Z77-6050

TECHNICAL INFORMATION EXCHANGE

IBM

IBM SYSTEM/360 PROGRAMMER'S GUIDE

Mr. J. M. Bower
IBM Corporation
1439 Peachtree Street
N. E. Atlanta, Georgia 30309

March 16, 1966

This paper presents a set of standards and guidelines for systems design and programming of the System/360 using the full Operating System. These recommendations were developed for a specific installation using a Model 40 CPU, but most of the subject matter could be applied to the Basic Programming System or the Basic Operating System. The major topics are Assembly Language Programming Techniques, Coding Standards, Program Design, and Environmental Considerations. All standards are based on current pre-release specifications of the Operating System and are, therefore, necessarily preliminary.

For IBM Internal Use Only

IBM Corporation, Technical Publications Dept., 112 E. Post Road, White Plains, N.Y. 10601

Z77-6050

TABLE OF CONTENTS

INTRODUCTION

The purpose of this document is to establish standards and guidelines to be used in the design and programming of data processing applications on the IBM System/360. This initial version is concerned primarily with Assembly Language programming, but it will be extended later to cover other programming languages, tele-processing, and various related considerations.

The specific recommendations contained herein were developed for a given installation--a 262K System/360 Model 40 using Operating System/360 with option 4 and other advanced capabilities. Nevertheless, many of the standards and techniques would be equally valuable on a smaller or larger system, or one using the Basic Programming System or Basic Operating System.

It should be clearly understood that these standards were developed for one reason only--to increase the productivity of the System/360 and the people who use it. It is not intended to restrict initiative or discourage originality. By prescribing certain best or desirable techniques of programming and systems design, the installation manager encourages programmers and analysts to turn their attentions to the most important aspect of their jobs--problem definition and synthesis of a solution. When a standard no longer meets its original objective, it should be changed, but until such time as it is changed, it should be observed by all persons affected.

In writing this document, it is assumed that the reader has access to all the information on the System/360 and the full Operating System. The reader, however, need not have read all the System/360 publications to understand most of what is contained herein.

PROGRAM DOCUMENTATION

The need for documentation of a program is as obvious as the need for the program itself. This section, therefore, will describe two things: (1) the typical procedure of developing a program, and (2) what is considered acceptable documentation.

Program Preparation:

The following series of steps describes the way in which a program is developed and brought to the functional state. It should be noted that program documentation is developed during this process, not after the program is in operation.

Step 1. The programmer receives, or develops, a concise statement of the objectives of the program. Every attempt should be made to feret out all pertinent information before proceeding to the next logical step. Required at this point are the standard layout of the input data sets, as well as the format of the printed documents, punched cards, or other data sets which are to be outputs of the program.

Step 2. From the statement of the problem, the programmer plans his program and draws up a logic chart which shows the major functions to be accomplished in the program and their sequence of occurrence. This is the point at which the programmer should plan the sectioning and linkage of his program and determine what pre-coded routines are to be used, if any are applicable.

Step 3. The programmer takes his logic chart and data set layouts and develops a detailed flowchart of each logical segment of the program. He continues to study and test his logic and to revise his flowcharts as necessary until he is satisfied that he has a complete and accurate solution of the problem. Each block on the detailed flowchart should be equivalent to not more than two or three instructions in Assembly Language, or not more than one statement in PL/1, COBOL, or FORTRAN.

Step 4. The program is now coded, keypunched, and, if possible, key-verified. Then the source cards are machine listed 80-80 on plain paper. Regardless of whether the program was key-verified, the programmer should check the 80-80 listing against the coding sheets, line for line.

Program Preparation (continued)

Any uncommon or non-printing multi-punches should be verified by personally inspecting the punched card. The programmer should desk check the 80-80 listing for keypunch or coding errors and should review the flowcharts for errors in logic. A good rule is that he should desk check the program until he is quite tired of looking at it.

Step 5. The program can now be assembled or compiled, and, if there are no errors, tested. If there are any errors, they should be corrected in the source deck, and it should be re-compiled. System/360 software facilitates re-compiling only those sections (CSECT'S) of coding which are in error. The Linkage Editor can then combine the old and new sections to form an executable load module. The programmer must use his own judgment to determine whether it is better to re-compile the entire program or only the portion in error. A program is tested first on a small, but representative, collection of test data prepared by the programmer. Once it has passed this test, it can be run on a larger volume of data, which may be the live data set. After each test, a brief report should be prepared which tells the name of the program and the results of the test. Test time could best be furnished by the Operating System's job step accounting facility.

Step 6. With the program now checked out, the documentation should be reviewed to insure that it documents exactly what the program now does. The source deck can be maintained in card, tape, or disk form, depending on the facilities available and upon the frequency of references to the source deck. Object modules should be cataloged and stored in the system library for rapid call-down and execution.

Acceptable Documentation:

As previously stated, the documentation is developed as the programmer is defining, writing, and testing the program. Some of the total documentation will be maintained by the operations department for running the job, some will be maintained by the programmer for history and for making subsequent revisions, and some will be maintained by the installation manager for purposes of planning, control, and measurement. The following list prescribes the total documentation that is required without specifying at this point who will keep what information.

Acceptable Documentation (continued)

1. Identification -- the name by which the program is commonly known, as well as the identifier under which it is cataloged in the system, job, or private library.

2. Narrative -- a brief statement of what the program does.

3. Data Sets -- the standard names of all input, output, and work data sets, and their format, if not documented elsewhere.

4. Logic Chart -- showing each major segment of coding, its purpose and sequence of execution.

5. Detail Flowchart -- step-by-step flowchart of the entire program, by sections. Both the flowchart and the logic chart should be drawn using the symbols on the IBM Flowcharting Template, Form No. X20-8020.

6. Decision Tables -- if necessary, to explain the coding of complex decisions in the program.

7. Assumptions -- a list of all the significant assumptions that were made in writing the program.

8. Sub-routines -- listing of all pre-coded sub-routines used and their source (system, job, or private library).

9. Notes -- explanation of any unusual situations or exotic techniques used. If floating point, direct control, storage protection, or decimal arithmetic instructions are used, this must be stated.

10. Size -- maximum core storage used.

11. Program Characteristics --all the attributes of the coded program as described in the section "Program Design."

12. Operating Information -- set-up information, sample control cards (JOB, EXEC, DD), parameter card values, explanation of all operator messages, and normal execution time for the program, if known.

13. History -- programmer name, date assigned, date completed, number of compilations, number of tests, date and reason for major changes in requirements.

PROGRAM DESIGN

The purpose of this section is to list those attributes which an object program may possess that are significant to the way in which it is run, and to provide some guidelines for designing a program so that it will have the desired set of attributes. It is beyond the scope of this document to explain fully the concepts and terminology discussed; further clarification may be obtained by reading the various OS/360 publications. For the purpose of this discussion, a "program" is considered to be any logically related group of coding. More precisely, it may be a "task," a "segment" or a "control section," but for this discussion, the single term program is used. In addition, the term "sub-program" is used to denote a group of coding which is so small in size or so specialized in function that it cannot stand alone; it is used to perform a portion of the work in a larger "program."

Types of Sub-Programs

For the purpose of standardization, three types of sub-programs are defined for this installation: types A, B, and C.

1. A Type A sub-program is defined to be one that is available for use by any program. It shall be stored in the System Library, and it must use standard Operating System linkage conventions. Such programs should be generalized in nature and must not have access to any data sets except those made available to it by the calling program.

2. A Type B sub-program is defined to be a routine which is restricted in use to only those programs which are in a given application area, such as payroll, accounts receivable, or inventory. A Type B sub-routine must use standard Operating System linkage conventions, and if security of the routine is important, it should be stored in a Job Library for its application area. The routine should not reference any data sets which are not within its application area.

3. A Type C sub-program is by definition a small or specialized routine coded by the programmer for use in one given program only. It will be assumed to use non-standard, or Type "S" linkage (see "LINKAGE CONVENTIONS"), and it is not to be stored in any sub-program library. If the routine becomes

generalized in nature or employs standard linkage conventions, it must be re-classified as Type A or B. The Type C sub-routine may reference only those data sets used by the program in which it is included. The use of Type C sub-routines is to be discouraged because of the non-standard linkage and the lack of generality.

Types of Coding

There are three attributes of coding which affect its availability for use at a given time, as determined by the Control Program.

1. Reentrable coding is that which does not modify itself in any way during execution. All manipulation is done in the general registers or in a work area which is conveyed to it by the calling program. Because of this characteristic, reentrable coding can be shared in core by two or more tasks which are being executed concurrently. All sub-programs or programs which will be executed in a multi-tasking environment should be coded as reentrable, if there is any chance that two or more tasks could require the same coding concurrently.

2. Serially Reuseable coding is coding which will completely initialize itself whenever execution begins at its starting point. Although it cannot be shared by multiple tasks, requests for the same coding can be queued so that they will await completion of use of the coding by the using task. In this way, the same copy of the coding may be reused, and fresh copies need not be brought in to satisfy each request for it. In general, all coding which is not reentrable should be serially reuseable. It takes only a small amount of extra effort and planning to make a program or sub-program serially reuseable when it would not be otherwise.

3. Non-reuseable coding can be neither shared nor reused; a fresh copy of the coding is required each time it is requested. This makes it of less value in a multi-tasking

environment. Generally speaking, the programming counselor should approve the use of non-reuseable coding. As stated above, only a small amount of extra effort is required to make non-reuseable coding serially reuseable; with a little more planning it can be reentrable.

Program Structure

The structure of the program modules may be defined as being one of three types: simple, overlay, or dynamic.

1. A simple program structure is one in which all modules of the program are loaded at once. Execution commences after loading, and at program termination the entire area occupied by the program is made available for loading of another program. The simple program structure is the one that is easiest to use, but the storage required at any given time is equal to the size of the entire program. For this reason, simple program structure should be used only for small to moderate size programs, or those programs which contain large tables or which, for some other reason, cannot be easily segmented. The CALL macro is used for linkage between sub-programs in a simple program structure.

2. An overlay program structure defines a root segment, which resides in core during execution of the entire program. In addition, one or more additional segments may be loaded as required into a separate area of core. The maximum core storage required is the sum of the sizes of the root segment and the largest of the remaining segments of the program. Maximum core requirement is generally less than with the same program in a simple structure. If multiple regions of core are reserved for program segments, it is possible to overlap processing in the current segment with loading of the next segment. An overlay program structure must be carefully planned, but it has the advantage of reducing core storage requirements as compared to the simple structure. An overlay structure can be employed to advantage when the program has a small amount of input/output, or when a small data set can be passed against successive phases, or segments, of the program to develop the results in a series of steps. The overlay structure is defined to the

Linkage Editor by the programmer with appropriate control cards; Linkage Editor organizes the segments for loading by the Control Program as required. The macro instructions to load a segment are SEGLD and SEGWT. The CALL macro is used to link between various modules of the segments once they are in core.

3. A dynamic program structure is one in which various additional modules of coding may be requested at any time, and in any sequence, by the calling program, whereas an overlay program structure requires that a certain pattern be established for the overlays. Dynamic program design is the most flexible program design of the three possible organizations of the program modules. It requires less core than the corresponding program in a simple design, but loading of additional modules generally is not as efficient as in overlay design. Using LOAD, CALL, and DELETE macros, however, the dynamic program can operate in the same manner as an overlay program. The LINK, XCTL, and ATTACH macros provide the additional flexibility of searching the library for a load module if it is not in core and available. ATTACH will create a task associated with the coding it requests, and DETACH may be used to terminate the task and release the core occupied by the coding.

Linkage Conventions:

For use at this installation, there shall be two acceptable conventions for program linkage -- standard and non-standard.

1. "Standard" linkage is that defined by the Operating System. There are four types: I, II, III, and IV. Types III and IV are for system use, and the programmer need not concern himself with their mechanics.

   a. Type I, or direct, linkage is used to link two sub-programs of the users program when both sub-programs are in core. Although linkage could be hand-coded, standard for this installation will be the CALL macro. If a parameter list is passed, it should be treated as a variable length list, even though it actually may be of fixed length. When it is necessary to communicate to the called sub-program a given option, this should be done by means of the binary calling sequence identifier, if practical. The called sub-program shall begin with the macro SAVE (14, 12) to save all general registers. At the conclusion of the called sub-program,

control can be passed back to the calling sub-program by the RETURN macro. To indicate what action was taken by the called sub-program, a binary return code can be inserted in register 15. If the code is already loaded in 15, the macro should be RETURN (14, 12), T, RC = (15). To supply the return code (designated here by n), the macro would be written RETURN (14, 12), T, RC = n. The return code should be a multiple of 4 in the range of 0 to 4092. It is written as a decimal number in the macro, but appears in binary format in register 15.

   b. Type II, or supervisor-assisted, linkage is used to link a sub-program of the users program to another sub-program. If the called sub-program is in core and is reentrable or serially reuseable, control is passed to it there; otherwise, it is located in the library, is loaded, and then control is passed. Type II linkage is used only in a dynamic program, and the macros applicable are LINK, XCTL, and ATTACH. Detailed standards on the use of these macros will be published at a later date.

2. "Non-standard" linkage is linkage which does not use the full conventions of the Operating System. For this installation, there will be only one "non-standard" format which will be acceptable. This is designated "Type S" linkage and is illustrated by the following example:

```
* CALLING SUB-PROGRAM
          }
          LA        15, ENTRYPT
          BAL       14, Ø (15)
          }
* CALLED SUB-PROGRAM
          }
ENTRYPT   ST        14, SAVE 14
          }
          L         14, SAVE 14
          BR        14
SAVE 14   DS        F
```

The called sub-program saves register 14 for return linkage. It may use registers Ø - 2 destructively without saving them. If more registers are required, if variable information is passed to the called sub-program, or if the called sub-program becomes general in nature, the linkage should be changed to "standard" Type I or Type II linkage.

In summary, all Type A or B sub-programs will use "standard" linkage, which will be either Type I or II depending on whether the called sub-program is called dynamicly. Type C sub-programs are by nature small or specialized, such as printer overflow routine, and they will use the pre-scribed "non-standard," or Type S linkage. The objective of Type S linkage is to simplify linkage to certain frequently used routines which are applicable to one program only. However, any large or complex sub-program should use a form of "standard" linkage, even though it might be used only in one program.

Program Sectioning:

A load module is composed of one or more related control sections (CSECT), and a control section is simply a group of related coding as defined by the programmer. To insure the most efficient use of core storage, the programmer should design his program such that the load module will be relocatable, as well as scatter loadable, and each control section will be no larger than 4095 bytes in length. The sole exception is the overlay program design in which linkage editor constructs a program that is block loadable by segments, where each segment is composed of those modules which must be in core at the same time.

ENVIRONMENTAL CONSIDERATIONS

Program Parameters: Program parameters such as dates, factors, constants, etc., may best be entered in card form in the input job stream. The programmer must not rely on the operator to key in information at the console when the same data can be entered through the card reader or applicable system input device. The Operating System permits up to 40 bytes of data to be entered as the PARM= option in the job control EXEC card. Standard procedure for this installation shall be to enter all parameters for a given program in this manner, if the parameters can be contained in 39 bytes or less. A lozenge (⌑) shall precede the list of parameters to positively indentify the list to the users program. In addition, it is recommended that commas be used as delimiting characters between the parameters in the list. As an example, if it is desired to enter a date to be used in printing payroll checks, as well as another program parameter of $1,200.00, the EXEC card might be coded as follows:

```
// EXEC PGM=PAYCHK, PARM=⌑031767, 120000, ACCT=...
```

The commas and lozenge must be counted in considering the 40 character limit for this option. The users program must interpret and edit the entire list of parameters, and it is recommended that this routine be overlaid when completed. The Operating System communicates this data to the users program by storing it in core preceded by a binary half-word giving length of the list in bytes. The address of the half-word is stored in register 1 before passing control from the Operating System to the users initial sub-program.

When the program parameters cannot be contained in 39 bytes, the programmer shall define an appropriate format for a data set and enter all the parameters in card form following a DD* card in the input job stream.

All parameters should assume a default, or normal, value when they are not specifically supplied in the input job stream for a given execution of the program.

Program to Program Communication: There are several ways to communicate information automatically from program to program. Large volumes of data can be communicated as a passed data set on tape or, preferably, direct access. The Operating System, however, provides a convenient means of passing information that can be contained in 12 bits. When a job step, or program, terminates with the RETURN macro, the low order 12 bits of register 15 will be considered as the return code issued by that job step. All subsequent job steps in the same job can interogate that code by means of the COND. stepname = option of the 11 EXEC card. A job

step should issue a return code of zero to signify a normal completion, or no unusual situations encountered. The Operating System will either execute or bypass a job step depending on the value of the return codes it interogates; the return code does not actually pass to succeeding job steps.

Job Step Accounting and Statistics: The installation accounting routine should maintain the following information by jobs and job steps: elapsed clock time, active CPU time, I/O devices utilized. On the basis of this information, accounting costs can be assigned to each job, and system throughput can be measured more precisely, particularly when multi-programming. Each program, or job step, should maintain any statistics which are pertinent to it such as counts of records processed by types, turnaround time per transaction on-line, etc. These statistics can be written in the system log with a WTL macro instruction at the time of job step completion.

Operator Communication: Generally speaking, it is unnecessary for the problem program to communicate with the system operator via the console. Program parameters or options should be entered in the input job stream, and the Operating System will provide standard messages for conditions, such as I/O devices not ready that require the operator's attention. Run documentation can be written in the system log with the WTL macro instruction.

If it is necessary to communicate with the operator, the macro WTO may be used to write a message on the console, and WTOR to write a message and receive a reply. A message shall consist of a six character message code followed by two blank spaces and the text of the message in 72 characters or less.

Messages should be kept as brief as possible. If 72 characters are not enough to explain the situation, the explanation shall be found in the operators guide book. This book will list and describe, in sequence by message code, all messages issued directly by user programs at this installation. Message codes are composed of either 5 or 6 characters, defined as follows:

character	1:	application code (alphabetic); assigned by programming counselor.
	2-3:	program number (numeric); assigned by programming counselor.
	4-5:	message number (numeric); assigned by programmer.
	6:	the character A if a reply is required of the operator.

Operator replies should be kept short, preferably one character. Message codes should be so chosen that they do not duplicate any codes used by the Operating System or other IBM-supplied software.

Job Control Cards: The job control cards JOB, EXEC, and DD should be coded with the same card column conventions as assembly language program statements. The installation standards for assembly language coding are stated elsewhere in this publication. When a name field is coded on a JOB, EXEC, or DD card, the name should be 6 characters long, although an 8 character name may be used, if necessary, to provide uniqueness.

Catalogued Procedures: All regular production jobs, and common utility procedures such as compile, link edit, and execute should be catalogued in the system library. The entire procedure should be catalogued so that only a JOB card and an EXEC card will be required to execute the program or series of programs as they are normally run. Each EXEC or DD card that is catalogued should have an identifying name so that it will be possible to override the options in it by supplying an appropriate card with the same name in the input job stream.

DASD Allocation: In general, space should be allocated on direct access storage devices by relative blocks of so many cylinders or tracks. Actual assignment of given addresses should be left to the Operating System except in those special situations where throughput can be improved by controlling the actual areas assigned. In the latter case, the concurrence of the programming counselor is required, as the addresses assigned must be consistent with other device assignments of the installation.

Design of Input Data Sets: In designing card input data sets for the S/360, it should be considered that the data will always be read by the Reader/Interpreter module of the Operating System and not by the problem program. This imposes two restrictions: (1) data must be in EBCDIC format, and (2) selective stacking of input cards is not permitted. All input cards will be stacked in the same pocket of the card reader. To facilitate programming and to simplify record design, card types should be identified by numeric codes, normally one or two digits. The practice of coding special conditions by means of X punches is to be discouraged at all costs except for one situation only. A numeric field shall be considered negative if it has an X punch over its units position. If the field is positive, there shall be no zone punch in the units position. All other conditions should be coded with numeric digits in a fixed location in the card.

Design of Output Data Sets: Data sets which are to be printed or punched by the system should be designed as though they will always be processed by the Output Writer module(s) of the Operating System and will never be printed or punched on-line by the problem program. The purpose of this section is to set standards for form design and programming that will insure that this requirement is met and that will provide maximum efficiency of system operation.

I. Punched Cards: Data sets which are to be punched shall consist of one or more fixed length logical records per block. The format of each logical record shall be as follows:

1 byte - control character for Output Writer
80 bytes - data in EBCDIC mode
3 bytes - blanks (hexadecimal 4Ø)

The control character controls the pocket into which the punched card is selected. For the 2540 reader/punch the allowable characters are:

binary	decimal	hexadecimal	
00000001	1	Ø1	- stacker P1
01000001	65	41	- stacker P2
10000001	129	81	- stacker RP3

(The CNTRL macro must never be used to control stacker selection.) Unless cards must be separated, all punched cards should be selected to pocket, or stacker, P2. If a card punch error occurs, the card is automatically selected to pocket P1 on the 2540. When the output data set is blocked, the blocking factor should be a multiple of 2 if possible. Physical record size will then be a multiple of 8 bytes in length, and maximum selector channel performance will be assured for any current model of System/360. The only absolute limitation on blocking is that physical record size cannot exceed the size of the buffers allocated for the Output Writer. Normally this buffer size is 512 bytes; the exact size for this installation will be specified later.

II. Printed Documents: Data sets which are to be printed shall consist of one or more fixed length logical records per block. The format of each logical record shall be as follows:

1 byte - control character for Output Writer
132 bytes - data in EBCDIC mode
3 bytes - blanks (hexadecimal 4Ø)

Since logical record length is a multiple of 8 bytes, any blocking factor may be chosen, provided block size does not exceed the size of the buffers allocated to the Output Writer(s).

The control character controls printer spacing for each print line. The allowable control characters for a 1443 printer or any model of the 1403 printer are:

binary	decimal	hexadecimal	
00000001	1	Ø1	- suppress spacing after print
00001001	9	Ø9	- space 1 line after print
00010001	17	11	- space 2 lines after print
00011001	25	19	- space 3 lines after print
10001001	137	89	- skip to channel 1 after print
10010001	145	91	- skip to channel 2 after print
10011001	153	99	- skip to channel 3 after print
10100001	161	A1	- skip to channel 4 after print
10101001	169	A9	- skip to channel 5 after print
10110001	177	B1	- skip to channel 6 after print
10111001	185	B9	- skip to channel 7 after print
11000001	193	C1	- skip to channel 8 after print
11001001	2Ø1	C9	- skip to channel 9 after print
11010001	2Ø9	D1	- skip to channel 10 after print
11011001	217	D9	- skip to channel 11 after print
11100001	225	E1	- skip to channel 12 after print

It is evident from the above that all spacing is to be controlled in terms of spacing after print rather than before. If spacing of more than 3 lines is required, it may be necessary to put out blank lines to get the additional spacing, but this should be avoided if possible. In any event, all spacing is to be controlled with the control character; the CNTRL macro is not permitted, since it forces printing to be "on-line." For the same reason, PRTOV macro is not permitted. Page overflow must be determined by the programmer, who will maintain a line counter to keep track of the line on which he is now printing. An 11 inch form provides 66 lines at 6 lines per inch, and 88 lines at 8 lines per inch.

All documents which are printed on 11 inch long stock paper (regardless of number of parts) shall be designed to use a common carriage tape which shall be known as "S/360 Standard 11 inch." This one tape is applicable

to both 6 and 8 lines per inch spacing on any stock form 11 inches long, and it will provide considerable savings in operator setup time and effort.

The standard tape is 11" long, punched as follows:

Line No. at 6 per inch	Line No. at 8 per inch	Channel Punched
6	8	1
9	12	3
15	20	4
24	32	5
33	44	6
36	48	7
45	60	8
51	68	9
54	72	10
57	76	11
60	80	2
63	84	12

The tape is punched with all channel punches to prevent the possibility of runaway forms, and the programmer has 11 possible predetermined lines to which he can skip for headings, first detail line, total lines, page footings, etc. Channel 1 will normally denote the first print line. The programmer can simplify his own programming effort by limiting the number of channels to which he will skip, preferably using only channel 1. Since spacing and the line counter are directly related to whether spacing is 6 or 8 lines per inch, it is recommended that line spacing be a parameter of the program that can be supplied for each execution. Similarly, it may be desirable to easily change spacing of the detail lines to single, double, or triple, and this, too, could be a program parameter.

Reports on stock paper with one heading line only should normally have the following information in the heading in this order: title, date, page number. Date would normally be written as dd/mm/yy, and page number would appear as PAGE mmmm, where mmmm is the page number, 4 digits long with high order digits suppressed. The first page should be numbered 1, not Ø. However, since theprinter must be restored to channel 1 to begin printing, the programmer may find it convenient to make the first line of output a heading line with page number Ø and control character of 89 in hex. This will indicate the name of the document somewhere on the first sheet and will restore the printer to begin printing of the output.

Design of System Data Sets: System data sets are considered to be those data sets which reside on a high speed I/O device such as a drum, disk, or tape. These data sets may be designed to use any record format and access method which is supported by Operating System/360, except that FORM U records will not be permitted in this installation except for specialized applications, and then only with the concurrence of the programming counselor. Blocking, if supported, is left to the discretion of the programmer, and it may be easily changed if it is made a DCB option in the DD card. Logical record length should be a multiple of 4 bytes, and it is preferable that blocking factor be so chosen that physical record size will be a multiple of 8 bytes. This will insure maximum selector channel performance on any current model of System/360.

The choice of data management access method may be simplified by a few generalizations which will cover most cases.

1. BSAM or QSAM should be used for data sets which are processed sequentially only.
2. BDAM should be considered for a data set which can be or is always processed non-sequentially.
3. BISAM or QISAM should generally be used for data sets which must be processed both sequentially and non-sequentially.
4. BPAM is intended to handle certain special situations such as library residence, and would not normally be used in the average problem program.
5. BTAM and QTAM are specifically designed to handle telecommunications input/output only.

Allowable Character Sets: Tne characters which are allowable in input and output are determined by the input/output devices used. All EBCDIC characters are acceptable to disks, drums, and 9-track tapes (or 7-track with data conversion feature), but other I/O devices have a character set usually consisting of about 45 of the 88 EBCDIC graphics. The programmer should determine the allowable character set from the IBM manual describing the appropriate devices if in doubt.

Equipment Configuration: The exact configuration of System/360 installed at this installation will be printed in this section at a later date.

Operating System Configuration: The configuration of the Operating System, including available options, will be printed in this section at a later date.

ASSEMBLY LANGUAGE PROGRAMMING TECHNIQUES

Register Assignments

The System/360 processing unit contains 16 general purpose registers and, optionally, 4 floating point registers. The floating point registers may be used interchangeably at the programmer's discretion to perform the various floating point arithmetic functions. There are, however, certain restrictions and conventions which apply to the use of the general purpose registers. The purpose of this section is to enumerate those restrictions and conventions and to develop the installation standards for general register usage. "Restrictions" are considered to be those limitations imposed by the architecture of System/360 hardware, while "conventions" are considered to be the limitations imposed by the linkage conventions of the full Operating System software.

1. Restrictions: General register Ø may not be used as a base register or an index register. In addition, when Ø appears as operand R2 of a BALR, BCR, or BCTR instruction, branching is suppressed, regardless of the contents of register Ø. General register 1 may contain a 24 bit address stored as a result of a TRT or EDMK instruction, and general register 2 may contain a byte of data which is stored in its low order 8 bits as a result of a TRT inst ruction. The foregoing restrictions are the only ones which apply to specific registers; however, the following instructions always require an even/odd pair of (adjacent) registers: MR, M, DR, D, SLDA, SRDA, SLDL, SRDL. The instructions BXH and BXLE may be programmed to use either 2 or 3 registers. When 2 registers are used, they are R_1 (any register), and R_3 (an odd-numbered register). For the care of 3 registers, we have R_1 (any register) and an even/odd pair of registers represented by R_3 (odd) and $R_3 + 1$ (even), respectively.

2. Conventions: IBM supplied macros will use registers Ø and 1 as required in the macro expansion. The initial contents of these registers will not be saved prior to use by the macro, nor will they be cleared after use. The following describes the functions of the registers that are used for Type I and II linkages under the Operating System:

Register Assignments (continued)

Register Ø -- When the calling sub-program passes 8 or fewer bytes to the called sub-program, these are passed in registers Ø and 1.

Register 1 -- contains data passed explicitely in conjunction with register Ø if the passed parameters are 8 bytes or less. If a list of parameters is passed, register 1 contains the high order address of the parameter list.

Register 13 -- contains the address of the save area in the calling sub-program.

Register 14 -- contains the return point, or address in the calling sub-program to which control is to be passed after the called sub-program is executed.

Register 15 -- contains the address of the entry point of the called sub-program when it is called. When the called sub-program returns control to the calling sub-program, register 15 may contain, at the programmer's option, a binary return code in its low-order 12 bits.

3. Standards: When using the Operating System, it is theoretically possible to use all 16 general purpose registers if one observes the hardware restrictions outlined previously, and if one saves and restores all registers. (The standards of this installation require saving and restoring all registers except when using non-standard "Type S" linkage.) The following standards restrict, but simplify, the choice of general registers. The objective is to prevent unforeseen destruction or modification of the contents of a register.

   A. Register Ø - 2: These registers should be by the programmer only after all others are exhausted. They can be used for simple fixed point arithmetic sequences such as load, add, store, providing there are no intervening macro instructions. When TRT or EDMK instructions are used, registers 1 and 2 should be reserved for storage of addresses or data by these instructions, as noted under "Restrictions."

   B. Registers 3 - 12: These registers should be assigned in sequential blocks for any of the following uses the programmer may make of them: fixed point arithmetic, data manipulation, branching and loop control, index registers, base registers. The number of registers

reserved for base registers should not be greater than 3, even in the largest program. Lastly, all registers that are used should be assigned so that they can be initialized with a single load multiple instruction.

C. Register 13: Register 13 should not be disturbed except to insert the address of a save area when calling a sub-program with standard linkage.

D. Register 14: This register may contain only one thing--the address of the return point in the calling sub-program.

E. Register 15: Register 15 has only two uses which are allowable: (1) to contain the entry point address of a sub-program that is being called, or (2) to contain a 12-bit binary return code passed back to the calling sub-program.

As a final note, it should be clearly understood that the linkage and register conventions used by the Operating System are not the same as those used by the Basic Programming System and the Basic Operating System. BPS and BOS differ primarily in that they reserve registers 12 and 13 for the use of the Interrupt Supervisor. These differences are important if, for example, a program is tested with BOS and run with OS, or if it were run first under BPS, then adapted to OS.

Program Switches

Programmer defined switches shall be of two types, character switches and bit switches. If a sub-program or sub-routine requires only one switch, or if the switch may have more than two possible values, the programmer should define a one-byte character switch. The instruction to set the switch to a given value would be MVI. To test the switch one would use the instructions CLI and BC or CLC and BC. The most compact and most flexible two-way switch is the bit switch. One byte can be defined as eight switches, each individually addressable. By convention, a one-bit shall be defined as the "on" value of the switch, a zero bit as the "off" value. To set a switch on, the programmer should use an appropriate form of the OR instruction with an operand of one. To set the switch off, he would use a form of the AND instruction with an operand of zero. To reverse the setting of the switch, irrespective of its current setting, the programmer can use a form of the EXCLUSIVE OR instruction with an operand of one.

Choice of Arithmetic Formats

Inasmuch as the various System/360 processing units have different relative speeds of execution for fixed point, floating point, and decimal arithmetic, the choice of arithmetic format will depend upon processing unit as well as application. In the case of commercial programming on the System/360 Model 40, floating point is not particularly suitable because of the problem of conversion to and from floating point format. When a typical commercial problem is coded in fixed point and in decimal it will be found that the two solutions are approximately equal in speed and in core storage required. There are, however, several advantages of decimal arithmetic which favor its use for commercial type computations:

1) EBCDIC data can be converted to or from packed decimal format in one instruction; binary conversion requires an additional instruction in both cases.

2) The Edit and Edit and Mark instructions require that data be in the packed decimal format.

3) Packed decimal numbers will always appear as the equivalent decimal number on a hexadecimal storage printout, thus facilitating debugging.

4) Decimal fractions can be manipulated easily.

5) Field size may be as great as 31 digits, whereas a full word binary number can have a maximum equivalent size of only 9 decimal digits.

For this installation, decimal arithmetic and packed decimal format shall be standard for handling such numeric data as codes, quantities, prices, dates, etc. Fixed point arithmetic should be used only in those situations where it is particularly suitable, such as address manipulation, loop control, counting, and linkage call and return codes. Whenever fixed-point arithmetic is used, every effort should be made to use the half-word instructions to save core storage and reduce execution time.

Loop Control

The key to efficient programming of the System/360 lies in making full use of the general registers, and loop control is no exception. Basically, it is a matter of using RR format instructions instead of RX, and of using powerful looping instructions such as BCT, BXH, and BXLE instead of more lengthy sequences of instructions as coded on previous computers.

Loop Control (continued)

The following example illustrates a loop with both forward and backward branching, coded with RX, then RR format branch instructions. The RR format branch is about twice as fast as the RX format, yet the second sequence of coding is only two bytes longer than the first.

```
TEST1   CLI   CODE, C'1'          LA    3, TEST2
        BC    7, TEST2           BALR  4,Ø
        ʃ                  TEST1  CLI   CODE, C'1'
TEST2   CLI   CODE, C'2'          BCR   7,3
        B     TEST1                ʃ
                           TEST2  CLI   CODE, C'2'
                                  BR    4
```

Miscellaneous Instructions:

The following examples describe the best way to do various manipulations of the general registers. The usual criterion is minimum execution time.

1) To clear register n: SR n,n.
2) To load or store more than one register: LM or STM.
3) To load an unsigned 8-bit binary number as positive:
   SR n,n
   IC n, DATA
4) To store an 8-bit binary number:
   STC n, DATA
5) To increment (decrement) register n by a value contained in register p:
   AR n,p (SR n,p)
6) To load register n with a value V not exceeding 4095:
   LA n,V(Ø)
7) To increment register n by a value V which is not contained in a register and which does not exceed 4095:
   LA n,V(n)

In situations where storage to storage operations involve only one character, execution time and core storage can be saved by using the SI format instructions: NI, CLI, XI, MVI, and OI instead of SS format instructions with a length of 1 (NC, CLC, XC, MVC, and OC).

DCB MACRO: To insure a maximum degree of device independence, and to provide flexibility in selecting buffers, blocking, etc., at run time, the DCB macro should be coded with only that information that cannot be supplied conveniently from the DD card or data set label. In general this information will consist of symbolic name of the DCB, data set organization, and macro type code (MACRF=...). The remainder of the pertinent DCB

options should be supplied in the job control DD card, which should be stored in the system in a catalogued procedure. Changes to the DCB specifications in the catalogued DD card can be made by introducing a DD card in the input job stream, containing only those options which are to be changed.

ASSEMBLY LANGUAGE CODING STANDARDS

The following standards are intended to prescribe the details of coding an Assembly Language program. Considerations such as instruction usage and linkage conventions are covered in other sections of this publication; the objective of this section is to describe how to fill out a coding sheet for a given program.

1. Coding Form -- The most recent revision of IBM Form No. X28-6509, "System/360 Assembler Coding Form," is to be used for coding of all Assembly Language programs. The Assembly Language statement ICTL is not to be used to change card format from that shown on the coding form.

2. Coding of Characters -- For the sake of accurate keypunching, it is essential that characters be coded legibly and in a unique fashion. There are a number of pairs of characters whose coding can lead to confusion. Some of these are Ø and 0, V and U, B and 8, Z and 2, 0 and D, P and D, S and 5, 1 and I. The standard coding of numeric characters is as follows:

   Ø, 1, 2, 3, 4 or 4, 5, 6, 7, 8, 9.

   Alphabetic characters are to be coded as follows:

   A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z.

   All alphabetic characters are to be coded as upper-case letters. (The use of the EBCDIC lower case letters, though possible on S/360, is not envisioned for this installation at this time.) Special characters are to be coded as nearly as possible like the EBCDIC symbols shown on the "System/360 Reference Data" card, Form No. X20-1703. The programmer is specifically cautioned, however, that not all the EBCDIC special characters are acceptable to all System/360 input/output units. The specific special characters that may be used will depend, therefore, upon the I/O units with which the job is to be run. There is no character set limitation imposed by magnetic tape or direct access storage devices; limitations of the other I/O devices are defined in this publication under "ALLOWABLE CHARACTER SETS."

3. Labels -- All programmer assigned labels, or symbols, must begin with an alphabetic character and may not be greater than 8 characters in length. The following conventions are to be observed by the programmer in assigning various types of labels in a program.

   a. External Symbols -- All symbols which will be placed in the External Symbol Dictionary (ESD) by the Assembler (e.g., the label of a CSECT statement) must be 6 to 8 characters in length, of which 1 or 2 characters may be numeric, the rest alphabetic. The first, or entry point, label in every Class A or B sub-routine is to be assigned according to this convention. Class A and B sub-routines are defined in this publication under "Types of Sub-Programs."External symbols must be unique within the context in which the coding is used. Thus, a sub-routine which may be used by the entire installation must have external symbols which are absolutely unique (different from all other external symbols in use in the installation), while a sub-routine used only on payroll programs will have external symbols which need only be unique within the scope of all payroll programs. The uniqueness of a symbol can be tested by reviewing the ESD in the assembly listings of all subroutines and sub-programs that are in the same scope of application as the proposed symbol.

   b. Labels of Procedural Steps -- The labels of all instructions or macros which are not assigned according to the convention for external symbols (see "a" above) shall consist of two alphabetic characters (which identify the segment of coding) and four digits (identifying the statement within the segment of code). The symbols are to be assigned initially by the programmer as he needs them, spaced by intervals of ten. Thus, the initial assembly of a source module ALPHA containing two logical segments of coding might have the following sequence of labels -- ALPHAØ1, AAØØ1Ø, AAØØ2Ø, AAØØ3Ø, BBØØ1Ø, BBØØ2Ø. Labels of procedural steps must be unique within a given assembly. This convention facilitates providing the required uniqueness.

c. Labels of Data -- The programmers shall assign labels to data and work areas at his discretion, using the following suggestions as guidelines. . .

To facilitate providing the required uniqueness within the assembly, data labels should be at least 5 characters long. The label itself should be descriptive of the field it names, preferably all alphabetic. Coding of the program may be easier if corresponding fields in different records or work areas are assigned labels which differ only by a single character prefix such as I for input, O for output, M for master, T for transaction, W for work area, H for hold area. Thus, one can sequence check IACCTNO and HACCTNO, or add TAMOUNT to MAMOUNT and move the result to OAMOUNT.

4. Operation -- The operation field of the coding form is used for the instruction and macro mnemonics, as well as other assembly language statements such as DS, DC, ORG, TITLE, CSECT, etc. In the case of the Branch On Condition instruction, the programmer should use the extended mnemonic codes provided by the assembler whenever they are applicable. This will improve the readability of the program in addition to simplifying the construction of the appropriate mask for the Branch On Condition instruction. The productive programmer will use all the facilities of the Assembler to minimize his coding effort and the possibility of error.

5. Operand -- The following conventions are to be followed in coding the OPERAND portion of a statement, instruction, or macro.

a. All core storage is to be addressed symbolicly so that all object modules will be relocatable. Actual core addresses will not be permitted, nor are they needed.

b. Character adjustment of addresses is to be avoided whenever possible because of the possibility of error in computing the adjustment. Branching by means of

the location counter and a character adjustment is likewise a questionable technique that should be avoided.

c. General and floating point registers are not to be assigned symbolic names. They should always be addressed by their actual number in 1 or 2 digit form as needed. The mnemonic op code of the instruction is sufficient to indicate that the numbers represent general or aloating point registers.

d. Base registers are not to be coded in actual form except in instructions which manipulate the base register value, such as Load, Add, BXLE. The USING and DROP statements are to be used to denote the base registers that may be used by the assembler, as well as their assumed values.

e. The use of literals is encouraged to reduce coding effort. With fixed-point arithmetic, half-word literals should be used where possible to reduce execution time and core storage required.

6. Comments -- If comments are to be included in the same card as a statement, instruction, or macro, they should be coded beginning in card column 41 and extending to card column 71. If the text of the comment is a continuation from the comments portion of the previous card (regardless of whether a continuation indicator character was punched in column 72 of the previous card), the comment should be coded and punched beginning in column 43. In general, frequent comments are desirable. Comments on instructions help explain what is being done or how; comments on DC or DS statements are useful to explain the contents of the field, particularly if it is not evident from the label of the field.

7. Identification -- A three-character program deck identification will be punched in card columns 73-75. The first two columns will contain a two-digit number which is assigned by the programmer when the program is undertaken. Column 75 will contain an alphabetic character designating the programmer. A given programmer, then, might identify his assemblies (programs or sub-routines)

Assembly Language Coding Standards (continued)

as EØ1, EØ2, EØ3, etc. The programmer -- identifying character will be assigned to each programmer in the installation by the person designated as the programming counselor.

8. Sequence -- Card columns 76-80 will contain a program card sequence number consisting of a two-digit page number followed by a three-digit line number. The last digit of line number should be Ø initially to allow for insertions. In planning his program, the programmer should allocate one block of page numbers for instructions and one block for data definitions and work areas, in order that instructions and data defining statements can be assembled together as only two separate groups of coding. When a program is assembled, the statement ISEQ 73, 80 is to be used to sequence check all the cards in the source deck.

9. Comments Cards -- Lengthy comments can be included in the program by coding an asterisk in card column 1 and the comment in columns 4-71. If the text of a particular paragraph extends beyond one card, the comments cards after the first should have the continued text coded in columns 6-71. The programmer should include comments cards at the beginning of each assembly to identify the source module, programmer and date, and at the beginning of each group of coding to briefly describe what it does and any special situations or unusual techniques. Comments cards should describe at the beginning of a routine what each general purpose register is used for and its initial or normal contents. Usage of the floating point registers should be described in those routines which use the floating point instructions.