checking on them.

EXAMPLES

The following is an example of an expansion of the MATHSAVE macro: of MATH's OP-CODES and OPERANDS.

```
TEST1    MATH   ' A * B / (2) + A+4 - 99'

  MATH  ' DUBL( A) - B'

TEST3    MATH   ' A+8 * B(5)',ANS=(FPR6)

TEST4    MATH   ' A+4 / B',ANS=B+8($5)

TEST5    MATH   ' A * ( B - A * ( A+8 - B ) )'

TEST6    MATH   ' A * ( B - A * ( A+8 - 10))',REG=2

TEST7    MATH   ' (2) / (4) * (6) + .01 - A(5)',REG=0,ANS=B,TRACE=OFF

TEST8    MATH   '    A     * ( 1 +  A  * ( 1 + A  *    ( 1 + A *              X
                ( 1    +  A ))))',ANS=B,TRACE=OFF,TYP=D

TEST9    MATH   '  A  - ( A + B - A+8($5)    *  .0199E-24 )',ANS=(FPR6)

TEST10   MATH   ' ABS( A - B) - 1.99'

TEST11   MATH   ' A - ABS( A - B + (2) )'

         MATH   ' NEG( A - B PLUS (2) ) PLUS A',TRACE=OFF,TYP=
TEST12   MATH   '        A  +  ABS(  A - ABS( A - B))'

         MATH   'ABS( NEG( A MINUS B) + A) PLUS A'

TEST13   MATH   ' A / ABS( A - B  ) + ABS( ( B - 100 ) / ( A - .99
                   ))',REG=4,ANS=B

TEST14   MATH   'A +    ABS(   A   -  B  -  100)      +
                (    A  *  A  *  A * A * A)',REG=6

         MATH   'SQAR( SQAR( A)) TIMES A + ABS( A - B - 100) + A',REG=6
```

```
TESTALL  MATH  '1 + ABS( A / B )  +                                     X
                 ABS( ( A - B) - ( A+4 - B+4) )   /                     X
                 ABS( ( A+16 - B+16 ) -  ( A+20 - B+20 ))',             X
               REG=(2,6),ANS=A+4

         MATH  '1 + ABS( A / B )  +                                     X
                 ABS( ( A - B) - ( A+4 - B+4) )   /                     X
                 ABS( ( A+16 - B+16 ) -  ( A+20 - B+20 ))',             X
               REG=3,ANS=A+4,TYP=D,TRACE=OFF

         MATH  ' ABS( ABS( A - B) - A)'

         MATH  ' NEG( A - B)'

         MATH  ' DUBL( A - B)'

         MATH  ' SQAR( A - B)'

         MATH  ' COMP( A - B)'

         MATH  ' HALF( A)'



         MATH  ' COMP( A)'

         MATH  ' DUBL( A)'

         MATH  'SYSPARM(MCRFMN) - 12 WITH A BNE ZERO * SYSPARM(MCCFCU) X
                 STORE B+8     ',TYP=D,TRACE=OFF,REG=FPR6

         MATH  'A / ( TEST( A - 100) BZ ZERO) * ( A - B) BZ ZERO'

   MATH 'A MINUS B($5) TIMES 100 OVER -400  PLUS (2) HERE= BP200        X
               BM ZERO LABEL= BP201 BZ ZERO STORE B+8 LOAD A - B = B'

         MATH  '  ABS(  A - ( ABS( A - B) - A      ) - B )  ',REG=6

         MATH  ' ABS( NEG( DUBL( A)) / COMP( HALF( B)) * SQAR( A))'

         MATH  ' ( (4) / (6) - (2)) - A + 10      ',REG=0

         MATH  '( A + B) C ( A - 299 ) BP POSITIVE BZ ZERO * -400'

         MATH  ' ( ( ( ( A + B  )))) - B+4'

         MATH  ' ( ( ( (4) ) ) )'

         MATH  'TEST( A - B($5)  * 100  / -400  * (2) )
```

```
TESTER   MATH  '   A+8 +  SQAR( A - B )  - SQAR( HALF( HALF( (6))))     X
                -  ABS(  A *  (FPR6)   ) +  ( ( A - B) / B(5) / -1000   X
               * ( A+16(5) - DUBL( A - .001)  +  ABS( A -   .1E-10) ) *X
                COMP( DUBL(  A * B) - (FPR2))           EQU BP300    /  X
               NEG( HALF( SQAR( B * 9.5 )))))',REG=FPR4,ANS=(FPR4),     X
               TYP=D,TRACE=OFF

TESTER2  MATH  '   A+8 +  SQAR( A - B )  - SQAR( HALF( HALF( (6))))     X
                -  ABS(  A *  (FPR6)   ) +  ( ( A - B) / B(5) / -1000   X
               * ( A+16(5) - DUBL( A - .001)  +  ABS( A -   .1E-10) ) *X
                COMP( DUBL(  A * B) - (FPR2))           EQU BP301    /  X
               NEG( HALF( SQAR( B * 9.5 )))))',REG=(FPR4,FPR6),         X
               TYP=D,TRACE=OFF,ANS=(FPR4)

  MATH ' SQAR( SQAR( SQAR( SQAR( SQAR( SQAR( SQAR( B))))))))'   = B**128

         MATH  '( A * B = B+4 * B+8 = B+16 * B+20) C 100                X
               BH POSITIVE BE ZERO = B+24'


         MATH  '( A * B = B+4 * B+8 = B+16 * B+20) WITH 100             X
               BH POSITIVE BE ZERO = B+24',TRACE=OFF,TYP=H,REG=7

  MATH ' A PLUS B MINUS 100 OVER (2)  STORE B+4 TIMES SYSPARM(MCRFMN8)'

         MATH  ' SYSPARM(MKTYPE) ',REG=5,TYP=H

         MATH  'A / ( A+4 - 0.001 BNP ZERO .   10 * A)'

         MATH  '   A  * SYSPARM(MCRFMN) / SYSPARM(MHRSYT04)'

         MATH  '  A * SYSPARM(MCRFMN)  = B',TYP=D

         MATH  '  SYSPARM(MHRSYT16) STORE  A',TYP=,REG=5

         MATH  '  SQAR( SYSPARM(MCRFMN)) = A',TYP=D


         MATH  '  SQAR( SYSPARM(MCRFMN)) STORE A',TYP=D

         MATH  ' A - B BP BP1 . 100 EQU BP1 * ( A - SQAR( A)) = B+4'

         MATH  'A C 5 BE ZERO C 10 BE POSITIVE C 15 BE ZERO             X
                 C 20 BL   ZERO   = B LOAD 100 STORE B',TRACE=OFF

         MATH  ' TEST( A ) BZ ZERO C (3)  BE POSITIVE + B  = B',TYP=,   X
               REG=7
```

```
        MATH  'A + A+4 + A+8 + A+12 + A+16 = B',TYP=,REG=5

    MATH  ' SYSPARM(MGJBAT) LOAD  $0+16(3)  STORE-IN B',TYP=,REG=3

        MATH  ' A   HERE= ASDFASDF * ( A - B)'

        MATH  ' A     $    GHJKFGHJ * ( A - B)'

        MATH  ' A TO B BE ZERO'

        MATH  ' A COMPARE B BNE ZERO'

        MATH  ' A COMPARE B BNE ZERO',TYP=D

        MATH  ' A COMPARE B BNE ZERO',TYP=

        MATH  '  A - B BNM BP8 - B    LABEL= BP8 * (  A * B)'

        MATH  'A WITH B BNE NOTEQ = B B ZERO LABEL= NOTEQ LOAD =F''100X
              '' STORE B B ZERO',TYP=,TRACE=OFF

        MATH  ' A AND B BZ ZERO OR ( A + B)  XOR =X''OFOFOFOF''       X
               SAVED-IN B+4  WITH A BE ZERO OR (3)  XOR (5)  = B ',   X
              REG=(7,9),TRACE=OFF,TYP=


        MATH  ' (FPRO) * (2) OVER ( (4) - A)',REG=(FPRO,4)
```

The following examples are expansions of some of the above MATH expressions:

```
1240          MATH  'A / ( A+4 - 0.001 BNP ZERO .   10 * A)'

1242+         LE    FPRO,A                 A SYMBOL
1243+         LE    FPR2,A+4                 A SYMBOL
1244+         SE    FPR2,=E'0.001'                NUMBER TYPE
1245+         BNP   ZERO                 BRANCH ON CONDITION
1246+         LE    FPR2,=E'10'                NUMBER TYPE
1247+         ME    FPR2,A                 A SYMBOL
1248                *,----------- REG = FPR2     NOW CONTAINS -------------*X
                     ' A+4 - 0.001 BNP ZERO .   10 * A'
1249+*              *------------------------------------------------------*
1250+         DER   FPRO,FPR2
1251                *,----------- REG = FPRO     NOW CONTAINS -------------*X
                     'A / ( A+4 - 0.001 BNP ZERO .   10 * A)'
1252+*              *------------------------------------------------------*
1253                *,--- REG = FPRO WAS USED IN EVALUATING THE EQUATION
1254                *,--- REG = FPR2 WAS USED IN EVALUATING THE EQUATION
1255+*                                                                     *
1256+*                                                                     *
1257+***** END ***** OF ***** EQUATION **************************************
```

```
1229            MATH   ' SYSPARM(MKTYPE) ',REG=5,TYP=H

1231+           L      1,=V(MKTYPE)             LOAD REG1 WITH ADDR OF SYSPARAM
1232+           LH     5,0(1)               OR USING VALUE OF SYSPARM
1233                   *,----------- REG = 5          NOW CONTAINS -------------*X
                        ' SYSPARM(MKTYPE) '
1234+*                 *-------------------------------------------------------*
1235                   *,--- REG = 5 WAS USED IN EVALUATING THE EQUATION
1236+*                                                                         *
1237+*                                                                         *
1238+***** END ***** OF ***** EQUATION ***************************************
```

```
1628            MATH   ' A AND B BZ ZERO OR ( A + B)  XOR =X''OFOFOFOF''
                        SAVED-IN B+4  WITH A BE ZERO OR (3)  XOR (5)  = B ',
                       REG=(7,9),TRACE=OFF,TYP=

1630+           L      7,A                 A SYMBOL
1631+           N      7,B                 A SYMBOL
1632+           BZ     ZERO                  BRANCH ON CONDITION
1633+           L      9,A                 A SYMBOL
1634+           A      9,B                 A SYMBOL
1635+           OR     7,9
1636+           X      7,=X'OFOFOFOF'                A SYMBOL
1637+           ST     7,B+4                 A SYMBOL
1638+           C      7,A                 A SYMBOL
1639+           BF     ZERO                  BRANCH ON CONDITION
1640+           OR     7,3                    A REG TYPE
1641+           XR     7,5                    A REG TYPE
1642+           ST     7,B                 A SYMBOL
1643                   *,--- REG = 7 WAS USED IN EVALUATING THE EQUATION
1644                   *,--- REG = 9 WAS USED IN EVALUATING THE EQUATION
1645+*                                                                         *
1646+*                                                                         *
1647+***** END ***** OF ***** EQUATION ***************************************
```

```
60 TEST1     MATH   ' A * B / (2) + A+4 - 99'                                      0


62+FPR0      EQU    0                 ***
63+FPR2      EQU    2                   ***   SET UP EQUATES FOR THE
64+FPR4      EQU    4                   ***    FLOATING POINT REGS
65+FPR6      EQU    6                 ***


67+TEST1     EQU    *

69+          LE     FPR0,A                 A SYMBOL
70+          ME     FPR0,B                 A SYMBOL
71+          DER    FPR0,2                    A REG TYPE
72+          AE     FPR0,A+4                A SYMBOL
73+          SE     FPR0,=E'99'                NUMBER TYPE
74                  *,----------- REG = FPR0     NOW CONTAINS -------------*X
                     ' A * B / (2) + A+4 - 99'
75+*                *-----------------------------------------------------*
76                  *,--- REG = FPR0 WAS USED IN EVALUATING THE EQUATION
77+*                                                                      *
78+*                                                                      *
79+***** END ***** OF ***** EQUATION **************************************




1260           MATH   '   A  * SYSPARM(MCRFMN) / SYSPARM(MHRSYT04)             0

1262+          LE     FPR0,A                A SYMBOL
1263+          L      1,=V(MCRFMN)            LOAD REG1 WITH ADDR OF SYSPARAM
1264+          ME     FPR0,0(1)               OP USING VALUE OF SYSPARM
1265+          L      1,=V(MHRSYT)            LOAD REG1 WITH ADDR OF SYSPARAM
1266+          DE     FPR0,04(1)               OP USING VALUE OF SYSPARM
1267                  *,----------- REG = FPR0     NOW CONTAINS -------------*X
                       '   A  * SYSPARM(MCRFMN) / SYSPARM(MHRSYT04)'
1268+*                *-----------------------------------------------------*
1269                  *,--- REG = FPR0 WAS USED IN EVALUATING THE EQUATION
1270+*                                                                      *
1271+*                                                                      *
1272+***** END ***** OF ***** EQUATION **************************************
```

```
857          MATH   ' ABS( NEG( DUBL( A)) / COMP( HALF( B)) * SQAR( A))'      0

859+         LE     FPR0,A                A SYMBOL
860                 *,----------- REG = FPR0     NOW CONTAINS -------------*X
                      ' A'
861+*               *--------------------------------------------------------*
862+         AER    FPR0,FPR0             SPECIAL OPERATION
863                 *,----------- REG = FPR0     NOW CONTAINS -------------*X
                      ' DUBL( A)'
864+*               *--------------------------------------------------------*
865+         LNER   FPR0,FPR0             SPECIAL OPERATION
866+         LE     FPR2,B                A SYMBOL
867                 *,----------- REG = FPR2     NOW CONTAINS -------------*X
                      ' B'
868+*               *--------------------------------------------------------*
869+         HER    FPR2,FPR2             SPECIAL OPERATION
870                 *,----------- REG = FPR2     NOW CONTAINS -------------*X
                      ' HALF( B)'
871+*               *--------------------------------------------------------*
872+         LCER   FPR2,FPR2             SPECIAL OPERATION
873+         DER    FPR0,FPR2
874+         LE     FPR2,A                A SYMBOL
875                 *,----------- REG = FPR2     NOW CONTAINS -------------*X
                      ' A'
876+*               *-------------------------------------------  --------*
877+         MER    FPR2,FPR2             SPECIAL OPERATION
878+         MER    FPR0,FPR2
879                 *,----------- REG = FPR0     NOW CONTAINS -------------*X
                      ' NEG( DUBL( A)) / COMP( HALF( B)) * SQAR( A)'
880+*               *--------------------------------------------------------*
881+         LPER   FPR0,FPR0             SPECIAL OPERATION
882                 *,----------- REG = FPR0     NOW CONTAINS -------------*X
                      ' ABS( NEG( DUBL( A)) / COMP( HALF( B)) * SQAR( A))'
883+*               *--------------------------------------------------------*
884                 *,--- REG = FPR0 WAS USED IN EVALUATING THE EQUATION
885                 *,--- REG = FPR2 WAS USED IN EVALUATING THE EQUATION
886+*                                                                        *
887+*                                                                        *
888+***** END ***** OF ***** EQUATION ****************************************
```

```
1435        MATH  'A + A+4 + A+8 + A+12 + A+16 = B',TYP=,REG=5                 0

1437+       L     5,A               A SYMBOL
1438+       A     5,A+4               A SYMBOL
1439+       A     5,A+8               A SYMBOL
1440+       A     5,A+12               A SYMBOL
1441+       A     5,A+16               A SYMBOL
1442+       ST    5,B               A SYMBOL
1443              *,----------- REG = 5       NOW CONTAINS -------------*X
                   'A + A+4 + A+8 + A+12 + A+16 = B'
1444+*            *----------------------------------------------------*
1445              *,--- REG = 5 WAS USED IN EVALUATING THE EQUATION
1446+*                                                                 *
1447+*                                                                 *
1448+***** END ***** OF ***** EQUATION *************************************
```

```
1213    MATH ' A PLUS B MINUS 100 OVER (2)  STORE B+4 TIMES SYSPARM(MCRFMN8)' C

1215+       LE    FPRO,A                 A SYMBOL
1216+       AE    FPRO,B                 A SYMBOL
1217+       SE    FPRO,=E'100'               NUMBER TYPE
1218+       DER   FPRO,2                  A REG TYPE
1219+       STE   FPRO,B+4               A SYMBOL
1220+       L     1,=V(MCRFMN)           LOAD REG1 WITH ADDR OF SYSPARAM
1221+       ME    FPRO,8(1)              OP USING VALUE OF SYSPARM
1222              *,----------- REG = FPRO     NOW CONTAINS -------------*X
                   ' A PLUS B MINUS 100 OVER (2)  STORE B+4 TIMES SYSPARM(X
                  MCRFMN8)'
1223+*            *----------------------------------------------------*
1224              *,--- REG = FPRO WAS USED IN EVALUATING THE EQUATION
1225+*                                                                 *
1226+*                                                                 *
1227+***** END ***** OF ***** EQUATION *************************************
```

```
445 TESTALL  MATH   '1 + ABS( A / B )   +                                         XC
                      ABS( ( A - B) - ( A+4 - B+4) )     /                        XC
                      ABS( ( A+16 - B+16 ) -  ( A+20 - B+20 ))',                   XC
                    REG=(2,6),ANS=A+4                                              C
446+TESTALL  EQU    *

448+         LE     2,=E'1'               NUMBER TYPE
449+         LE     6,A               A SYMBOL
450+         DE     6,B               A SYMBOL
451                 *,----------- REG = 6         NOW CONTAINS -------------*X
                     ' A / B '
452+*               *--------------------------------------------------------*
453+         LPER   6,6          SPECIAL OPERATION
454+         AER    2,6
455+         LE     6,A               A SYMBOL
456+         SE     6,B               A SYMBOL
457                 *,----------- REG = 6         NOW CONTAINS -------------*X
                     ' A - B'
458+*               *--------------------------------------------------------*
459+         STE    6,FRSAVE01     SAVE REGS CONTENTS
460+         LE     6,A+4              A SYMBOL
461+         SE     6,B+4              A SYMBOL
462                 *,----------- REG = 6         NOW CONTAINS -------------*X
                     ' A+4 - B+4'
463+*               *--------------------------------------------------------*
464+         STE    6,MATHTSVR         SPECIAL SAVE FOR NON-COMMUTE OPS
465+         LE     6,FRSAVE01          RETRIEVE SAVED DATA
466+         SE     6,MATHTSVR           PERFORM OPERATION
467                 *,----------- REG = 6         NOW CONTAINS -------------*X
                     ' ( A - B) - ( A+4 - B+4) '
468+*               *--------------------------------------------------------*
469+         LPER   6,6          SPECIAL OPERATION
470+         AER    2,6
471+         LE     6,A+16               A SYMBOL
472+         SE     6,B+16               A SYMBOL
473                 *,----------- REG = 6         NOW CONTAINS -------------*X
                     ' A+16 - B+16 '
474+*               *--------------------------------------------------------*
475+         STE    6,FRSAVE01     SAVE REGS CONTENTS
476+         LE     6,A+20               A SYMBOL
477+         SE     6,B+20               A SYMBOL
478                 *,----------- REG = 6         NOW CONTAINS -------------*X
                     ' A+20 - B+20 '
479+*               *--------------------------------------------------------*
480+         STE    6,MATHTSVR         SPECIAL SAVE FOR NON-COMMUTE OPS
481+         LE     6,FRSAVE01          RETRIEVE SAVED DATA
482+         SE     6,MATHTSVR           PERFORM OPERATION
483                 *,----------- REG = 6.        NOW CONTAINS -------------*X
                     ' ( A+16 - B+16 ) -  ( A+20 - B+20 )'
484+*               *--------------------------------------------------------*
485+         LPER   6,6          SPECIAL OPERATION
486+         DER    2,6
487                 *,----------- REG = 2         NOW CONTAINS -------------*X
                     '1 + ABS( A / B )  +                                          X
                                                                                   X
```

CONCLUSION

The number of possible combinations of options, OPERANDS, and OP-CODES is too large to discuss each one, even briefly. Therefore, as in learning any new language, probably the best way to learn how to write expressions is to use the definition and examples as a guide in coding up a few test cases.

71

MATH REFERENCE INFORMATION

OP-CODE	OPERATION	OP-CODE	OPERATION
+, PLUS	ADD	C, WITH	COMPARE
-, MINUS	SUBTRACT	TO, COMPARE	COMPARE
/, OVER	DIVIDE		
*, TIMES	MULTIPLY	., LOAD	LOAD
**	POWER ROUTINE	RELOAD	LOAD
=, SAVED-IN	STORE		
STORE, STORE-IN	STORE	XOR	Exclusive OR
OR	OR	AND	AND

OP-CODE	OPERATION
$, HERE=	Place label on next instruction
EQU, LABEL=	Place label on next instruction
B	Branch on Condition 15
BH, BP	Branch on Condition 2
BL, BM	Branch on Condition 4
BE, BZ	Branch on Condition 8
BO	Branch on Condition 1
BNH, BNP	Branch on Condition 13
BNL, BNM	Branch on Condition 11
BNE, BNZ	Branch on Condition 7

VALID OPERAND

Any valid SYMBOL
Any valid LITERAL
Any valid REG
Any valid PREFIXED-EXP.

Any valid TERM
Any valid NUMBER
Any valid EXP

OPERAND	STARTING CHARACTERS
NUMBERS	. , - , 0 → 9
SYMBOL	A → Z, $, and @ (only 8 characters at maximum)
TERM	A → Z, $, and @ (any assembly language operand)
REG	"(" followed immediately by a symbol or number
EXP	"(" followed by at least one blank
LITERAL	= sign (like in assembly language except quote doubled)

VALID PREFIXES FOR EXP'S

ABS
NEG
TEST
COMP
HALF
DUBL
SQAR

SPECIAL TERM FOR SYSTEM PARAMETERS

SYSPARM (MXXXXXNN)
MXXXXX = SYSPARM NAME
NN = null or 0 — 99

NAME - NIBIT

DESCRIPTION

The function of the NIBIT macro is to generate an AND IMMEDIATE instruction which utilizes the length code of the symbol specified to "turn off" a desired bit in a byte.

DEFINITION

[symbol]	NIBIT	Symbol

where Symbol is the label of a data base definition which has an associated length code.

EXPANSION

NAME	OPERATION	OPERAND
[symbol]	NIBIT	LABEL
+[symbol]	NI	LABEL,X'FF'-L'LABEL

GENERAL NOTES

- The NIBIT macro will be utilized most often in conjunction with the B IT macro, since BIT generates a desired length code associated with a valid label.

NAME - OIBIT

DESCRIPTION

(See XIBIT)

NAME - BIT

DESCRIPTION

The purpose of the BIT macro is to generate a data base definition whose length can be used as a key to test or manipulate a specific bit in a byte.

DEFINITION

symbol	BIT	{Bit number, or list of bit numbers, or binary 8-bit configuration} [,ON]

where

- **symbol --** any valid non-blank label. If omitted, an error condition will be raised with a condition code of 12.
- **bit number --** an unsigned decimal integer, 0 through 7, representing standard bit notation.
- **list of bit numbers -- a list of bit numbers separated by commas. The entire list must be enclosed by parenthesis.**
- **binary 8-bit configuration --** notation of the form B'XXXXXXXX', **where X is 1 if the corresponding bit is to be** represented by this label and X is 0 if the corresponding bit is not to be represented by this label.
- **ON -- indicates the bit or bits indicated in the first operand are to set to 1 in a global variable which is passed to the BYTE macro.**

FUNCTION

The BIT macro performs its operations as follows:

- checks to see if there is a valid non-blank label attached to the macro.
- processes the information passed by the first operand, checking each time for an invalid bit number or binary character.
- generates a DS and ORG statement to establish a length which can be used to test or manipulate bit(s), and reset the location counter setting. (There is an exception to this -- if the name of the CSECT currently being processed starts with SCDB, the DS and ORG statement will not be generated.

EXAMPLES OF THE USE

The following are included to give the user a feeling of what can and cannot be done with the BIT macro:

Example 1

NAME	OPERATION	OPERANDS
FIRST	BIT	0
+FIRST	DS	XL(B'10000000')
+	ORG	*-B'10000000'

Example 2

NAME	OPERATION	OPERANDS
SECOND	BIT	(0, 1, 5, 7), ON
+SECOND	DS	XL(B'11000101')
+	ORG	*-B'11000101'

Note: In the above example, specifying 'ON' had no effect upon the expansion of the macro.

NAME - ORELSE

DESCRIPTION

The function of the ORELSE macro is to generate the branch and labels that correspond with the branch instructions generated by the EXITIF macro and the labels generated by the ENDLOOP macro. See the STRTSRCH macro.

NAME - STRTSRCH

DESCRIPTION

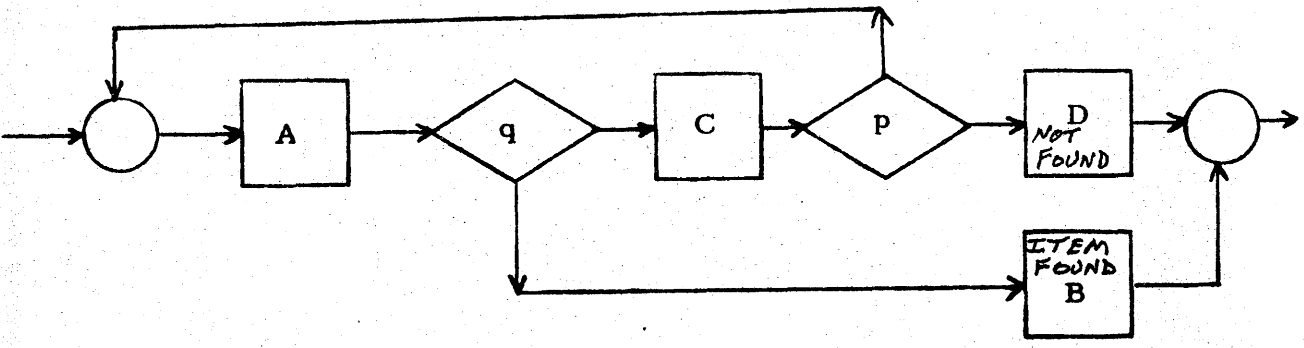
The search macros are used to generate the logic which is typical to what a programmer does when he sets up a loop to search through a table. The programmer's intent is to exit when he finds what he is searching for and perform process B. If he does not find what he is looking for, he executes process D before joining the alternate path. The ORELSE is optional and if it is omitted, box C does not appear in the flowchart. The following shows the format of the STRTSRCH format.

$$\text{STRTSRCH} \quad \left\{\begin{matrix}\text{WHILE}\\ \text{UNTIL}\end{matrix}\right\} , \quad \text{(condition),} \quad \left\{\begin{matrix}\text{OR}\\ \text{AND}\\ \text{DO}\end{matrix}\right\} \left[\text{, REG=}\right]$$

The STRTSRCH macro used the WHILE/UNTIL field to generate a WHILE or UNTIL macro statement. The condition format is the same as the WHILE and UNTIL macro.

EXAMPLE

```
STRTSRCH condition p
   Process A
EXITIF condition q
   Process B
ORELSE
   Process C
ENDLOOP
   Process D
ENDSRCH
```



Note:

When using these macros care should be taken not to confuse the ENDLOOP and ENDSRCH macros. The ENDLOOP is used to define the end of the loop and the ENDSRCH indicates the end of the complete macro set.

If a programmer is nesting these macros, he must be certain that each macro set is completely embedded within the process boxes of the higher level ones. If the user does not do this the following sequence of code would generate incorrect branching because of the manner in which the stacks are manipulated.

NAME - TMBIT

PURPOSE

The function of the TMBIT macro is to generate a test under mask instruction which utilizes the length code of the symbol to be tested as the mask byte.

DEFINITION

[symbol]	TMBIT	Symbol

where Symbol is the label of a data base definition which has an associated length code.

EXPANSION

NAME	OPERATION	OPERAND
[symbol]	TMBIT	LABEL
+[symbol]	TM	LABEL, L'LABEL

GENERAL NOTES

- The TMBIT macro will be utilized most often in conjunction with the BIT macro, since BIT generates a desired length code associated with a valid label.

NAME - UNTIL

DESCRIPTION

The function of the UNTIL macro is to generate the labels and instructions that branch to these labels to accomplish the programming function of iteration. The UNTIL macro supports both instruction for incrementing/decrementing indexes and instructions for terminating the loop based upon a change in a logical condition. The UNTIL statements support loops in which the indexing/condition-testing instructions are executed after the first pass through the code-body.

The UNTIL MACRO specifications: There are three difference UNTIL statements, the UNTIL-DO, UNTIL-OR-DO, and the UNTIL-AND-DO. For the flowcharts of the UNTIL statements, see the ENDDO macro writeup.

The general format for the UNTIL-DO is:

a. Indexed - UNTIL-DO:

```
UNTIL (index-instructions), DO
  code-body
ENDDO
```

which reads "UNTIL the following index-instructions fail to branch, continue to execute the code-body."

b. Logical - UNTIL-DO:

```
UNTIL (condition), DO

code-body

ENDDO
```

which reads "UNTIL the following conditions are true, continue to execute the code-body."

The general format for the UNTIL-OR-DO is:

```
UNTIL (index-instruction), OR
UNTIL (index-instruction), DO

code-body

ENDDO

UNTIL (condition), OR
UNTIL (condition), DO

code-body

ENDDO

UNTIL (index-instruction), OR
UNTIL (condition), DO

cody-body

ENDDO
```

The general format for the UNTIL-AND-DO is:

UNTIL (index-instruction), AND
UNTIL (index-instructions), DO

code-body

ENDDO

UNTIL (condition), AND
UNTIL (condition), DO

code-body

ENDDO

UNTIL (index-instruction), AND
UNTIL (condition), DO

code-body

ENDDO

The following shows the format of UNTIL:

$$\text{UNTIL}\ \left(\begin{bmatrix}\text{BCT}\\ \text{BXH}\\ \text{BXLE}\\ *\\ \text{BIT}\\ \text{B}\\ \text{H}\\ \text{F}\\ \text{E}\\ \text{D}\\ \text{C}\\ \text{T}\end{bmatrix},\begin{matrix}\text{(R1)}\\ \text{LABEL1}\end{matrix},\begin{Bmatrix}\text{(R3)}\\ \text{IS}\\ \text{GT}\\ \text{LT}\\ \text{GE}\\ \text{LE}\\ \text{EQ}\\ \text{NE}\end{Bmatrix},\begin{pmatrix}\text{(R2)}\\ \text{LABEL2}\\ \text{ONE(S)}\\ \text{OVERFLOW}\\ \text{PLUS}\\ \text{MINUS}\\ \text{MIXED}\\ \text{ZERO(S)}\\ \text{NMINUS}\\ \text{NPLUS}\\ \text{NONE(S)}\\ \text{NZERO(S)}\\ \text{OFF}\\ \text{ON}\end{pmatrix}\right),\begin{Bmatrix}\text{OR}\\ \text{AND}\\ \text{DO}\end{Bmatrix}\ \left[\text{, REG=}\right]$$

$$\text{UNTIL}\ \left(\left[\text{TYPE}\right]\text{, LABEL, OPERATION, CONDITION}\right),\begin{Bmatrix}\text{OR}\\ \text{AND}\\ \text{DO}\end{Bmatrix}\ \left[\text{, REG=}\right]$$

INDEXED UNTIL

The different types are:

1. BCT:

   UNTIL (BCT,R1),$\left\{\begin{matrix}\text{OR}\\\text{AND}\\\text{DO}\end{matrix}\right\}$

   Example:

```
         UNTIL   (BCT,$1),DO
+LABEL1  EQU  *

   CODE-BODY

   ENDDO
+  BCT   $1,LABEL1
```

2. BXH and BXLE:

   UNTIL ($\left\{\begin{matrix}\text{BXH}\\\text{BXLE}\end{matrix}\right\}$,R1,R3),$\left\{\begin{matrix}\text{OR}\\\text{AND}\\\text{DO}\end{matrix}\right\}$

   Examples:

```
            UNTIL   (BXH,$1,$3),DO
   +LABEL1  EQU   *

   CODE-BODY

   ENDDO
+  BXLE  $1,$3,LABEL1
```

LOGICAL UNTIL

The different types are:

1. An * in the type field stands for the condition is already set. When using the * type, the Operation and Condition fields cannot be omitted.

   Example:

```
          UNTIL  (*,,IS,PLUS),DO
+LABEL   EQU   *

   CODE-BODY

   ENDDO
+  BC       13,LABEL
```

2. Bit type: will generate a test under mask. The only valid operation parameter is (IS).

$$\text{UNTIL (BIT,LABEL,IS,}\left\{\begin{array}{l}\text{ZERO}\\\text{ONE}\\\text{ON}\\\text{OFF}\\\text{MIXED}\\\text{NONE}\\\text{NZERO}\\\text{NMIXED}\end{array}\right\}\text{),}\left\{\begin{array}{c}\text{OR}\\\text{AND}\\\text{DO}\end{array}\right\}$$

   Example:

```
          UNTIL   (BIT,A,IS,ZERO),DO
+LABEL1   EQU     *

   CODE-BODY

   ENDDO
+  TM   A,L'A
+  BC   7,LABEL1
```

3. B Type

$$\text{UNTIL (B,LABEL1,IS,}\left\{\begin{matrix}\text{ZERO(S)}\\ \text{PLUS}\\ \text{MINUS}\\ \text{NPLUS}\\ \text{NMINUS}\\ \\ \text{NZERO(S)}\end{matrix}\right\}\text{),}\left\{\begin{matrix}\text{OR}\\ \text{AND}\\ \text{DO}\end{matrix}\right\}$$

$$\text{UNTIL (B,LABEL1,}\left\{\begin{matrix}\text{GT}\\ \text{LT}\\ \text{EQ}\\ \text{NE}\\ \text{GE}\\ \text{LE}\end{matrix}\right\}\text{,}\left\{\begin{matrix}\text{T'}\\ \text{L'}\\ \text{X'4F'}\\ \text{C'FF"}\\ \text{B'01'}\\ \text{LABEL2}\\ \text{(R2)}\end{matrix}\right\}\text{),}\left\{\begin{matrix}\text{OR}\\ \text{AND}\\ \text{DO}\end{matrix}\right\}\text{[,REG=]}$$

[REG=] DEFAULTS TO $0.

Examples:

```
        UNTIL  (B,A,IS,ZERO),DO
+LABEL  EQU    *

   CODE-BODY

   ENDDO
+  CLI        A,X'00'
+  BC         7,LABEL

         UNTIL  (B,A,EQ,AAAAAAAA+16),DO
+LABEL   EQU    *

   CODE-BODY
```

```
        ENDDO
+       IC         $0,AAAAAAAA+16
+       STC        $0,*+5
+       CLI        A,X'00'
+       BC         7,LABEL

           UNTIL    (B,A,EQ,($1)),DO
+LABEL      EQU      *

        CODE-BODY

        ENDDO
+       STC        $1,*+5
+       CLI        A,X'00'
+       BC         7,LABEL
```

These B forms of the UNTIL statement alters executable code and are not usable if the program is to be reentrant.

```
        UNTIL  (B,ABLE,EQ,BAKER),DO
+L1     DS   0H

        CODE-BODY

        ENDDO
+       CLC        0+ABLE,BAKER
+       BC         8,L1

        UNTIL  (B,ABLE,EQ,($1)),DO
+L1     DS         0H

        CODE-BODY

        ENDDO
+       EX         $1,*+8
+       B          *+8
+       CLI        ABLE,0
+       BC         8,L1
```

Reentrant B TYPE

```
              UNTIL  (B,A,GT,138),DO
+LABEL  EQU    *

        CODE-BODY

        ENDDO
+       CLI      A,138
+       BC       13,LABEL

          UNTIL  (B,A,GT,0+MUD),DO
+LABEL    EQU   *

        CODE-BODY

        ENDDO
+       CLI      A,0+MUD
+       BC       13,LABEL

           UNTIL  (B,A,GT,X'4F'),DO
+LABEL   EQU   *

        CODE-BODY

        ENDDO
+       CLI      A,X'4F'
+       BC       13,LABEL
```

4. **Fixed-Point (H or F)**

$$\text{UNTIL} \;\; \left(\left\{\begin{matrix}\text{H}\\ \text{F}\end{matrix}\right\}, \begin{matrix}\text{(R1)}\\ \text{LABEL1}\end{matrix}, \text{IS}, \left\{\begin{matrix}\text{ONE(S)}\\ \text{PLUS}\\ \text{MINUS}\\ \text{ZERO(S)}\\ \text{NZERO(S)}\\ \text{NMINUS}\\ \text{NPLUS}\\ \text{NONE(S)}\end{matrix}\right\}\right), \left\{\begin{matrix}\text{OR}\\ \text{AND}\\ \text{DO}\end{matrix}\right\} \left[\text{, REG=}\right]$$

$$\text{UNTIL} \;\; \left(\left\{\begin{matrix}\text{H}\\ \text{F}\end{matrix}\right\}, \begin{matrix}\text{(R1)}\\ \text{LABEL1}\end{matrix}, \left\{\begin{matrix}\text{GT}\\ \text{LT}\\ \text{GE}\\ \text{EQ}\\ \text{NE}\\ \text{LE}\end{matrix}\right\}, \left\{\begin{matrix}\text{LABEL2}\\ \text{(R2)}\\ \text{=F'\quad'}\\ \text{=H'\quad'}\\ \text{=X'\quad'}\\ \text{=C'\quad'}\end{matrix}\right\}\right), \left\{\begin{matrix}\text{OR}\\ \text{AND}\\ \text{DO}\end{matrix}\right\} \left[\text{, REG=}\right]$$

REG= Defaults to $0.

```
        UNTIL  (H,A,IS,PLUS),DO
+LABEL   EQU    *

         CODE-BODY

         ENDDO
+        LH     $0,A
+        LTR    $0,$0
+        BC     2, LABEL

         UNTIL  (H,($1),IS,ZERO),DO
+LABEL    EQU    *

         CODE-BODY

         ENDDO
+        LTR     $1,$1
+        BC      7,LABEL
```

```
          UNTIL   (H,A,GT,B),DO
+LABEL     EQU    *

           CODE-BODY

           ENDDO
+          LH     $0,A
+          CH     $0,B
+          BC     13,LABEL

          UNTIL   (H,A,EQ,($1)),DO
+LABEL     EQU    *

           CODE-BODY

           ENDDO
+          CH     $1,A
+          BC     7,LABEL

          UNTIL   (H,($1),EQ,($2)),DO
+LABEL     EQU    *

           CODE-BODY

           ENDDO
+          CR     $1,$2
+          BC     7,LABEL

           UNTIL  (F,A,IS,PLUS),DO
+LABEL     EQU    *

           CODE-BODY

           ENDDO
+          L      $0,A
+          LTR    $0,$0
+          BC     13,LABEL
```

```
         UNTIL    (F,($1),IS,ZERO),DO
+LABEL    EQU      *

          CODE-BODY

          ENDDO
+         LTR      $1,$1
+         BC       7,LABEL

         UNTIL    (F,A,GT,B),DO
+LABEL    EQU      *

          CODE-BODY

          ENDDO
+         L        $0,A
+         C        $0,B
+         BC       13,LABEL

         UNTIL    (F,($1),GT,B),DO
+LABEL    EQU      *

          CODE-BODY

          ENDDO
+         C        $1,B
+         BC       13,LABEL

         UNTIL    (F,A,EQ,($1)),DO
+LABEL    EQU      *

          CODE-BODY

          ENDDO
+         C        $1,A
+         BC       7,LABEL
```

```
         UNTIL    (F,($1),EQ,($2)),DO
+LABEL    EQU      *

          CODE-BODY

          ENDDO
+         CR       $1,$2
+         BC       8,LABEL
```

5. **Floating Point (E or D)**

```
                             ( ONE(S)   )
                             ( PLUS     )
                             ( MINUS    )      ( OR  )
             ( E )   (R1)    ( ZERO(S)  )      ( AND )
    UNTIL   (( D ),LABEL1,IS,( NZERO(S) )),    ( DO  )   [,REG=]
                             ( NMINUS   )
                             ( NPLUS    )
                             ( NONE(S)  )

                          ( GT )   ( LABEL2  )
                          ( LT )   ( (R2)    )       ( OR  )
             ( E )  (R1)  ( GE )   ( =E'   ' )       ( AND )
    UNTIL   (( D ) ,LABEL1,( EQ ) , ( =D'   ' ) ),   ( DO  )   [,REG=]
                          ( NE )
                          ( LE )
```

[REG=] Defaults to FPRO.

```
          UNTIL    (E,A,IS,PLUS),DO
+LABEL     EQU      *

          CODE-BODY

          ENDDO
+         LE       FPRO,A
+         LTER     FPRO,FPRO
+         BC       13,LABEL
```

```
         UNTIL   (D,(FPRO),IS,ZERO),DO
+LABEL    EQU     *

          CODE-BODY

          ENDDO .
+         LTDR    FPRO,FPRO
+         BC      7,LABEL

         UNTIL   (E,A,GT,B),DO
+LABEL    EQU     *

          CODE-BODY

          ENDDO
+         LE      FPRO,A
+         CE      FPRO,B
+         BC      13,LABEL

         UNTIL   (D,(FPRO),GT,B),DO
+LABEL    EQU     *

          CODE-BODY

          ENDDO
+         CD      FPRO,B
+         BC      13,LABEL

         UNTIL   (E,A,EQ,(FPR2)),DO
+LABEL    EQU     *

          CODE-BODY

          ENDDO
+         CE      FPR2,A
+         BC      7,LABEL
```

```
           UNTIL   (D,(FPRO),EQ,(FPR2)),DO
+LABEL      EQU     *

            CODE-BODY

            ENDDO
+           CDR     FPRO,FPR2
+           BC      7,LABEL
```

6. CHARACTER(C)

```
                             ⎧LT⎫
                             ⎪GT⎪              ⎧ OR ⎫
            UNTIL   (C,LABEL1,⎨GE⎬ ,LABEL2),  ⎨AND ⎬
                             ⎪EQ⎪              ⎩ DO ⎭
                             ⎪LE⎪
                             ⎩NE⎭
```

```
            UNTIL   (C,ABLE,EQ,BETA),DO
+LABEL       EQU    *

            CODE-BODY

            ENDDO
+           CLC     ABLE,BETA
+           BC      7,LABEL

            UNTIL   (C,=C'SED',EQ,0($3)),DO
+LABEL       EQU    *

            CODE-BODY

            ENDDO
+           CLC     =C'SED',0($3)
+           BC      7,LABEL
```

7. **Test Under Mask (T)**

$$\text{UNTIL (T, LABEL1,} \left\{ \text{MASK} \right\}, \left\{ \begin{array}{l} \text{ZERO} \\ \text{ONE} \\ \text{ON} \\ \text{OFF} \\ \text{MIXED} \\ \text{NONE} \\ \text{NZERO} \\ \text{NMIXED} \end{array} \right\} \text{),} \left\{ \begin{array}{l} \text{OR} \\ \text{AND} \\ \text{DO} \end{array} \right\}$$

```
          UNTIL  (T,A,X'11', ZERO), DO
+LABEL     EQU    *

          CODE-BODY
+         TM     A,X'11'
+         BC     7,LABEL
```

VIII. **Type Field Omitted:**

```
          UNTIL   (,A,IS,ZERO),DO
+LABEL    EQU     *

          CODE-BODY

          ENDDO
+         LE      FPRO,A
+         LTER    FPRO,FPRO
+         BC      7,LABEL



A         DC      E'0'
```

PROGRAMMING NOTES

Also see programming notes for IF macro.

1. "Index-Instruction" can be any one of the following:

   a. BCT,r1
   b. BXH,r1,r3
   c. BXLE,r1,r3

2. "Code-Body" can be any group of valid machine and/or macro instructions, including a maximum of twenty nested WHILE/UNTIL's. Multiple index-instructions in the same loop are also supported.

3. The expansion of the UNTIL macro causes the indexing and/or logical instructions to be assembled after the code-body and executed after the first pass through the code-body.

4. The level of a nested WHILE/UNTIL statement can be found in the LABELS that are generated.

```
          UNTIL       (condition),DO
+UN/5/xxxx  EQU       *
```

   The /5/ stands for the level of this nested UNTIL statement.

5. Any time a register notation is used in a logical-UNTIL statement, the register must be in parentheses. It does not make any difference whether a register is in parentheses or not with an Indexed-UNTIL statement.

6. Misspelling and abbreviation of "conditions" mnemonices is not allowed.

7. Restriction: Expressions cannot be over sixteen characters in length.

8. Reentrant programs that use the B TYPE (BYTE) UNTIL statements should set the global flag &$RENT to 1. This flag will assure that the code generated by the UNTIL macro is reentrant. This reentrant code is slower than the none reentrant code and should be used only in reentrant programs. The global flag has to be defined and set before a CSECT statement. See the examples of the B TYPE UNTIL statement.

   Example:

```
         GBLB  &$RENT
&$RENT   SETB  1
XXXXXX   CSECT
```

NAME - WHILE

DESCRIPTION

The function of the WHILE macro is to generate the labels and instructions that branch to these labels to accomplish the programming function of iteration. The WHILE macro supports both instructions for incrementing/decrementing indexes and instructions for terminating the loop based upon a change in a logical condition. The WHILE statements support loops in which the indexing/condition-testing instructions are executed before the first pass through the code-body.

The WHILE MACRO specifications.

There are three different WHILE statements, the WHILE-DO, WHILE-OR-DO, and the WHILE-AND-DO. For the flowcharts of the WHILE statements, see the ENDDO macro writeup.

The general format for the WHILE-DO is:

1. Indexed WHILE-DO:

```
WHILE (index-instruction), DO

     code-body

ENDDO
```

which reads, "WHILE the index-instruction branches, continue to execute the code-body."

2. Logical WHILE-DO:

```
WHILE (condition), DO

     code-body

ENDDO
```

which reads, "WHILE the indicated condition is true, continue to execute the code-body."

The general format for the WHILE-OR-DO is:

```
WHILE (index-instruction), OR
WHILE (index-instruction), DO

     code-body

ENDDO

WHILE (condition), OR
WHILE (condition), DO
```

```
     code-body

ENDDO

WHILE (index-instruction), OR
WHILE (condition), DO

     code-body

ENDDO
```

The general format for the WHILE-AND-DO is:

```
WHILE (index-instruction), AND
WHILE (index-instruction, DO

     code-body

ENDDO
```

```
WHILE  (condition),AND
WHILE  (condition),DO

   code-body

ENDDO

WHILE  (index-instruction),AND
WHILE  (condition),DO

   code-body

ENDDO
```

The following shows the format of WHILE:

$$\text{WHILE}\ \left(\left\{\begin{matrix}\text{BCT}\\ \text{BXH}\\ \text{BXLE}\\ *\\ \text{BIT}\\ \text{B}\\ \text{H}\\ \text{F}\\ \text{E}\\ \text{D}\\ \text{C}\\ \text{T}\end{matrix}\right\},\begin{matrix}\text{(R1)}\\ \text{LABEL1}\end{matrix},\left\{\begin{matrix}\text{(R3)}\\ \text{IS}\\ \text{GT}\\ \text{LT}\\ \text{GE}\\ \text{LE}\\ \text{EQ}\\ \text{NE}\end{matrix}\right\},\left\{\begin{matrix}\text{(R2)}\\ \text{LABEL2}\\ \text{ONE(S)}\\ \text{OVERFLOW}\\ \text{PLUS}\\ \text{MINUS}\\ \text{MIXED}\\ \text{ZERO(S)}\\ \text{NMINUS}\\ \text{NPLUS}\\ \text{NONE(S)}\\ \text{NZERO(S)}\\ \text{OFF}\\ \text{ON}\end{matrix}\right\}\right),\left\{\begin{matrix}\text{OR}\\ \text{AND}\\ \text{DO}\end{matrix}\right\}\ \left[\text{,REG=}\right]$$

$$\text{WHILE}\ \left(\left[\text{TYPE}\right]\text{, LABEL, OPERATION, CONDITION}\right),\left\{\begin{matrix}\text{OR}\\ \text{AND}\\ \text{DO}\end{matrix}\right\}\ \left[\text{,REG=}\right]$$

<u>INDEXED WHILE</u>

The different types are:

1. BCT

$$\text{WHILE}\ \text{(BCT,R1)},\left\{\begin{matrix}\text{OR}\\ \text{AND}\\ \text{DO}\end{matrix}\right\}$$

Example:

```
                WHILE     (BCT,$1),DO
+                B        LABEL1
+LABEL2          EQU      *

                 CODE-BODY
                 ENDDO
+LABEL1          EQU      *
+                BCT      $1,LABEL2
```

2. BXH and BXLE

```
                                  ⎧ OR  ⎫
   WHILE  (  BXH    ,R1,  R3),   ⎨ AND ⎬
             BXLE                 ⎩ DO  ⎭
```

Examples:

```
                WHILE     (BXH,$1,$3),DO
+                B        LABEL1
+LABEL2          EQU      *

                 CODE-BODY

                 ENDDO
+LABEL1          EQU      *
+                BXH      $1,$3,LABEL2

                WHILE     (BXLE,$1,$3),DO
+                B        LABEL1
+LABEL2          EQU      *

                 CODE-BODY

                 ENDDO
+LABEL1          EQU      *
+                BXLE     $1,$3,LABEL2
```

LOGICAL WHILE

The different types are:

1. An * in the type field stands for the condition is already set. When using the * type, the Operation and Condition fields cannot be omitted.

   Example:

```
          WHILE   (*,,IS,PLUS),DO
+         B       LABEL1
+LABEL2   EQU     *

          CODE-BODY

          ENDDO
+LABEL1   EQU     *
+         BC      2,LABEL2
```

2. Bit type: will generate a test under mask. The only valid operation parameter is (IS).

$$\text{WHILE}\quad (\text{BIT},\text{LABEL},\text{IS},\left\{\begin{array}{l}\text{ZERO}\\ \text{ONE}\\ \text{ON}\\ \text{OFF}\\ \text{MIXED}\\ \text{NONE}\\ \text{NMIXED}\\ \text{NZERO}\end{array}\right\}),\left\{\begin{array}{c}\text{OR}\\ \text{AND}\\ \text{DO}\end{array}\right\}$$

   Example:

```
          WHILE   (BIT,A,IS,ZERO),DO
+         B       LABEL1
+LABEL2   EQU     *

          CODE-BODY
```

```
              ENDDO
+LABEL1       EQU        *
+             TM         A,L'A
+             BC         8,LABEL2
```

3. B Type

$$\text{WHILE}\quad \text{(B,LABEL1,IS,}\left\{\begin{array}{l}\text{ZERO(S)}\\ \text{PLUS}\\ \text{MINUS}\\ \text{NPLUS}\\ \text{NMINUS}\\ \text{ONE(S)}\\ \text{NZERO(S)}\\ \text{NONE(S)}\end{array}\right\}\text{),}\left\{\begin{array}{c}\text{OR}\\ \text{AND}\\ \text{DO}\end{array}\right\}$$

$$\text{WHILE}\quad \text{(B,LABEL1,}\left\{\begin{array}{c}\text{GT}\\ \text{LT}\\ \text{EQ}\\ \text{NE}\\ \text{GE}\\ \text{LE}\end{array}\right\}\text{,}\left\{\begin{array}{l}\text{T'}\\ \text{L'}\\ \text{X'4F'}\\ \text{C'FF'}\\ \text{B'01'}\\ \text{LABEL2}\\ \text{(R2)}\end{array}\right\}\text{),}\left\{\begin{array}{c}\text{AND}\\ \text{DO}\end{array}\right\}\ \left[\text{,REG=}\right]$$

[REG=] DEFAULTS TO $0

Examples:

```
              WHILE      (B,A,IS,ZERO),DO
+             B          LABEL1
+LABEL2       EQU        *

              CODE-BODY

              ENDDO
+LABEL1       EQU        *
+             CLI        A,X'00'
+             BC         8,LABEL2

              WHILE      (B,A,EQ,AAAAAAAA+16),DO
+             B          LABEL1
+LABEL2       EQU        *
```

```
                CODE-BODY

                ENDDO
+LABEL1         EQU         *
+               IC          $0,AAAAAAAA+16
+               STC         $0,*+5
+               CLI         A,X'00'
+               BC          8,LABEL2

                WHILE       (B,A,EQ,($1)),DO
+               B           LABEL1
+LABEL2         EQU         *

                CODE-BODY

                ENDDO
+LABEL1         EQU         *
+               STC         $1,*+5
+               CLI         A,X'00'
+               BC          8,LABEL2

                WHILE       (B,A,GT,138),DO
+               B           LABEL1
+LABEL2         EQU         *

                CODE-BODY
```

These B forms of the WHILE statement alters executable code and are not usable if the program is to be re-entrant

```
                WHILE  (B,ABLE,EQ,BAKER),DO
+    B          LABEL1
+LABEL2         DS    0H

      CODE-BODY

+LABEL1         DS    0H
+      CLC      0+ABLE,BAKER
+      BC       8,LABEL2

                WHILE  (B,ABLE,EQ,($1)),DO              Reentrant B TYPE
+      B        LABEL1
+LABEL2         DS    0H

      CODE-BODY

+LABEL1         DS    0H
+      EX       $1,*+8
+      B        *+8
+      CLI      ABLE,0
+      BC       8,LABEL2

                ENDDO
+LABEL1         EQU         *
+               CLI         A,138
+               BC          2,LABEL2

                WHILE       (B,A,GT,0+MUD),DO
+               B           LABEL1
+LABEL2         EQU         *

                CODE-BODY

                ENDDO
+LABEL1         EQU         *
+               CLI         A,0+MUD
+               BC          2,LABEL2
```

```
          WHILE    (B,A,GT,X'4F'),DO
+         B        LABEL1
+LABEL2   EQU      *

          CODE-BODY

          ENDDO
+LABEL1   EQU      *
+         CLI      A,X'4F'
+         BC       2,LABEL2
```

4. Fixed Point (H or F)

```
                                  ⎧ ONE(S)   ⎫
                                  ⎪ PLUS     ⎪
                                  ⎪ MINUS    ⎪   ⎧ OR  ⎫
                   (R1)           ⎪ ZERO(S)  ⎪   ⎪ AND ⎪
WHILE  (⎧H⎫, LABEL1, IS,⎨ NZERO(S) ⎬),⎨ DO  ⎬  [,REG=]
        ⎩F⎭                       ⎪ NMINUS   ⎪   ⎩     ⎭
                                  ⎪ NPLUS    ⎪
                                  ⎩ NONE(S)  ⎭

                          ⎧ GT ⎫ ⎧ LABEL2 ⎫
                          ⎪ LT ⎪ ⎪ (R2)   ⎪     ⎧ OR  ⎫
                   (R1)   ⎪ GE ⎪ ⎪ =F'  ' ⎪     ⎪ AND ⎪
WHILE  (⎧H⎫, LABEL1,⎨ EQ ⎬,⎨ =H'  ' ⎬),⎨ DO  ⎬  [,REG=]
        ⎩F⎭               ⎪ NE ⎪ ⎪ =X'  ' ⎪     ⎩     ⎭
                          ⎩ LE ⎭ ⎩ =C'  ' ⎭
```

[REG=] **Defaults to $0.**

```
              WHILE     (H,A,IS,PLUS),DO
+             B         LABEL1
+LABEL2       EQU       *

              CODE-BODY

              ENDDO
+LABEL1       EQU       *
+             LH        $0,A
+             LTR       $0,$0
+             BC        13,LABEL2

              WHILE     (H,($1),IS,ZERO),DO
+             B         LABEL1
+LABEL2       EQU       *

              CODE-BODY

              ENDDO
+LABEL1       EQU       *
+             LTR       $1,$1
+             BC        8,LABEL2

              WHILE     (H,A,GT,B),DO
+             B         LABEL1
+LABEL2       EQU       *

              CODE-BODY

              ENDDO
+LABEL1       EQU       *
+             LH        $0,A
+             CH        $0,B
+             BC        2,LABEL2

              WHILE     (H,A,EQ,($1)),DO
+             B         LABEL1
+LABEL2       EQU       *
```

```
                CODE-BODY

                ENDDO
+LABEL1         EQU        *
+               CH         $1,A
+               BC         7,LABEL2

                WHILE      (H,($1),EQ,($2)),DO
+               B          LABEL1
+LABEL2         EQU        *

                CODE-BODY

                ENDDO
+LABEL1         EQU        *
+               CR         $1,$2
+               BC         8,LABEL2

                WHILE      (F,A,IS,PLUS),DO
+               B          LABEL1
+LABEL2         EQU        *

                CODE-BODY

                ENDDO
+LABEL1         EQU        *
+               L          $0,A
+               LTR        $0,$0
+               BC         2,LABEL2

                WHILE      (F,($1),IS,ZERO),DO
+               B          LABEL1
+LABEL2         EQU        *

                CODE-BODY

                ENDDO
+LABEL1         EQU        *
+               LTR        $1,$1
+               BC         8,LABEL2
```

```
              WHILE      (F,A,GT,B),DO
+             B          LABEL1
+LABEL2       EQU        *

              CODE-BODY

              ENDDO
+LABEL1       EQU        *
+             L          $0,A
+             C          $0,B
+             BC         2,LABEL2

              WHILE      (F,($1),GT,B),DO
+             B          LABEL1
+LABEL2       EQU        *

              CODE-BODY

              ENDDO
+LABEL1       EQU        *
+             C          $1,B
+             BC         2,LABEL2

              WHILE      (F,A,EQ,($1)),DO
+             B          LABEL1
+LABEL2       EQU        *

              CODE-BODY

              ENDDO
+LABEL1       EQU        *
+             C          $1,A
+             BC         7,LABEL2

              WHILE      (F,($1),EQ,($2)),DO
+             B          LABEL1
+LABEL2       EQU        *

              CODE-BODY
```

```
         ENDDO
+LABEL2  EQU     *
+        CR      $1,$2
+        BC      7,LABEL2
```

5. **Floating Point (E or D)**

$$\textbf{WHILE}\ \ (\begin{Bmatrix}\text{E}\\ \text{D}\end{Bmatrix}\overset{\text{(R1)}}{\text{,LABEL1}}\text{,IS,}\begin{Bmatrix}\text{ONE(S)}\\ \text{PLUS}\\ \text{MINUS}\\ \text{ZERO(S)}\\ \text{NZERO(S)}\\ \text{NMINUS}\\ \text{NPLUS}\\ \text{NONE(S)}\end{Bmatrix}\text{),}\begin{Bmatrix}\text{OR}\\ \text{AND}\\ \text{DO}\end{Bmatrix}\ [\text{,REG=}]$$

$$\textbf{WHILE}\ \ (\begin{matrix}\text{E}\\ \text{D}\end{matrix}\overset{\text{(R1)}}{\text{,LABEL1}}\text{,}\begin{Bmatrix}\text{GT}\\ \text{LT}\\ \text{GE}\\ \text{EQ}\\ \text{NE}\\ \text{LE}\end{Bmatrix}\text{,}\begin{Bmatrix}\text{LABEL2}\\ \text{(R2)}\\ \text{=E'}\quad\text{'}\\ \text{=D'}\quad\text{'}\end{Bmatrix}\text{),}\begin{Bmatrix}\text{OR}\\ \text{AND}\\ \text{DO}\end{Bmatrix}\ [\text{,REG=}]$$

[REG=] Defaults to FPRO.

```
         WHILE   (E,A,IS,PLUS),DO
+        B       LABEL1
+LABEL2  EQU     *

         CODE-BODY

         ENDDO
+LABEL1  EQU     *
+        LE      FPRO,A
+        LTER    FPRO,FPRO
+        BC      2,LABEL2
```

```
          WHILE     (D,(FPRO),IS,ZERO),DO
+         B         LABEL1
+LABEL2   EQU       *

          CODE-BODY

          ENDDO
+LABEL1   EQU       *
+         LTDR      FPRO,FPRO
+         BC        8,LABEL2

          WHILE     (E,A,GT,B),DO
+         B         LABEL1
+LABEL2   EQU       *

          CODE-BODY

          ENDDO
+LABEL1   EQU       *
+         LE        FPRO,A
+         CE        FPRO,B
+         BC        2,LABEL2

          WHILE     (D,(FPRO),GT,B),DO
+         B         LABEL1
+LABEL2   EQU       *

          CODE-BODY

          ENDDO
+LABEL1   EQU       *
+         CD        FPRO,B
+         BC        2,LABEL2

          WHILE     (E,A,EQ,(FPR2)),DO
+         B         LABEL1
+LABEL2   EQU       *

          CODE-BODY
```

```
          ENDDO
+LABEL1   EQU       *
+         CE        FPR2,A
+         BC        7,LABEL2

          WHILE     (D,(FPR0),EQ,(FPR2)),DO
+         B         LABEL1
+LABEL2   EQU       *

          CODE-BODY

          ENDDO
+LABEL1   EQU       *
+         CDR       FPR0,FPR2
+         BC        8,LABEL2
```

6. Character (C)

$$\text{WHILE}\quad (\text{C, LABEL1}, \left\{\begin{matrix}\text{LT}\\ \text{GT}\\ \text{GE}\\ \text{EQ}\\ \text{LE}\\ \text{NE}\end{matrix}\right\}, \text{LABEL2}), \left\{\begin{matrix}\text{OR}\\ \text{AND}\\ \text{DO}\end{matrix}\right\}$$

```
          WHILE     (C,ABLE,EQ,BETA),DO
+         B         LABEL1
+LABEL2   EQU       *

          CODE-BODY

          ENDDO
+LABEL1   EQU       *
+         CLC       ABLE,BETA
+         BC        8,LABEL2

          WHILE     (C,=C'SED',EQ,0($3)),DO
+         B         LABEL1
+LABEL2   EQU       *

          CODE-BODY
```

```
              ENDDO
+LABEL1       EQU         *
              CLC         =C'SED',O($3)
+             BC          8,LABEL2
```

7. Test Under Mask (T)

```
                                     ( ZERO   )
                                     | ONE    |
                                     | ON     |
                                     | OFF    |          ( OR  )
   WHILE  (T,LABEL1, { MASK },       { MIXED  } ),       { AND }
                                     | NONE   |          ( DO  )
                                     | NMIXED |
                                     ( NZERO  )
```

```
              WHILE       (T,A,X'11',ON),DO
+             B           LABEL1
+LABEL2       EQU         *

              CODE-BODY

              ENDDO
+LABEL1       EQU         *
+             TM          A,X'11'
+             BC          8,LABEL2
```

8. Type Field Omitted

```
              WHILE       (,A,IS,ZERO),DO
+             B           LABEL1
+LABEL2       EQU         *

              CODE-BODY

              ENDDO
+LABEL1       EQU         *
+             LE          FPRO,A
+             LTER        FPRO,FPRO
+             BC          8,LABEL2
              {
              {
A             DC          E'O'
```

PROGRAMMING NOTES

Also see programming notes for IF macro.

1. "Index-Instruction" can be any one of the following:

   a. BCT, r1
   b. BXH, r1, r3
   c. BXLE, r1, r3

2. "Code-Body" can be any group of valid machine and/or macro-instructions, including a maximum of twenty nested WHILE/UNTIL's. Multiple index-instructions in the same loop are also supported.

3. The WHILE function causes the indexing instructions to be assembled at the end of the loop but generates a branch past the code-body to cause the indexes to be incremented/decremented before the first pass through the code-body.

4. The level of a nested WHILE/UNTIL statement can be found in the LABELS that are generated.

```
             WHILE    (condition),DO
+            B        W1/5/xxxx
+W2/5/xxxx
```

   The /5/ stands for the level of this nested WHILE statement.

5. Any time a register notation is used in a logical-WHILE statement, the register must be in parentheses. It does not make any difference whether a register is in parentheses or not with an Indexed-WHILE statement.

6. Misspelling and abbreviation of "conditions" mnemonices is not allowed.

7. Restriction: Expressions cannot be over sixteen characters in length.

8. Reentrant programs that use the B TYPE (BYTE) WHILE statements should set the global flag &$RENT to 1. This flag will assure that the code generated by the WHILE macro is reentrant. This reentrant code is slower than the none reentrant code and should be used only in reentrant programs. The global flag has to be defined and set before a CSECT statement. See the examples of the B TYPE WHILE statement.

   Example:

```
         GBLB  &$RENT
&$RENT    SETB  1
XXXXXX    CSECT
```

NAME - XIBIT-OIBIT

PURPOSE

The purpose of the XIBIT and OIBIT macros are to generate an EXCLUSIVE OR IMMEDIATE instruction to invert a specified bit, and an INCLUSIVE OR IMMEDIATE instruction to "turn on" a specified bit, respectively. Both utilize the length code of the symbol to be operated upon.

DEFINITION

[symbol]	XIBIT	Symbol

[symbol]	OIBIT	Symbol

where Symbol is the label of a data base definition having an associated length code.

EXPANSION

NAME	OPERATION	OPERAND
[symbol]	XIBIT OIBIT	LABEL
+[symbol]	XI OI	LABEL, L'LABEL

GENERAL NOTES

- The XIBIT and OIBIT macros will be utilized most often in conjunction with the BIT macro, since BIT generates a desired length code associated with a valid label.

4. USE WITH RTPM

4.1 PRE- AND POST-ASSEMBLY PROCESSORS

The concept of structured programming involves a physically structured program listing as an integral part. In order to automate this (permit source coding to be aligned as per OS standards: columns 1, 10, 16), two processors were written to generate either a structured source listing or a structured assembly listing. The post-assembly processor also optionally deletes unreferenced labels from DSECTs and assembly cross-references.

The pre- and post-assembly processors are invoked by coding the following PARM keyword parameter on the EXEC card which invokes RTPM:

```
PARM.STEPNAME=',,,,SMTPGASM,SMXRPASM'
```

and adding the following DD card:

```
//GSSCPRNT  DD  UNIT=DISK,SPACE=(TRK,(X,Y))
```

where X is typically 50 - the largest assembly listing that is expected. This is not the total amount of assembler output that the jobstep will generate.

The program SMTPGASM serves as the linkage between RTPM and the pre-assembly processor. When SMTPGASM receives control from RTPM (this occurs when a GASM control card is used in place of a ASSM control card) register 1 points to the same parameter list that will be passed to the assembler following the LINK to SMTPGASM. Upon receiving control SMTPGASM issues an OPEN (a QSAM get-locate type) on the input source member. It then reads the source member for a card that begins with a "*)" pattern. Upon finding this card it scans this card looking for a valid (whose name is in a table) control ID. If a valid ID is found it LINKs to the associated module, defined in a JOBLIBXX DD card added to the RTPM step.* Otherwise it will continue reading the source cards for a valid ID. If a valid ID is never found SMTPGASM will return to RTPM without any special error condition set.

* This capability is not part of HLAL.

When used to generate a structured source listing, no *) cards need to be included in the source member, but an additional DD card must be added to the step for the pre-assembly processor to place the structured source listing:

```
//STRUCTUR  DD  UNIT=DISK,SPACE=(TRK,(X,Y))
```

where X is typically 5 - the largest single source member that is being used.

If no STRUCTUR DD card is included in the jobstep, no structured source listing will show up on ASMPRINT. Note that when the STRUCTUR DD card is used, ./ GASM must be used in place of ./ ASSM.

The post-assembly processor is used to indent assembled code and eliminate non-referenced labels from assembly listings and cross reference only if SMXRPASM is specified in the PARM field of the EXEC card. REHDRTPM will link to SMXRPASM just after linking to the assembler and prior to a BALR to the collection tape writer. SMXRPASM will be passed the DDNAME of the assembler - written print data set and the DDNAME that the assembler would have used if RTPM was not in post-assembly user exit mode. After execution of SMXRPASM, assembly print will be located on the data set that would have been used if no post-assembly exit has occurred; thus normal RTPM processing can resume after the exit.

SMXRPASM will always produce a structured listing unless a $$$$$$$# is found in the cross-reference. The structured listing will be indented three spaces for each new logical section of code and restored three spaces for each logical section of code that is terminated. In addition a level number will be output to indicate the level of indention on all statements. Wrap-around will occur after the level of indention exceeds three levels.

To initiate the non-referenced labels deletion function of SMXRPASM a $ EQU * card must be included in the source listing at the location the deletion is desired to start. A $$ EQU * would stop the deletion function, a $$$ EQU * would start it again, and a $$$$ EQU * would stop it. At this point it could not be started again. SMXRPASM reads through the assembly listing until finding the cross-reference. If a $ EQU * card has been included in the source listing it will be the first label in the cross-reference. If this $ card is found SMXRPASM builds a table of all cross-reference labels that are referenced or whose definition statement number is outside the $ cards limit. It also builds a table of the definition statement numbers of all cross-reference labels that are not referenced and whose definition number is in the $ cards limit. Once these tables are built SMXRPASM goes back to the beginning of the assembly listing and deletes all statements whose numbers are in the table built above. In addition if the statement following the deleted statement has a blank or an asterisk in card column one it will also be deleted. In the cross-reference it deletes all labels whose name does not appear in the other table built above.

4.2 INVOKING THE HLAL MACROS

To invoke the HLAL common MACROs, the following SYSLIB concatenation is suggested:

```
//SYSLIB  DD   DSN=&TMPSRC,DISP=(SHR,PASS),VOL=REF=*.SYSTEMPS
//        DD   DSN=SYS1.MACHAL,DISP=(SHR,PASS),UNIT=DISK,          X
//                   VOL=SER=PRODSK
//        DD   DSN=(User MACRO Library)
//        DD   DSN=SYS1.MACLIB,DISP=SHR
```

The HLAL MACROs reside on PRODSK (SYS1.MACHAL) and the pre- and post-assembly processors are in SYS1.RTPMLIB.

IBM HLAL Coding Reference Data

(REF. SKYLAB USERS' GUIDE)

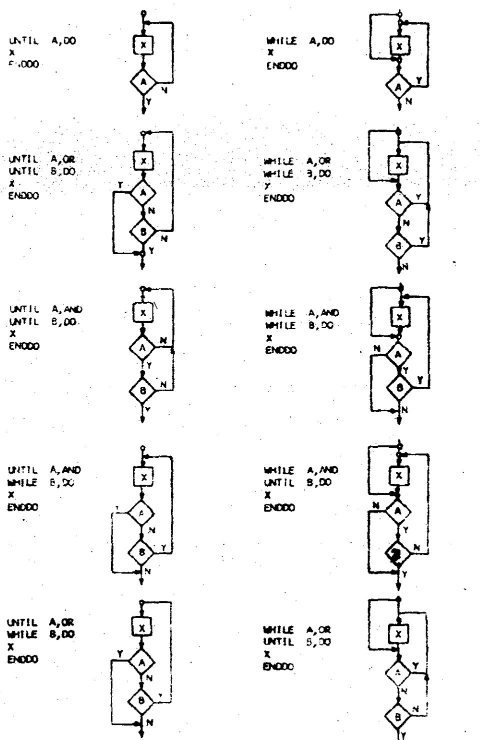
STRUCTURED CODE MACROS

```
IF [{type}],{label 1},{operation},{condition},{THEN} [,REG-(R2)]
   [{ *  }] {(R1)   } {mask     } {label 2  } {OR  }
                                  {(R2)     } {AND }
ELSE
ENDIF

       ( (BCT,(R1))                                              )
       ( ({BXH },(R1),(R3)[,(R2)])                               )  {DO }
UNTIL  (  {BXLE}                                                 ), {OR }
WHILE  ( ([{type}],{label 1},{operation},{condition})            )  {AND}
       (  [{ *  }] {(R1)   } {mask     } {label 2  }             )
       (                                 {(R2)     }             )  [,REG=(R4)]
ENDDO

STRTSRCH {UNTIL}, {UNTIL/WHILE operand}
         {WHILE}
EXITIF {IF operand}

ORELSE
ENDLOOP
ENDSRCH

DO segment {,reg}
           {,$14}

BGNSEG segment[,reg]
ENDSEG segment

                         {AT=address list}  [,INDX=number]
                         {BT=address list}
[symbol] CASE  case reg, {    (R)        }
                         {LAT=address    }  [,RETREG=register]
                         {    (R)        }
                         {LBT=address    }
```

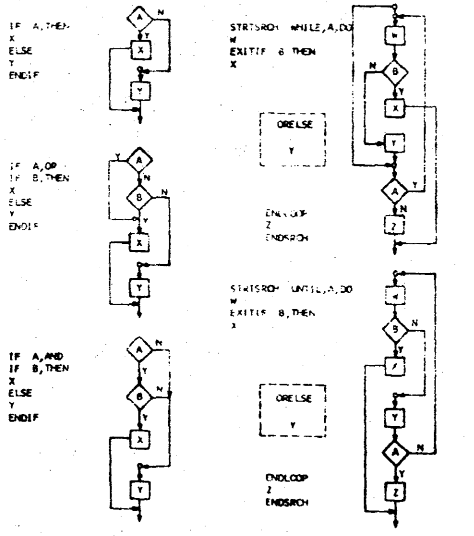
type	operation/mask	condition/label 2/(R2)
*	IS	ONE,OVERFLOW,PLUS,MINUS,MIXED,ZERO, NMINUS,NPLUS NONE,NZERO,NMIXED
*	GT,LT,GE,LE,EQ,NE	label 2(ignored)
BIT	IS	ONE,MIXED,ZERO,NONE,NZERO,NMIXED, ON,OFF
Z	IS	PLUS,MINUS,[illegible] NMINUS,NPLUS, NZER
B	GT,LT,GE,[illegible],NE	label [illegible] R2),[illegible] F',C'F',B'01'
{H} {F}	IS	ONE,PLUS,MINUS,ZERO,NMINUS,NPLUS, NONE,NZER
{H} {F}	GT,LT,GE,LE,EQ,NE	label 2,[illegible] literal
{E} {D}	IS	ONE,PLUS,MINUS,ZE[illegible],NMINUS,NPLUS, NONE,NZERO
{E} {D}	GT,LT,GE,LE,EQ,NE	label 2,(R2),literal
[illegible]	GT,LT,GE,LE,EQ,NE	label 2
T	mask	ONE,MIXED,ZERO,[illegible] NZERO,NMIXED, ON,OFF

DATA BASE DEFINITION

```
                {bit number                  }
[symbol] BIT    {(list of bit numbers)       } [,ON]
                {binary 8-bit configuration  }

[symbol] BYTE [one byte hex value]
```



NOTES IN AN UNTIL A BCT = YES WHEN THE REGISTER = 0 AFTER EXECUTION OF BCT
IN A WHILE A BCT = NO WHEN THE REGISTER = 0 AFTER EXECUTION OF BCT



MISCELLANEOUS MACROS

```
csectname  HEADC  [INTP=YES][,RET=YES]

[INTP]     ENTER  (csectname,entry2[,entry3,entry4,...])
                  [,(label1,label2,label3,label4,...)]
```

FORMAT OF MATH MACRO

```
[symbol]  MATH  'math expression'[,REG=register list][,TRACE=ON/OFF]
                      (reg)
                [ANS=symbol] [,TYP=E,D,H, or null]
```

MATH REFERENCE SHEET

OP-CODE	OPERATION	OP-CODE	OPERATION
+, PLUS	ADD	C, WITH	COMPARE
-, MINUS	SUBTRACT	TO, COMPARE	COMPARE
/, OVER	DIVIDE		
*, TIMES	MULTIPLY	.,LOAD	LOAD
		RELOAD	LOAD
=,SAVED-IN	STORE		-,*
STORE, STORE-IN	STORE	XOR	Exclusive OR
OR	OR	AND	AND

OP-CODE	OPERATION
$,HERE=	Place label on next instruction
EQU. LABEL=	Place label on next instruction
B	Branch on Condition 15
BH, BP	Branch on Condition 2
BL, BM	Branch on Condition 4
BE, BZ	Branch on Condition 8
BO	Branch on Condition 1
BNH, BNP	Branch on Condition 13
BNL, BNM	Branch on Condition 11
BNE, BNZ	Branch on Condition 7

VALID OPERAND

Any valid SYMBOL	Any valid TERM
Any valid LITERAL	Any valid NUMBER
Any valid REG	Any valid EXP
Any valid PREFIXED-EXP	

OPERAND	STARTING CHARACTERS
NUMBERS	[illegible] 0 —— 9
SYMBOL	A —— Z, $, and @ (only 8 characters at maximum)
TERM	A —— Z, $, and @ (any assembly language operand)
REG	[illegible] followed immediately by a symbol or number
EXP	[illegible] followed by at least one blank
LITERAL	[illegible] sign (like in assembly language except quote doubled)

VALID PREFIXES FOR EXP'S

ABS	HALF
NEG	DUBL
TEST	SQAR
COMP	

SPECIAL TERM FOR SYSTEM PARAMETERS

SYSPARM (MXXXXXNN)
MXXXXX = SYSPARM NAME
NN = null or 0 —— 99

5. EXAMPLE

The following pages present an example of the use of HLAL. First, is the CSECT's structured source listing and second is the same CSECT's structured assembly listing. As discussed previously, both or either the assembled or source structuring may be obtained using the pre- and post-assembly processors with RTPM.

In looking at the source statements, note the effective use of comments with HLAL MACRO statements, producing a much more readable listing.

```
1  2  3           ELSE  ,      ...........THIS IS A NEW POSITION CONTROL WORD      06300000
1  2  3  4           L       $0,0($IN)           LOAD THE NPOW INTO GPRO                 06400000
1  2  3  4           SRL     $0,6                                                        06500000
1  2  3  4           SRDL    $0,9                                                        06600000
1  2  3  4           SRL     $1,23               GPR1 NOW CONTAINS THE NEW X POSITION    06700000
1  2  3  4           N       $0,=X'000001FF'      GPRO NOW CONTAINS THE NEW Y POSITIN    06800000
1  2  3  4           S       $0,=F'12'           Y-12                                    06900000
1  2  3  4           SRL     $0,3                (Y-12)/8                                07000000
1  2  3  4           MH      $0,=AL2(LMARG+NCOL+RMARG)                                   07100000
1  2  3  4           LR      $3,$0                                                       07200000
1  2  3  4           LA      $3,LMARG($3)                                                07300000
1  2  3  4           IF      F,($0),GT,($YMAX),THEN   THIS IS NEW YMAX                   07400000
1  2  3  4  5            LR      $YMAX,$0                                                    07500000
1  2  3  4           ENDIF                                                               07600000
1  2  3  4           BCTR    $1,0                (X-1)                                   07700000
1  2  3  4           SR      $0,$0                                                       07800000
1  2  3  4           D       $0,=F'7'            (X-1)/7                                 07900000
1  2  3  4           AR      $3,$1               ((Y-12)/8)*(LINE WIDTH)+LMARG+(X-1)/7 08000000
1  2  3  4           IF      F,($3),GT,=A((NLIN-1)*(LMARG+NCOL+RMARG)),THEN              08100000
1  2  3  4  5            LA      $15,16         ERROR = BAD X,Y COORDINATE                   08200000
1  2  3  4  5            B       RSCOWDHC   *-------------RETURN-------------------------*   08300000
1  2  3  4           ENDIF                                                               08400000
1  2  3  4*                                                                              08500000
1  2  3  4*   GPR3 NOW EQUALS THE STARTING DISPLACEMENT INTO THE BUFFER OF THE           08600000
1  2  3  4*                      NEXT CHARACTER                                          08700000
1  2  3  4*                                                                              08800000
1  2  3           ENDIF                                                              08900000
1  2  3           LA     $IN,4($IN)       .....POINT $IN TO NEXT WORD.....           09000000
1  2          ENDDO                                                                  09100000
1           ELSE                                                                09200000
1  2*                                                                               09300000
1  2*                                                                               09400000
1  2*  PUT VECTOR LOGIC HERE                                                        09500000
1  2*                                                                               09600000
1  2*                                                                               09700000
1           ENDIF                                                               09800000
1           WHILE  (F,0($IN),NE,C+COMMAND),DO         ****                      09900000
1  2            LA     $IN,4($IN)                      *** TEMP                    10000000
1           ENDDO                                     ****                      10100000
          ENDDO                                                           10200000
          SR      $15,$15              * NO ERRORS IN FORMATTING DATA     10300000
          B       RSCOWDHC *-*-*-*-*-* RETURN *-*-*-*-*-*-*-*-*-*-*-*-*   10400000
          SPACE 5                                                         10500000
MCVGEBTR  DC      X'00' ********* MCVG TO EBCDIC TR TABLE **************** 10600000
          DC      C'1234567890* '                                         10700000
          DC      X'0D0F'                                                 10800000
          DC      C'O'                 * OMEGA                            10900000
          DC      X'10'                                                   11000000
          DC      C'/STUVWXYZG/TPDO-JKLMNOPQRS='                          11100000
          DC      X'2D2E2F'                                               11200000
          DC      C'+ABCDEFGHI(.L'                                        11300000
          DC      C'3D'                                                   11400000
          DC      C'**'                                                   11500000
************************************************************************ 11600000
*   NOTES                                                                11700000
*                                                                        11800000
*   OMEGA (LOWER & CAPS ) =  O                                           11900000
*   GAMA                   =  G                                          12000000
*   COLON                  =  /                                          12100000
*   THETA (CAPS)           =  T                                          12200000
*   PHI   (LOWER)          =  P                                          12300000
*   DELTA (CAPS)           =  D                                          12400000
*   SIGMA (LOWER)          =  S                                          12500000
*   POINTER                =  -                                          12600000
*   ANGLE                  =  (                                          12700000
*   LAMBDA (LOWER)         = L                                           12800000
*   PSI   (LOWER)          =  X                                          12900000
*   DEGREE                 =  *                                          13000000
************************************************************************ 13100000
          SPACE 3                                                        13200000
CHARWIDT  DC      C'O'                                                   13300000
          SPACE 3                                                        13400000
          LTORG                                                          13500000
          END                                                            13600000
```

```
******************************************************************* 00100000
*                                                                 * 00200000
*  FUNCTION -- TO FORMAT THE INPUT MCVG MESSAGE DATA INTO THE CORRECT * 00300000
*              EBCDIC FORM, IN THE OUTPUT BUFFER AREA             * 00400000
*                                                                 * 00500000
*  INPUTS ---- GPR1 POINTS TO DOUBLE WORD = ADDR INPUT DATA        * 00600000
*                                          ADDR OUTPUT BUFFER      * 00700000
*                                                                 * 00800000
*-----------------------------------------------------------------* 00900000
*                                                                   01000000
STOWDHCE HEADC RET=YES                                              01100000
RMARG    EQU   10                    = NO. COLUMNS TO RIGHT OF DISPLAY 01200000
LMARG    EQU   30                    = NO. COLUMNS TO LEFT OF DISPLAY  01300000
NCOL     EQU   72                    = NO. COLUMNS/LINE IN DISPLAY     01400000
NLIN     EQU   62                    = NO. LINES IN OUTPUT DISPLAY     01500000
IMDCW    EQU   X'80'                 ... INSERT MEMOR DEVIDE           01600000
EOMCW    EQU   X'40'                 ... END OF MESSAGE                01700000
EMSCW    EQU   X'08'                 ... ERASE MEMORY SUBSECTION       01800000
SCCW     EQU   X'04'                 ... START SMALL CHARACTERS        01900000
LCCW     EQU   X'02'                 ... START LARGE CHARACTERS        02000000
VCW      EQU   X'01'                 ... START VECTORS                 02100000
COMMAND  EQU   X'3D'                 COMMAND WORD                      02200000
NOINC    EQU   X'10'                 NO MEMORY ADDR. INCREMENT         02300000
STRTBLNK EQU   X'0D'                 START BLINK                       02400000
STOPBLNK EQU   X'0E'                 STOP  BLINK                       02500000
$IN      EQU   $10                                                     02600000
$YMAX    EQU   $2                    ... DISP. INTO BUFFER OF LAST LIN 02700000
         SPACE 3                                                       02800000
         L     $IN,0($1)             ... A( INPUT DATA )               02900000
         L     $11,4($1)             ... A( OUTPUT BUFFER )            03000000
         SR    $YMAX,$YMAX                                             03100000
         SPACE 3                                                       03200000
         WHILE (B,0($IN),NE,0+COMMAND),DO    FIND  FIRST CMD WORD      03300000
  1        LA    $IN,4($IN)                                              03400000
         ENDDO                                                         03500000
         SPACE 3                                                       03600000
         WHILE (T,3($IN),X'0'+EMSCW+ECMCW,ZERO),DO  LOOP THROUGH DATA  03700000
  1        IF    B,3($IN),NE,0+VCW,THEN THIS IS NOT A START VECTOR MODE  03800000
  1  2        IF    B,3($IN),EQ,0+SCCW,THEN SIZE = SMALL NOW               03900000
  1  2  3       MVI   CHARWIDT+3,1                                          04000000
  1  2        ELSE                                                          04100000
  1  2  3       IF    B,3($IN),EQ,0+LCCW,THEN SIZE = LARGE NOW              04200000
  1  2  3  4       MVI   CHARWIDT+3,2         SIZE NOW = LARGE                  04300000
  1  2  3       ENDIF                                                       04400000
  1  2        ENDIF                                                       04500000
  1  2        LA    $IN,4($IN)           BUMP $IN PAST CONTROL WORD       04600000
  1  2        WHILE (B,0($IN),NE,C+COMMAND),DO                            04700000
  1  2  3       IF    T,3($IN),X'10',ZERO,OR                                04800000
  1  2  3       IF    T,3($IN),X'2F',NZERO,THEN THIS IS A DATA WORD         04900000
  1  2  3  4       L     $4,0($IN)                                             05000000
  1  2  3  4       LA    $6,5                     = A MAX OF 5 CHAR./WORD      05100000
  1  2  3  4       UNTIL (BCT,$6),DO                                           05200000
  1  2  3  4  5       SRDL  $4,6                                                  05300000
  1  2  3  4  5       SRL   $5,26                                                 05400000
  1  2  3  4  5       IF    H,($5),EQ,=X'0010',THEN  STOP PROCESSING THIS WORD    05500000
  1  2  3  4  5  6       LA    $6,1                                                  05600000
  1  2  3  4  5       ELSE                                                        05700000
  1  2  3  4  5  6       IC    $5,MCVGEBTP($5)                                       05800000
  1  2  3  4  5  6       STC   $5,0($3,$11)                                          05900000
  1  2  3  4  5  6       A     $3,CHARWIDT                                           06000000
  1  2  3  4  5       ENDIF                                                       06100000
  1  2  3  4       ENDDO                                                       06200000
```

```
                                                                                                   PAGE    2

OC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                              ASM H V 02 05.09 02/21/72

                                   1 *********************************************************************** C0100000
                                   2 *                                                                     * 00200000
                                   3 *  FUNCTION -- TO FORMAT THE INPUT MCVG MESSAGE DATA INTO THE CORRECT * C0300000
                                   4 *                EBCDIC FORM, IN THE OUTPUT BUFFER AREA               * C0400000
                                   5 *                                                                     * 00500000
                                   6 *  INPUTS ---- GPR1 POINTS TO DOUBLE WORD = ADDR INPUT DATA           * 00600000
                                   7 *                                           ADDR OUTPUT BUFFER        * 00700000
                                   8 *                                                                     * 0080C0C0
                                   9 *-------------------------------------------------------------------* 00900000
                                  1C *                                                                       C1000000
                                  11 SCCWDHCF HEADC RET=YES                                                  C1100000
000000                            12+SCCWDHCF CSECT                                                          01-HEACC
000000 47F0 F00C            0000C 13+         B      12(0,15) .     ROUTINE ENTRY POINT REQUIRED IN REG15    C1-HEACC
000004 07E2C3D6E6C4C8C3           14+         DC     AL1(7),CL7'SCOWDHCF'                                   X01-HEACC
                                    +                                     PROVIDES ROUTINE NAME IN FORMATED DUMF
00000C 90EC D00C            0000C 15+         STM    14,12,12(13) .SAVES REGISTERS FOR CALLING ROUTINE       01-HEACC
000010 45E0 F06A            0006A 16+         BAL    14,106(0,15)                                            C1-HEACC
000014 0000200000000000           17+         DC     X'00002000',F'0',8CL8'SCOWDHCF'                        X01-HEACC
00001C F2C3D6E6C4C8C3C6             +                                              FILLS SAVE WITH CSECT NAME
00005C 5800 D004            00004 18+RSCOWDHC L      13,4(0,13) .   **PROVIDES A RESTORE OF REGISTERS AND     C1-HEACC
000060 58ED 000C            0000C 19+         L      14,12(13) .    RESTORES ALL REGISTERS                   01-HEACC
000064 980C D014            00014 20+         LM     0,12,20(13) .        EXCEPT REGISTER 15                 01-HEACC
000068 07FE                       21+         BR     14 .     RETURN TO CALLER WITH B R(CSECT NAME)          01-HEACC
00006A 50D0 F004            00004 22+         ST     13,4(0,14) .   STORES OLD SAVE AREA ADDRESS IN NEW AREA  C1-HEACC
00006E 50F0 D008            00008 23+         ST     14,8(0,13) .   STORES NEW SAVE AREA ADDRESS IN OLD       C1-HEACC
000072 18DE                       24+         LR     13,14 .        LOADS NEW SAVE AREA ADDRESS IN REG13      C1-HEACC
00000014                          25+         USING SCOWDHCF+20,13                                          XC1-HEACC
                                    +                            .   ESTABLISHES REG13 AS THE BASE REGISTER

                                  27+*                              GOES THRU REGISTER EQUATE ONLY ONCE
00000000                          28+$0       EQU    0              **                                        02-ECUAT
00000001                          29+$1       EQU    1              **                                        02-ECUAT
00000002                          30+$2       EQU    2              **                                        02-ECUAT
00000003                          31+$3       EQU    3              **                                        C2-ECUAT
00000004                          32+$4       EQU    4              **                                        02-ECUAT
00000005                          33+$5       EQU    5              **                                        C2-ECLAT
00000006                          34+$6       EQU    6              **IF THESE SUBSTITUTES ARE USED AS         C2-ECUAT
00000007                          35+$7       EQU    7              **REGISTER NUMBERS THE CROSS-REFERENCE     C2-ECUAT
00000008                          36+$8       EQU    8              **TABLE WILL PROVIDE A LIST OF WHERE       02-ECUAT
00000009                          37+$9       EQU    9              **EACH REGISTER WAS USED                   C2-ECUAT
0000000A                          38+$10      EQU    10             **                                        02-ECUAT
0000000B                          39+$11      EQU    11             **                                        C2-ECLAT
0000000C                          40+$12      EQU    12             **                                        C2-ECUAT
0000000D                          41+$13      EQU    13             **                                        C2-ECLAT
0000000E                          42+$14      EQU    14             **                                        02-ECUAT
0000000F                          43+$15      EQU    15             **                                        02-ECUAT
00000000                          44+FPR0     EQU    0                                                        02-ECUAT
00000002                          45+FPR2     EQU    2                                                        02-ECUAT
00000004                          46+FPR4     EQU    4                                                        02-ECUAT
00000006                          47+FPR6     EQU    6                                                        C2-ECUAT
```

```
    ---SCOWDHCF--------SCOWDHCF--------SCOWDHCF--------SCOWDHCF-------SCOWDHCF----                      PAGE    3

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                 ASM H V 02 05.C9 02/21/72

                                   51+*,********************************,*                       01-HEACC
                                   52+*,********************************,*                       01-HEACC
                                   53+*,**      CONTROL SECTION       **,*                       C1-HEACC
                                   54+*,**          SCOWDHCF                   **,*              01-HEACC
                                   55+*,********************************,*                       01-HEACC
                                   56+*,********************************,*                       C1-HEACC

0000000A                           58 RMARG     EQU   10                = NO. COLUMNS TO RIGHT OF DISPLAY    C120C0C0
0000001E                           59 LMARG     EQU   30                = NO. COLUMNS TO LEFT OF DISPLAY     C13CC0C0
00000048                           60 NCCL      EQU   72                = NO. COLUMNS/LINE IN DISPLAY        C140C0C0
0000003E                           61 NLIN      EQU   62                = NO. LINES IN OUTPUT DISPLAY        C150C0C0
00000080                           62 IMDCW     EQU   X'80'             ... INSERT MEMOR DEVIDE              C160C000
00000040                           63 EOMCW     EQU   X'40'             ... END OF MESSAGE                   C170C0CC
00000008                           64 EMSCW     EQU   X'08'             ... ERASE MEMORY SUBSECTION          0180C0C0
00000004                           65 SCCW      EQU   X'04'             ... START SMALL CHARACTERS           C190C000
00000002                           66 LCCW      EQU   X'02'             ... START LARGE CHARACTERS           C200C000
00000001                           67 VCW       EQU   X'01'             ... START VECTORS                    C2100000
00000030                           68 COMMAND   EQU   X'30'             COMMAND WORD                         C220C0C0
00000010                           69 NCINC     EQU   X'10'             NO MEMORY ADDR. INCREMENT            C230C000
0000000D                           7C STRTBLAK  EQU   X'0D'             START BLINK                          C240C000
0000000E                           71 STOPBLNK  EQU   X'0E'             STOP  BLINK                          C250C0C0
0000000A                           72 $IN       EQU   $10                                                    C260C000
00000002                           73 $YMAX     EQU   $2                ... DISP. INTO BUFFER OF LAST LIN    C270C0C0

000074 58A1 0000          00000    75             L     $IN,0($1)         ... A( INPUT DATA )                 C290C00C
000078 58B1 0004          00004    76             L     $11,4($1)         ... A( OUTPUT BUFFER )              C30C0000
00007C 1B22                        77             SR    $YMAX,$YMAX                                           C310C000

                                   79             WHILE (9,0($IN),NE,0+COMMAND),DO   FIND  FIRST CMD WORD     C330C0C0
00007E 47F0 C072          00C86  1   80+          B     W210003                                                01-WHILE
000082                           1   81+W110003   DS    0H                                                     01-WHILE
000082 41AA 0004          00004  1   82           LA    $IN,4($IN)                                             03400000
                                   83             ENDDO                                                        C350C0C0
000086                             84+W210003   DS    0H                                                       01-ENDCC
000086 9530 A000    00000          85+          CLI   0($IN),0+COMMAND                                         01-ENDCC
00008A 4770 C06E          00C82    86+          BC    7,W110003                                                01-ENDCC

                                   88             WHILE (T,3($IN),X'0'+EMSCW+EOMCW,ZERO),DO  LOOP THROUGH DATA  C370C0C0
00008E 47F0 C156          0016A  1   89+            B     W210005                                              01-WHILE
000092                           1   90+W110005   DS    0H                                                     01-WHILE
                                 1   91           IF    B,3($IN),NE,0+VCW,THEN THIS IS NOT A START VECTOR MODE 03800000
000092 9501 A003    00003          2   92+            CLI   3($IN),0+VCW                                       01-IF
000096 4780 C146          0015A    2   93+            BC    8,IF10006                                          01-IF
                                   2   94             IF    B,3($IN),EQ,0+SCCW,THEN SIZE = SMALL NOW         C390000C
00009A 9504 A003    00003          3   95+            CLI   3($IN),0+SCCW                                      01-II
00009E 4770 C096          000AA    3   96+            BC    7,IF20007                                          01-II
```

```
      ---SCOWDHCF=======SCCWDHC          3F=======SCOWDHCF=======SCOWDHCF        SCOWDHCF---                      PAGE    4

  LOC  OBJECT CODE    ADDR1 ADDR2           3 STMT    SOURCE STATEMENT                           ASM H  V 02  05.09 02/21/72

 0000A2 9201 D1AB      001BF                3    97          MVI    CHARWIDT+3,1                                    04000000
                                          2    98          ELSE                                                 04100000
 0000A6 47F0 D0A2            0C0B6          3    99+         B      IF20008                                         01-ELSE
 0000AA                                     3   100+IF20007        DS   0H                                          01-ELSE
                                            3   101          IF     B,3($IN),EQ,0+LCCW,THEN SIZE = LARGE NOW        04200000
 0000AA 9502 A003      00003                  4   102+         CLI    3($IN),0+LCCW                                 01-IF
 0000AE 4770 D0A2            00CB6            4   103+         BC     7,IF30009                                     01-IF
 0000B2 9202 D1AB      001BF        C00       4   104          MVI    CHARWIDT+3,2           SIZE NOW = LARGE       04300
                                            3   105          ENDIF                                                  04400000
 0000B6                                     3   106+IF30009         DS   0H                                         01-ENDIF
                                          2   107          ENDIF                                                04500000
 0000B6                                   2   108+IF20008       DS   0H                                         01-ENDIF
 0000B6 41AA 0004            00CC4          2   109          LA     $IN,4($IN)            BUMP $IN PAST CONTROL WORD  C4600000
                                          2   110          WHILE (B,0($IN),NE,0+COMMAND),DO                      C4700000
 0C00BA 47FC D13'            0C14E          3   111+         B      W220012                                         01-WHILE
 0000BE                                     3   112+W120012  DS    0H                                               01-WHILE
                                            3   113          IF     T,3($IN),X'10',ZERO,OR                          04800000
 0000BE 911D A003      00003                3   114+         TM     3($IN),X'10'                                    01-IF
 0000C2 4780 D0B'            000CF          3   115+         BC     8,OR20013                                       01-IF
                                            3   116          IF     T,3($IN),X'2F',NZERO,THEN THIS IS A DATA WORD   04900000
 0000C6 912F A003      00003                  4   117+          TM     3($IN),X'2F'                                 01-IF
 0000CA 4780 D0F'            0C1C2            4   118+          BC     8,IF20013                                    01-IF
 0000CE                                       4   119+OR20013          DS   0H                                      01-IF
 0000CE 584A 0000            0CCC0 000        4   120          L      $4,0($IN)                                     05000
 0000D2 4160 0005            0CCC5 000        4   121          LA     $6,5                 = A MAX OF 5 CHAR./WORD  05100
                                   C00        4   122          UNTIL  (BCT,$6),DO                                   05200
 0000D6                            -UNTIL          5   123+UN30015          DS  0H                                   01
 0000D6 8C40 0006            0C006 3C0000          5   124          SRDL   $4,6                                      05
 0000DA 8850 001A            0C01A 400000          5   125          SRL    $5,26                                     05
                                   500000          5   126          IF     H,($5),EQ,=X'0010',THEN  STOP PROCESSING THIS WORD  05
 0000DE 4950 D1BC            0C1D0  C1-IF             6   127+          CH     $5,=X'0010'
 0000E2 4770 D0FA            0C0FF  01-IF             6   128+          BC     7,IF30016
 0000E6 4160 0001            0CCC1  C5600000          6   129          LA     $6,1
                                   7C0000          5   130          ELSE                                             05
 0000EA 47F0 D0F6            0CCFA  01-ELSE           6   131+          B      IF30017
 0000EE                             01-ELSE           6   132+IF30016               DS  0H
 0000EE 4355 D164            0C178  C580000C          6   133          IC     $5,MCVGEBTR($5)
 0000F2 4253 5000            0CC00  05900000          6   134          STC    $5,0($3,$11)
 0000F6 5A30 D1A8            0C1BC  06000000          6   135          A      $3,CHARWIDT
                                   100000          5   136          ENDIF                                            06
 0000FA                            -ENDIF          5   137+IF30017           DS   0H                                 01
                                   C00          4   138          ENDDO                                             06200
 0000FA 4660 D0C2            00006 D00          4   139+         BCT    $6,UN30015                                 01-EN
                                          3   140          ELSE   ...........THIS IS A NEW POSITION CONTROL WORD    06300000
 0000FE 47F0 D136            0C14A  SE          4   141+         B      IF20020                                    01-EL
 000102                             SE          4   142+IF20013         DS   0H                                    01-EL
 000102 580A 0000            0CCC0 C00          4   143          L      $0,0($IN)            LOAD THE NPOW INTO GPRO 06400
 000106 8800 0006            0CCC6 C00          4   144          SRL    $0,6                                        06500
 00010A 8C00 0009            0CCC9 C00          4   145          SRDL   $0,9                                        06600
 00010E 8810 0017            00017 C00          4   146          SRL    $1,23           GPR1 NOW CONTAINS THE NEW X POSITION 06700
 000112 5400 D1AC            0C1C0 C00          4   147          N      $0,=X'000001FF'  GPRO NOW CONTAINS THE NEW Y FOSITIN 06800
 000116 5B00 D1B0            0C1C4 C00          4   148          S      $0,=F'12'          Y-12                     06900
 00011A 8800 0003            00003 CC0          4   149          SRL    $0,3               (Y-12)/8                 07000
 00011E 4C00 D1BE            0C1D2 000          4   150          MH     $0,=AL2(LMARG+NCOL+RMARG)                   07100
 000122 1830                       C00          4   151          LR     $3,$0                                       07200
```

```
    ---SCOWDHCF-------SCOWDHCF-------SCOWDHCF-------SCOWDHCF-----SCOWDHCF---                              PAGE    3

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                               ASM H V 02 05.C9 02/21/72

                                   51+*,*******************************,*                                   01-HEACC
                                   52+*,*******************************,*                                   01-HEACC
                                   53+*,**      CONTROL SECTION      **,*                                   C1-HEACC
                                   54+*,**          SCOWDHCF                    **,*                        01-HEACC
                                   55+*,*******************************,*                                   01-HEACC
                                   56+*,*******************************,*                                   C1-HEACC

 0000000A                          58 RMARG    EQU   10              = NO. COLUMNS TO RIGHT OF DISPLAY      C120C0C0
 0000001E                          59 LMARG    EQU   30              = NO. COLUMNS TO LEFT OF DISPLAY       C13CC0C0
 00000048                          60 NCCL     EQU   72              = NO. COLUMNS/LINE IN DISPLAY          C140C0C0
 0000003E                          61 NLIN     EQU   62              = NO. LINES IN OUTPUT DISPLAY          C150C0C0
 00000080                          62 IMDCW    EQU   X'80'           ... INSERT MEMOR DEVIDE                C160C000
 00000040                          63 EOMCW    EQU   X'40'           ... END OF MESSAGE                     C170C0CC
 00000008                          64 EMSCW    EQU   X'08'           ... ERASE MEMORY SUBSECTION            0180C0C0
 00000004                          65 SCCW     EQU   X'04'           ... START SMALL CHARACTERS             0190C000
 00000002                          66 LCCW     EQU   X'02'           ... START LARGE CHARACTERS             0200C000
 00000001                          67 VCW      EQU   X'01'           ... START VECTORS                      0210C000
 0000003D                          68 COMMAND  EQU   X'3D'           COMMAND WORD                           0220C0C0
 00000010                          69 NCINC    EQU   X'10'           NO MEMORY ADDR. INCREMENT              0230C000
 0000000D                          7C STRTBLAK EQU   X'0D'           START BLINK                            C240C000
 0000000E                          71 STOPBLNK EQU   X'0E'           STOP  BLINK                            0250C0C0
 0000000A                          72 $IN      EQU   $10                                                    C260C000
 00000002                          73 $YMAX    EQU   $2              ... DISP. INTO BUFFER OF LAST LIN      C276C0C0

 000074 58A1 0000        00000     75                L     $IN,0($1)       ... A( INPUT DATA )                    C290C00C
 000078 58B1 0004        00004     76                L     $11,4($1)       ... A( OUTPUT BUFFER )                 C3C0C000
 00007C 1B22                       77                SR    $YMAX,$YMAX                                            0310C000

                                   79                WHILE (9,0($IN),NE,0+COMMAND),DO   FIND  FIRST CMD WORD        C330C0C0
 00007E 47F0 0072        00C86   1    80+           B     W210003                                                 01-WHILE
 000082                          1    81+W110003    DS    0H                                                      01-WHILE
 000082 41AA 0004        00004   1    82            LA    $IN,4($IN)                                              03400000
                                      83            ENDDO                                                         0350C0C0
 000086                               84+W210003    DS    0H                                                      01-ENDCC
 000086 9530 A000  00000              85+           CLI   0($IN),0+COMMAND                                        01-ENDCC
 00008A 4770 D06E        00C82        86+           BC    7,W110003                                               01-ENDCC

                                   88                WHILE (T,3($IN),X'0'+EMSCW+EOMCW,ZERO),DO  LOOP THROUGH DATA  C370C0C0
 00008E 47F0 D156        0016A   1    89+           B     W210005                                                 01-WHILE
 000092                          1    90+W110005    DS    0H                                                      01-WHILE
                                 1    91            IF    B,3($IN),NE,0+VCW,THEN THIS IS NOT A START VECTOR MODE  03800000
 000092 9501 A003  00003           2    92+           CLI   3($IN),0+VCW                                          01-IF
 000096 4780 D146        0C15A     2    93+           BC    8,IF10006                                             01-IF
                                   2    94            IF    B,3($IN),EQ,0+SCCW,THEN SIZE = SMALL NOW              C390000C
 00009A 9504 A003  00003           3    95+           CLI   3($IN),0+SCCW                                         01-I
 00009E 4770 D096        000AA     3    96+           BC    7,IF20007                                             01-I
```

```
      ---SCOWDHCF=======SCCWDHC          3F-------SCOWDHCF--------SCOWDHCF--------SCOWDHCF---                       PAGE    4

  LOC  OBJECT CODE    ADDR1 ADDR2            3 STMT    SOURCE STATEMENT                               ASM H V 02 05.09 02/21/72

0000A2 9201 D1AB      001BF                  3   97          MVI    CHARWIDT+3,1                                         04000000
                                           2   98           ELSE                                                         04100000
000046 47F0 D0A2            00CB6            3   99+           B      IF20008                                            01-ELSE
0000A4                                       3  100+IF20007         DS   0H                                              01-ELSE
                                             3  101           IF     B,3($IN),EQ,0+LCCW,THEN SIZE = LARGE NOW             04200000
0000A4 9502 A003      00003                    4  102+          CLI    3($IN),0+LCCW                                      01-IF
0000AE 4770 D0A2            00CB6              4  103+          BC     7,IF30009                                          01-IF
0000B2 9202 D1AB      001BF          C00       4  104           MVI    CHARWIDT+3,2          SIZE NOW = LARGE             04300
                                             3  105           ENDIF                                                       04400000
0000B6                                       3  106+IF30009          DS   0H                                              01-ENDIF
                                           2  107           ENDIF                                                         04500000
0000B6                                     2  108+IF20008          DS   0H                                                01-ENDIF
0000B6 41AA 0004            00CC4          2  109           LA     $IN,4($IN)            BUMP $IN PAST CONTROL WORD       C4600000
                                           2  110           WHILE (B,0($IN),NE,0+COMMAND),DO                              C4700000
0000BA 47FC D13'            0C14E            3  111+          B      W220012                                              01-WHILE
0000BE                                       3  112+W120012   DS    0H                                                    01-WHILE
                                             3  113           IF     T,3($IN),X'10',ZERO,OR                               04800000
0000BE 911O A003      00003                  3  114+          TM     3($IN),X'10'                                         01-IF
0000C2 4780 D0B'            00CCE            3  115+          BC     8,OR20013                                            01-IF
                                             3  116           IF     T,3($IN),X'2F',NZERO,THEN THIS IS A DATA WORD         04900000
0000C6 912F A003      00003                    4  117+          TM     3($IN),X'2F'                                       01-IF
0000CA 4780 D0F                 0C1C2          4  118+          BC     8,IF20013                                          01-IF
0000C                                          4  119+OR20013         DS   0H                                           01-IF
0000CE 584A 0000            00CC0 000          4  120           L      $4,0($IN)                                          05000
0000D2 4160 0005            00CC5 000          4  121           LA     $6,5                     = A MAX OF 5 CHAR./WORD   05100
                                  C00          4  122           UNTIL (BCT,$6),DO                                         05200
0000D6                            -UNTIL         5  123+UN30015         DS  0H                                          01
0000D6 8C40 0006            00C06 300000       5  124           SRDL   $4,6                                               05
0000DA 8850 001'            00C14 400000       5  125           SRL    $5,26                                              05
                                  500000       5  126           IF     H,($5),EQ,=X'0010',THEN  STOP PROCESSING THIS WORD  05
0000DE 4950 D1BC            0C1D0  C1-IF         6  127+          CH     $5,=X'0010'
0000E2 4770 D0DA            00CFE  01-IF         6  128+          BC     7,IF30016
0000E6 4160 0001            00CC1  C56C0000      6  129           LA     $6,1
                                  7C0000       5  130           ELSE                                                      05
0000EA 47F0 D0F6            00CFA  01-ELSE       6  131+          B      IF30017
0000EE                             01-ELSE       6  132+IF30016         DS   0H
0000EE 4355 D164            0C178  C580000C      6  133           IC     $5,MCVGEBTR($5)
0000F2 4253 3000            00C00  05SC0000      6  134           STC    $5,0($3,$11)
0000F6 5430 D1AB            0C1BC  06000000      6  135           A      $3,CHARWIDT
                                  100000       5  136           ENDIF                                                     06
0000FA                            -ENDIF       5  137+IF30017         DS   0H                                             01
                                  C00          4  138           ENDDO                                                     06200
0000FA 4660 D0C2            00006 000          4  139+          BCT    $6,UN30015                                         01-EN
                                             3  140           ELSE   ;  ............THIS IS A NEW POSITION CONTROL WORD   06300000
0000FE 47F0 D136            0C14A SE           4  141+          B      IF20020                                            01-EL
000102                            SE           4  142+IF20013         DS   0H                                             01-EL
000102 580A 0000            0CCC0 C00          4  143           L      $0,0($IN)          LOAD THE NPOW INTO GPRO          06400
000106 8800 0006            0CCC6 C00          4  144           SRL    $0,6                                               06500
00010A 8C00 0009            0CCC9 C00          4  145           SRDL   $0,9                                               06600
00010E 8810 0017            0CC17 C00          4  146           SRL    $1,23          GPR1 NOW CONTAINS THE NEW X POSITION 06700
000112 5400 D1AC            0C1C0 C00          4  147           N      $0,=X'000001FF'     GPRO NOW CONTAINS THE NEW Y POSITIN 06800
000116 5B00 D1B0            0C1C4 CC0          4  148           S      $0,=F'12'          Y-12                            06900
00011A 8800 0003            00003 CC0          4  149           SRL    $0,3               (Y-12)/8                        07000
00011E 4C00 D1BE            00192 000          4  150           MH     $0,=AL2(LMARG+NCOL+RMARG)                           07100
000122 1830                       C00          4  151           LR     $3,$0                                              07200
```

```
   ---SCOWDHCF-------SCOWDHCF-------SCOWDHCF-------SCOWDHCF------SCOWDHCF---                                   PAGE    3

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                   ASM H V 02 05.09 02/21/72

                                    51+*,*******************************,*                                   01-HEADC
                                    52+*,*******************************,*                                   01-HEADC
                                    53+*,**     CONTROL SECTION      **,*                                   01-HEADC
                                    54+*,**          SCOWDHCF                       **,*                    01-HEADC
                                    55+*,*******************************,*                                   01-HEADC
                                    56+*,*******************************,*                                   01-HEADC

 0000000A                           58 RMARG     EQU   10               = NO. COLUMNS TO RIGHT OF DISPLAY    C120C0C0
 0000001E                           59 LMARG     EQU   30               = NO. COLUMNS TO LEFT OF DISPLAY     C130C0C0
 00000048                           60 NCCL      EQU   72               = NO. COLUMNS/LINE IN DISPLAY        C140C0C0
 0000003E                           61 NLIN      EQU   62               = NO. LINES IN OUTPUT DISPLAY        C150C0C0
 00000080                           62 IMDCW     EQU   X'80'            ... INSERT MEMOR DEVIDE              C160C000
 00000040                           63 EOMCW     EQU   X'40'            ... END OF MESSAGE                   C170C000
 00000008                           64 EMSCW     EQU   X'08'            ... ERASE MEMORY SUBSECTION          0180C0C0
 00000004                           65 SCCW      EQU   X'04'            ... START SMALL CHARACTERS           C190C000
 00000002                           66 LCCW      EQU   X'02'            ... START LARGE CHARACTERS           C200C000
 00000001                           67 VCW       EQU   X'01'            ... START VECTORS                    C210C000
 0000003D                           68 COMMAND   EQU   X'3D'            COMMAND WORD                         C220C0C0
 00000010                           69 NCINC     EQU   X'10'            NO MEMORY ADDR. INCREMENT            C230C000
 0000000D                           7C STRTBLAK  EQU   X'0D'            START BLINK                          C240C000
 0000000E                           71 STOPBLNK  EQU   X'0E'            STOP BLINK                           C250C0C0
 0000000A                           72 $IN       EQU   $10                                                   C260C000
 00000002                           73 $YMAX     EQU   $2               ... DISP. INTO BUFFER OF LAST LIN    C270C0C0

 000074 58A1 0000          00000    75           L     $IN,0($1)        ... A( INPUT DATA )                  C290C00C
 000078 58B1 0004          00004    76           L     $11,4($1)        ... A( OUTPUT BUFFER )               C3C0C000
 00007C 1B22                        77           SR    $YMAX,$YMAX                                           0310C000

                                    79           WHILE (9,0($IN),NE,0+COMMAND),DO   FIND FIRST CMD WORD     C330C0C0
 00007E 47F0 0072          00C86  1   80+            B     W210003                                           01-WHILE
 000082                           1   81+W110003   DS    0H                                                  01-WHILE
 000082 41AA 0004          00004  1   82             LA    $IN,4($IN)                                        03400000
                                    83           ENDDO                                                       0350C0C0
 000086                             84+W210003   DS    0H                                                    01-ENDCC
 000086 9530 A000    00000          85+          CLI   0($IN),0+COMMAND                                      01-ENDCC
 00008A 4770 006E          00C82    86+          BC    7,W110003                                             01-ENDCC

                                    88           WHILE (T,3($IN),X'0'+EMSCW+EOMCW,ZERO),DO  LOOP THROUGH DATA   C370C0C0
 00008E 47F0 0156          0016A  1   89+            B     W210005                                           01-WHILE
 000092                           1   90+W110005   DS    0H                                                  01-WHILE
                                  1   91             IF    B,3($IN),NE,0+VCW,THEN THIS IS NOT A START VECTOR MODE   03800000
 000092 9501 A003    00003          2   92+            CLI   3($IN),0+VCW                                    01-IF
 000096 4780 0146          0C15A    2   93+            BC    8,IF10006                                       01-IF
                                    2   94             IF    B,3($IN),EQ,0+SCCW,THEN SIZE = SMALL NOW        C390000C
 00009A 9504 A003    00003          3   95+            CLI   3($IN),0+SCCW                                   01-II
 00009E 4770 0096          000AA    3   96+            BC    7,IF20007                                       01-II
```

```
        ---SCOWDHCF=======SCCWDHC           3F-------SCOWDHCF-------SCOWDHCF        SCOWDHCF---                              PAGE    4

  LOC  OBJECT CODE    ADDR1 ADDR2           3 STMT   SOURCE STATEMENT                                       ASM H  V 02  05.09 02/21/72
0000A2 9201 D1AB       001BF                 3   97           MVI    CHARWIDT+3,1                                              04000000
                                          2  98          ELSE                                                               04100000
000046 47F0 D0A2             0C0B6           3   99+          B      IF20008                                                   01-ELSE
0000A3                                       3  100+IF20007         DS   0H                                                    01-ELSE
                                             3  101           IF     B,3($IN),EQ,0+LCCW,THEN SIZE = LARGE NOW                04200000
0000A4 9502 A003       00003                   4  102+           CLI    3($IN),0+LCCW                                            01-IF
0000AE 4770 D042             0C0B6             4  103+           BC     7,IF30009                                                01-IF
0000B2 9202 D1A9       001BF         C00       4  104            MVI    CHARWIDT+3,2         SIZE NOW = LARGE                    04300
                                             3  105           ENDIF                                                            04400000
000086                                       3  106+IF30009         DS   0H                                                    01-ENDIF
                                          2  107          ENDIF                                                             04500000
0C0086                                    2  108+IF20008        DS   0H                                                     01-ENDIF
000086 41AA 0004             00004        2  109          LA     $IN,4($IN)            BUMP $IN PAST CONTPOL WORD          C4600000
                                          2  110          WHILE (B,0($IN),NE,0+COMMAND),DO                                  C4700000
0C0084 47FC D13'             0C14E           3  111+          B      W220012                                                   01-WHILE
0000BE                                       3  112+W120012   DS     0H                                                        01-WHILE
                                             3  113           IF     T,3($IN),X'10',ZERO,OR                                    04800000
0000BE 91ID A003       0C003                 3  114+          TM     3($IN),X'10'                                              01-IF
0000C2 4780 D0B'             000CE           3  115+          BC     8,OR20013                                                 01-IF
                                             3  116           IF     T,3($IN),X'2F',NZERO,THEN THIS IS A DATA WORD             04900000
000006 912F A003       00003                   4  117+           TM     3($IN),X'2F'                                             01-IF
0000CA 4780 D0F              0C1C2             4  118+           BC     8,IF20013                                                01-IF
0000C                                          4  119+OR20013       DS   0H                                                     01-IF
0000CE 584A 0000             0CCC0 000         4  120            L      $4,0($IN)                                                05000
0000D2 4160 0005             0CCC5 000         4  121            LA     $6,5                        = A MAX OF 5 CHAR./WORD     05100
                                   C00         4  122            UNTIL (BCT,$6),DO                                              05200
0000D6                             -UNTIL       5    123+UN30015      DS  0H                                                     01
0000D6 8C40 0006             0C006 3C0CCC        5  124            SRDL   $4,6                                                     05
0000DA 8850 001A             0C01A 400000        5  125            SRL    $5,26                                                    05
                                   500CC0        5  126            IF     H,($5),EQ,=X'0010',THEN  STOP PROCESSING THIS WORD        05
0000DE 4950 D1BC             0C1D0  C1-IF          6  127+           CH     $5,=X'0010'
0000E2 4770 D0DA             0C0FF  01-IF          6  128+           BC     7,IF30016
0000E6 4160 0001             0CCC1  C5600000       6  129            LA     $6,1
                                   7C0CC0        5  130            ELSE                                                           05
0000EA 47F0 D0F6             0C0FA  01-ELSE        6  131+           B      IF30017
0000EE                              01-ELSE        6  132+IF30016       DS  0H
0000EE 4355 D164             0C178  C580000C       6  133            IC     $5,MCVGEBTR($5)
0000F2 4253 3000             0CC00  05C00000       6  134            STC    $5,0($3,$11)
0000F6 5A30 D1A8             0C1BC  06000000       6  135            A      $3,CHARWIDT
                                   100000        5  136            ENDIF                                                          06
0000FA                              -ENDIF       5  137+IF30017           DS   0H                                                01
                                   C00         4  138            ENDDO                                                          06200
0000FA 4660 D0C2             00006 D00         4  139+           BCT    $6,UN30015                                              01-EN
                                             3  140           ELSE  ,  ...........THIS IS A NEW POSITION CONTROL WORD         06300000
0000FE 47F0 D136             0C14A SE          4  141+           B      IF20020                                                 01-EL
000102                             SE          4  142+IF20013       DS   0H                                                     01-EL
000102 580A 0000             0CCC0 C00         4  143            L      $0,0($IN)             LOAD THE NPDW INTO GPRO            06400
000106 8800 0006             0CCC6 C00         4  144            SRL    $0,6                                                     06500
00010A 8CC0 0009             0CCC9 C00         4  145            SRDL   $0,9                                                     06600
00010E 8810 0017             0CC17 C00         4  146            SRL    $1,23                 GPR1 NOW CONTAINS THE NEW X POSITION 06700
000112 5400 D1AC             0C1C0 C00         4  147            N      $0,=X'000001FF'        GPRO NOW CONTAINS THE NEW Y FOSITIN 06800
000116 5B00 D1B0             0C1C4 CC0         4  148            S      $0,=F'12'              Y-12                                06900
00011A 8800 0003             0C003 CC0         4  149            SRL    $0,3                  (Y-12)/8                            07000
00011E 4C00 D1BE             0C1D2 000         4  150            MH     $0,=AL2(LMARG+NCOL+RMARG)                                07100
000122 1830                        C00         4  151            LR     $3,$0                                                    07200
```

```
      ---SCOWDHCF-------SCOWDHC  5          4F--------SCOWDHCF--------SCOWDHCF--------SCOWDHCF---                                 PAGE

  LOC  OBJECT CODE     ADDR1 ADDR2 /72          4 STMT    SOURCE STATEMENT                                     ASM F V 02 C5.09 02/21

000124 4133 001E             00C1E C00          4  152             LA     $3,LMARG($3)                                            07300
                                   C00          4  153             IF     F,($0),GT,($YMAX),THEN    THIS IS NEW YMAX              07400
000128 1902                        -IF            5  154+              CR     $0,$YMAX                                              01
00012A 47D0 D11C             0C130 -IF            5  155+              BC     13,IF30021                                            01
00012E 1820                        500000         5  156               LR     $YMAX,$0                                              07
                                   000          4  157             ENDIF                                                          07600
000130                             DIF          4  158+IF30021         DS   0H                                                    01-EN
000130 0610                        000          4  159             BCTR   $1,0                    (X-1)                           07700
000132 1800                        C00          4  160             SR     $0,$0                                                   07800
000134 5D00 D1B4             001C8 000          4  161             D      $0,=F'7'                (X-1)/7                         07900
000138 1A31                        000          4  162             AR     $3,$1              ((Y-12)/8)*(LINE WIDTH)+LMARG+(X-1)/7 08000
                                   CC0          4  163             IF     F,($3),GT,=A((NLIN-1)*(LMARG+NCOL+RMARG)),THEN          08100
00013A 5930 D1B8             001CC -IF            5  164+              C      $3,=A((NLIN-1)*(LMARG+NCOL+RMARG))                    01
00013E 4700 D136             0C14A -IF            5  165+              BC     13,IF30023                                            01
0C0142 41F0 0010             00C10 2C0CCC         5  166               LA     $15,16         ERROR = BAD X,Y COORDINATE             08
000146 47F0 C048             000SC 300000         5  167               B      RSCOWDHC    *              RETURN                    * 08
                                   000          4  168             ENDIF                                                          08400
00014A                             DIF          4  169+IF30023         DS   0H                                                    01-EN
                                      170 *                                                                           085000C0
                                      171 *  GPR3 NOW EQUALS THE STARTING DISPLACEMENT INTO THE BUFFER OF THE        08600000
                                      172 *                     NEXT CHARACTER                                        C87CCOCO
                                      173 *                                                                           08800000
                                           3  174          ENDIF                                                          08900000
00014A                                     3  175+IF20020          DS   0H                                                01-ENDIF
00014A 41AA 0004             00004         3  176          LA     $IN,4($IN)        .....POINT $IN TO NEXT WORD.....     09000000
                                       2  177          ENDDC                                                             09100000
00014E                                 2  178+W220012   DS    0H                                                         C1-ENDDO
00014E 9530 A000             00000     2  179+          CLI    0($IN),0+COMMAND                                           C1-ENDDO
000152 4770 D0AA             00CBE     2  180+          BC     7,W120012                                                  01-ENDDO
                                    1  181          ELSE                                                                 09200000
000156 47F0 D146             0C15A     2  182+          B      IF10027                                                    01-ELSE
00015A                                 2  183+IF10006          DS   0H                                                   01-ELSE
                                      184 *                                                                           C93CC000
                                      185 *                                                                           C94CC000
                                      186 *  PUT VECTOR LOGIC HERE                                                    C95CCOCO
                                      187 *                                                                           C96CC000
                                      188 *                                                                           C97CCOCC
                                    1  189          ENDIF                                                                05800000
00015A                              1  190+IF10027          DS   0H                                                       C1-ENDIF
                                    1  191          WHILE  (8,0($IN),NE,0+COMMAND),DO     ****                            C5900000
00015A 47F0 D14E             00162     2  192+          B      W220029                                                    01-WHILE
00015E                                 2  193+W120029    DS    0H                                                         01-WHILE
00015E 41AA 0004             00004     2  194          LA     $IN,4($IN)                          *** TEMP               10000000
                                    1  195          ENDDO                                  ****                          1C1C0000
000162                              1  196+W220029   DS    0H                                                             01-ENDDO
000162 9530 A000       00000        1  197+          CLI    0($IN),0+COMMAND                                               C1-ENDDO
000166 4770 D14A             0015E  1  198+          BC     7,W120029                                                      01-ENDDO
                                    199          ENDDO                                                                   1020C0C0
00016A                              200+W210005   DS    0H                                                               01-ENDCC
00016A 9148 A003       00003        201+          TM     3($IN),X'0'+EMSCW+EOMCW                                         C1-ENDCC
00016E 4780 D07E             00092  202+          BC     8,W110005                                                       C1-ENDCC
000172 1BFF                         203           SR     $15,$15              * NO ERRORS IN FORMATTING DATA           1030C0C0
000174 47F0 C048             0005C  204           B      RSCOWDHC *-*-*-*-*-* RETURN *-*-*-*-*-*-*-*-*-*-*-*-*          104CCOCO
```

```
   ---SCOWDHCF-------SCOWDHCF-------SCOWDHCF-------SCOWDHCF-------SCOWDHCF                    FAGE    6

 LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                               ASM H V 02 05.09 02/21/72

000178 0C                          206 MCVGEBTR DC     X'00' ********* MCVG TO EBCDIC TR TABLE **************** 1060C000
000179 F1F2F3F4F5F6F7F8            207          DC     C'1234567890* '                                         107C0000
000185 0D0F                        208          DC     X'0D0E'                                                 10800000
000187 D6                          209          DC     C'O'                   = OMEGA                          109CC000
000188 10                          210          DC     X'10'                                                   1100C000
000189 61E2E3F4F5E6E7F8            211          DC     C'/STUVWXYZ8/TPDO JKLMNOPQR(S '                         111CC0C0
000145 2D2F2F                      212          DC     X'2D2E2F'                                               112CC000
000148 4EC1C2C3C4C5C6C7            213          DC     C'+ABCDEFGHI(.L'                                        1130C000
000185 F3C4                        214          DC     C'3D'                                                   1140C000
000187 5C5C                        215          DC     C'**'                                                   115CC0CC
                                   216 ************************************************************************ 1160C0C0
                                   217 *  NCTES                                                                1170C0C0
                                   218 *                                                                       118CCGCC
                                   219 *  OMEGA (LCWER & CAPS ) =  O                                           119CC0C0
                                   220 *  GAMA                  =  G                                           12C0C000
                                   221 *  CCLCN                 =  /                                           121CC0CC
                                   222 *  THETA (CAPS)          =  T                                           122CC000
                                   223 *  PHI   (LCWER)         =  P                                           1230C0C0
                                   224 *  DELTA (CAPS)          =  D                                           1240C0C0
                                   225 *  SIGMA (LOWER)         =  S                                           125CC0C0
                                   226 *  POINTER               =  -                                           126CC000
                                   227 *  ANGLE                 =  (                                           127CC0C0
                                   228 *  LAMBDA (LOWER)        = L                                            128CC0C0
                                   229 *  PSI   (LCWER)         =  X                                           129CC0C0
                                   23C *  DEGREE                =  *                                           13CCC0C0
                                   231 ************************************************************************ 131CC0CC


000139 000000
00018C 00000000                    233 CHARWIDT DC     F'0'                                                    1330C0C0


0001C0                             235          LTORG                                                          135CC0C0
0001C0 000001FF                    236                 =X'000001FF'
0001C4 0000000C                    237                 =F'12'
0001C8 00000007                    238                 =F'7'
0001CC 000011B0                    239                 =A((NLIN-1)*(LMARG+NCOL+RMARG))
0001D0 0010                        240                 =X'0010'
0001D2 0070                        241                 =AL2(LMARG+NCOL+RMARG)
                                   242          END                                                            1360C000
```

```
                                  CROSS REFERENCE                                                    PAGE    7

SYMBOL     LEN      VALUE     DEFN    REFERENCES                                       ASM H V 02 05.09 02/21/72

CHARWIDT 00004    0001BC     0233    0097   0104   0135
COMMAND  00001  00000030     0068    0085   0179   0197
EMSCW    00001  000000C8     0064    0201
EDMCW    00001  00000040     0063    0201
FPR0     00001  00000000     0044
FPR2     00001  00000002     0045
FPR4     00001  00000004     0046
FPR6     00001  00000006     0047
IF10006  00002    00015A     0183    0093
IF10027  00002    00015A     0190    0182
IF20007  00002    0000AA     0100    0056
IF20008  00002    000086     0108    0099
IF20013  00002    0001C2     0142    0118
IF20020  00002    00014A     0175    0141
IF30009  00002    000086     0106    0103
IF30016  00002    0000EF     0132    0128
IF30017  00002    0000FA     0137    0131
IF30021  00002    000130     0158    0155
IF30023  00002    00014A     0169    0165
IMDCW    00001  00000080     0062
LCCW     00001  00000002     0066    0102
LMARG    00001  0000001F     0059    0152   0239   0241
MCVGFPTP 00001    000178     0206    0133
NCOL     00001  00000048     0060    0239   0241
NLIN     00001  0000003F     0061    0239
NOINT    00001  00000010     0069
OP20013  00002    0000CF     0119    0115
RMARG    00001  0000000A     0058    0239   0241
RSCCWDHC 00004    00009C     0018    0167   0204
SCCW     00001  00000004     0065    0095
SCDWDHCF 00001  000000C0     0012    0025
STOPBLNK 00001  0000000F     0071
STRTBLNK 00001  0000000D     0070
UN30015  00002    000096     0123    0139
VCW      00001  00000001     0067    0092
W110003  00002    0000E2     0081    0088
W110005  00002    000052     0090    0202
W120017  00002    00009E     0112    0180
W120029  00002    00015E     0193    0198
W210003  00002    000086     0084    0080
W210005  00002    00016A     0200    0089
W220017  00002    00014F     0178    0111
W220029  00002    000162     0196    0192
$0       00001  00000000     0028    0143   0144   0145   0147   0148   0149   0150   0151   0154   0156   0160   0160   0161
$1       00001  00000001     0029    0075   0076   0146   0159   0162
$10      00001  0000000A     0038    0072
$11      00001  0000000B     0039    0076   0134
$12      00001  0000000C     0040
$13      00001  0000000D     0041
$14      00001  0000000E     0042
$15      00001  0000000F     0043    0166   0203   0203
$2       00001  00000002     0030    0073
$3       00001  00000003     0031    0134   0135   0151   0152   0152   0162   0164
$4       00001  00000004     0032    0120   0124
$5       00001  00000005     0033    0125   0127   0133   0133   0134
```

```
                                        CROSS REFERENCE                                                        PAGE     8

SYMBOL     LEN    VALUE    DEFN    REFERENCES                                                      ASM H V 02 05.09 02/21/72

$6        00001 00000006  0034    0121  0129  0139
$7        00001 00000007  0035
$8        00001 00000008  0036
$9        00001 00000009  0037
$IN       00001 0000000A  0072    0075  0082  0082  0085  0092  0095  0102  0109  0109  0114  0117  0120  0143  0178  0176
                                  0179  0194  0194  0197  0201
$YMAX     00001 00000002  0073    0077  0077  0154  0156
=AL2(LMARG+NCOL+RMARG)
          00002   0001D2  0241    0150
=F((NLIN-1)*(LMARG+NCOL+RMARG))
          00004   0001CC  0239    0164
=F'12'    00004   0001C4  0237    0148
=F'7'     00004   0001C8  0238    0161
=X'000001FF'
          00004   0001C0  0236    0147
=X'0010'  00002   0001D0  0240    0127
```

BIBLIOGRAPHY

Dijkstra, E. W., "A Constructive Approach to the Problem of Program Correctness", BIT, 8, (1968) pp. 174-186.

Dijkstra, E. W., "GO TO Statement Considered Harmful", Letter to Editor, Comm. ACM, 11, (March 1968) pp. 147-148.

Mills, H. D., Structured Programming (manuscript), FSD-IBM Gaithersburg, Maryland, October, 1970.

Mills, H. D., "Syntax - Directed Documentation for PL360", Comm. ACM, 13, (April 1970) pp. 216-222.

Skylab Simulation Programming Systems User's Guide, FSD-IBM Houston, Texas, March, 1972.

PART III STRUCTURING INTERPRETER FOR A MACRO PROCESSING LANGUAGE EXTENSION (SIMPLE)

STRUCTURING INTERPRETER FOR A MACRO PROCESSING LANGUAGE EXTENSION (SIMPLE)

1. PURPOSE AND SCOPE

It is the purpose of SIMPLE to reduce the CPU and elapsed time required to expand the HLAL macros, and to extend the capabilities of the present HLAL macros.

SIMPLE is a pre-assembly processor which expands the HLAL statements before they are passed to the Assembler, thus eliminating many accesses of the macro library by the Assembler.

SIMPLE also creates a structured source listing simultaneously with the expansion of the HLAL statements.

The table below is a list of macros supported by the basic SIMPLE pre-assemble processor.

BGNCASE
BGNSEG
BGNWHILE
BSEG
CASE
DO
ELSE
ENDALL
ENDCASE
ENDDO
ENDIF
ENDLOOP
ENDSEG
ENDSRCH
ERRETURN
ERREXIT
ERRMSG
ESEG
EXITIF
IF
INSERT
ORELSE
STRTSRCH
UNTIL
WHILE

Approval	Approval fwd

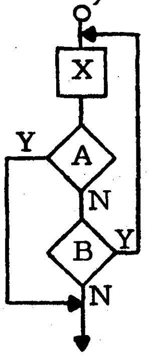
2. BACKWARD COMPATIBILITY

All HLAL statements will expand via SIMPLE in exactly the same manner as they would via the macro processor, with these exceptions.

a. The Combination Statement

```
UNTIL  (A), AND
WHILE  (B), DO
   X
ENDDO
```

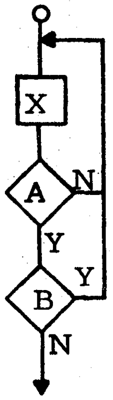
is expanded by the macro processor as,



NOTE: This logic is the same as

```
UNTIL  (A), OR
WHILE  (B), DO
```

But will be expanded by SIMPLE as,



b. The older 'T' type macro is not supported, but the newer, more commonly used 'T' type is supported.

i.e., OLD - IF T, MUD, EQ, '3', THEN
NEW - IF T, MUD, X'3', ZERO, THEN

c. The default registers used in 'IF' type statements are now \$0-\$15 and FPR0-FPR6 rather than 0-15 and 0, 2, 4, 6.

(e.g., L \$0, X rather than L 0, X.)

This notation may be changed via the REG control card (Section 5.2). Equates for the \$ and FPR must be furnished by the programmer.

(e.g., HEADC macro.)

d. If the macros are used incorrectly, the code generated by the HLAL Interpreter Extension may not coincide with the expansion of the HLAL macros. The pre-processor will attempt to create executable structured code by making logical assumptions such as the generation of needed endings (ENDIF, ENDDO, ENDSRCH, ENDLOOP) and the rejection of macros that are out of sequence.

3. ADDED CAPABILITIES

3.1 USE OF PARENTHESIS

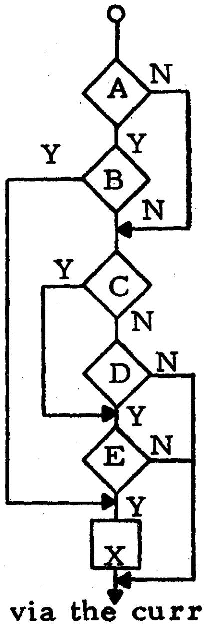
a. Parenthesis may be used or omitted when coding the IF macro.

```
e.g.,    IF   (F,A,GT,B),THEN    or
         IF    F,A,GT,B,THEN
```

b. Parenthesis may be used to form any legal chain of Boolean operations using the IF, WHILE, UNTIL, STRTSRCH macros.

(e.g., the operation (AB) + ((C + D)E) could be coded,

```
IF   (A,AND
IF   B),OR
IF   ((C,OR
IF   D),AND
IF   E),THEN
     X
ENDIF
```



c. Since parenthesis grouping is not supported via the current macro processing, Boolean operations have always expanded in a sequential manner.

e.g., AB + CD is expanded as A(B + CD)

The logical expansion would be (AB) + (CD). In order to maintain backward compatibility, SIMPLE will expand operations without parenthesis in a sequential mode unless logical is specified on an HLALEVEL control card (Section 5.3).

3.2 CONDITION CODES AND MASKS

For those occasions when the standard set of HLAL condition code mnemonics (EQ, LE, , , etc.,) do not describe the condition to be tested, a complete set of condition code and mask mnemonica has been added. They are M00 thru M15 and CC0, CC1, CC2, CC3.

e.g., The statement IF (*, ,IS, M04), THEN
Expands to BC 11, FALSELAB

3.3 ERROR PROCESSING STATEMENTS

One additional parameter has been added to the three error processing macros ERRMSG, ERRENTER, and ERRETURN. This allows the programmer to create more than one entry point for error handling. The new parameter must begin with the character $, and is of the form $X, where X is the name of an additional error entry point (maximum lentgh of X = 57 characters).

The new macro formats and expansions are:

```
                ERRENTER          $A,$X
+               B         ERREX$X       (if generated)
+ERXT$A         DS        0H
                ERRMSG            $A,$B,$X
+ERXT$A         BAL       $B,ERREX$X
                ERRETURN  $X
+ERREX$X        DS        0H
```

NOTE: All other rules previously established for these macros remain unchained.

Example:

```
*   Count a million one dollar bills from an input file.

    WHILE
        ERREXIT    (F,(COUNT),LT,MILLION), DO
        GET        A,DOLLAR
        ERREXIT    IF,B,DENOMINATION,GT,ONEDOLLAR,TOOBIG
        ERREXIT    IF,TYPE,EQ,SILVERCERTIFICATE,RECALL
        ERREXIT    IF,PICTURE,NE,WASHINGTON,COUNTERFEIT
        LA         COUNT,1(COUNT)
        BCTR       DOLLARSLEFT,0
    ENDDO
    RETURN

*
*   ERROR PROCESSING
*

        ERRENTER   RECALL
        CALL       RAREBILLCOLLECTOR
        ERRENTER   COUNTERFEIT
        CALL       TREASURYAGENT
        ERRETURN
        DISABLE    INPUT
        RETURN

*       ERRMSG     OUTOFMONEY,$MSGEXIT
        DC         CL50'FILE HAS LESS THAN ONE MILLION DOLLARS'
        ERRMSG     TOOBIG,$MSGEXIT
        DC         CL50'WRONG SIZE BILLS IN FILE'
        ERRETURN   $MSGEXIT
        MVC        MSGAREA(50),0($0)
        PUT        OUT,MSGAREA
        RETURN
```

3.4 CASE/BGNCASE/ENDCASE

3.4.1 CASE

The CASE macro has the capability of having inline segments. This modification is invoked by the keyword BEGIN which causes a return label to be generated. The BGNCASE and ENDCASE macros have been added to generate both the inline segments and the return label.

3.4.2 BGNCASE and ENDCASE

These macros are required for the extended CASE capability. The syntax for these macros is:

a. BGNCASE casename
which will generate -
+casename DS 0H

b. ENDCASE $\left\{\begin{array}{l}\text{ALL}\\ \text{casename}\end{array}\right\}$
which will generate -
+genlabel DS 0H - for the ALL option
+ B genlabel for the "casename" option

Example:

```
                CASE          $5, BEGIN, BT=(X, Y, Z)
        +       LA            14, GENLAB01
        +       B             *+4($5)
        +       B             X
        +       B             Y
        +       B             Z
        +       BGNSEG        X
        +X      DS            0H
                .
                .
                .
                ENDSEG        X
        +       BR            $14
                BGNCASE       Y
        +Y      DS            0H
                .
                .
                .
```

```
                    ENDCASE        Y
+                   B              GENLAG01
                    BGNCASE        Z
+Z                  DS             0H
                    .
                    .
                    .
                    ENDCASE        ALL
+GENLAB             1   DS         0H
```

3.5 INSERT, BSEG, And ESEG

These macros allow the user to segment his code, creating a page effect in his structured source listing. The conditional assembly instructions, AGO and ANOP, are employed to achieve this effect. The syntax for these macros is:

a. INSERT segname
which will generate -

```
+                AGO      .segname1
+.segname2           ANOP
```

b. BSEG segname
which will generate -

```
+                AGO      .segname3
+.segname1           ANOP
```

c. ESEG segname
which will generate -

```
+                AGO      .segname2
+.segname3           ANOP
```

Example: Listing page 1 -

```
                       .
                       .
                       .
                       .

                   IF A,THEN
                   INSERT     CODE
+                  AGO        .CODE1
```

```
+.CODE2    ANOP
           ENDIF
           .
           .
           .
```

Example: Listing page 2 -

```
           .
           .
           .
           BSEG        CODE
+          AGO        .CODE3
+.CODE1    ANOP
           ESEG        CODE
+          AGO        .CODE2
+.CODE3    ANOP
           .
           .
           .
           END
```

NOTE: The segmented code will be assembled inline but will appear separately in the structured source listing.

3.6 ENDALL

The ENDALL statement generates, without printing on the structured listings, closings (ENDIF's, ENDDO's, etc.) for previous IF's, WHILE's, etc.

The statement format is:

```
ENDALL     X    where X is the number of logic levels
                to close. (X = blank, closes all levels.)
```

Example:

```
IF  (A), THEN
    WHILE  (B), DO
        UNTIL  (C), DO
            IF  (D), THEN
                X
    ENDALL  3  (generates ENDIF, ENDDO, ENDDO)
    WHILE  (E), DO
        IF  (F), THEN
            X
ENDALL          (generates ENDIF, EDDDO, ENDIF)
```

4. PROCESSING ASSEMBLER CONTROL INSTRUCTIONS

The ICTL statement is always honored if it is the first statement in the input stream. At the user's option the SPACE, TITLE, and EJECT assembler instructions will also be honored. If they are honored, then spacing or page ejection will occur in the structured data set, and the words SPACE or EJECT will not appear. (See Section 5. 1.)

5. SIMPLE CONTROL STATEMENTS

SIMPLE control statements may be placed anywhere in the input stream. They are of the format *) OPERATOR OPERAND and consist of several commands.

5.1 PRINTER CONTROL FOR THE STRUCTURED DATA SET

*) SPACE $\begin{Bmatrix}\underline{\text{ON}}\\ \text{OFF}\end{Bmatrix}$

*) EJECT $\begin{Bmatrix}\underline{\text{ON}}\\ \text{OFF}\end{Bmatrix}$

*) TITLE $\begin{Bmatrix}\underline{\text{ON}}\\ \text{OFF}\end{Bmatrix}$

If ON is selected, the SPACE, EJECT, or TITLE cards will be honored for the structured listing; otherwise, these cards will be ignored.

*) STRUCTUR $\begin{Bmatrix}\underline{\text{START}}\\ \text{STOP}\end{Bmatrix}$

When the structured listing becomes so deeply nested that one statement will not fit on one print line (120 characters) the remainder or overflow will be printed, right adjusted, on the next line. If the overflow becomes too large, the structured listing may become unreadable. STRUCTUR STOP causes the structure level to be frozen at its current level. The level continues to be maintained internally and structuring will resume at the proper level when the STRUCTUR START command is received.

5.2 REGISTER CONTROL

The user may define, for use by the SIMPLE macro processor, symbolic names for any of his general purpose or floating point registers. A prefix symbol(s) may be specified for any or all fixed-point or floating-point registers. It will be the user's responsibility to set up the equates needed for these symbolic parameters.

Operator = REG
Operands = 0 → 15, FPR0 → FPR6, FIX, FLOAT,
where: 1. 0 → 15 is one of the 16 general purpose registers
2. FPR0 → FPR6 is one of the 4 floating-point registers

3. FIX specifies a prefix for all 16 general purpose registers
4. FLOAT specifies a prefix for all floating-point registers

Examples:

```
*)  REG    1=ONE, 2=TWO, 3=THREE
*)  REG    FIX=GPR, FLOAT=FP
*)  REG    0=R1, 1=F1, FLOAT=POINT
```

Example 1 equates fixed-point registers 1, 2, 3 to ONE, TWO, THREE respectively.
Example 2 equates prefix GPR to all 16 fixed-point registers as follows: GPR0, GPR1, GPR2,.....
GPR 15 and FP is prefixed to all floating-point registers
Example 3 is a combination of examples 1 and 2.

NOTE: The default values are \$1, \$2, \$3... \$15 and FPR0, FPR2, FPR4, FPR6.

5.3 "IF-TYPE" MACRO PROCESSING

The HLALEVEL macro control card allows the user to specify whether he wants SIMPLE to perform logical or sequential processing on the "IF-TYPE" macros (IF, EXITIF, WHILE, UNTIL, and STRTSRCH)

Operator = HLALEVEL

Operand = $\left\{\frac{\text{SEQ}}{\text{LOG}}\right\}$

where: SEQ = sequential processing
LOG = logical processing

"IF-TYPE" statement AB + CD would be handled one of two ways depending on whether the user specified LOG or SEQ.

if SEQ	AB+CD=A(B+CD)
if LOG	AB+CD=(AB) + (CD)

NOTE: Default is SEQ.