CS 91

THE PL360 SYSTEM

BY

NIKLAUS WIRTH

JOSEPH W. WELLS, JR.

EDWIN H. SATTERTHWAITE, JR.

TECHNICAL REPORT NO. CS 91
APRIL 1, 1968

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

THE PL360 SYSTEM

By

Niklaus Wirth

Joseph W. Wells, Jr.

Edwin H. Satterthwaite, Jr.

Computer Science Department

Stanford University

Stanford, California

THE PL360 SYSTEM

Table of Contents

Table of Contents

1. Introduction and Survey

This report describes the use of two operating systems which serve as environments for the PL360 language defined in the companion report CS 53 [1]. Some additions to that language, not described in CS 53, are documented in the Appendix. One of the systems is a stand-alone, self-loading program specifically designed for PL360; the other is a subsystem operating under IBM's Operating System/360 (OS). With the minor exceptions noted in Chapter 5, these two systems were designed to be entirely compatible at the source language level.

The core of both systems is a job sequencer which accepts batches of jobs and receives instructions in the form of control cards. These job control instructions are described in Chapter 2. A collection of standard programs, including the PL360 compiler, is provided in each system. The use of these programs is described in Chapter 3; in particular, section 3.1 defines the symbol representations, the restrictions imposed on the language by the implementation, the available system functions (particularly those for input and output), and the usage of the compiler.

Chapter 4 contains further information about the environment provided. Storage organization is described, and the reaction of the system to the occurrence of program checks and other unintended events is indicated.

Chapter 5 outlines the use of the PL360 OS subsystem, including a description of the OS deck setup and Job Control Language statements required. Chapter 6 describes the use of the PL360 stand-alone system,

including information for the computer operator.

While Chapters 2-6 are intended to serve as a user's reference manual, Chapter 7 contains information about the configuration requirements and internal organization of the two systems. Knowledge of this chapter is not required for the routine use of either system.

The PL360 operating environment was designed with the goal of providing a convenient, efficient, and easy-to-use tool for the development of compilers and operating systems. This goal is reflected in the set of standard library programs. In addition to listing and card duplicating programs, there are programs to maintain and edit tape files, which the compiler and other programs are able to accept as input in place of cards. A system generation program is also provided to rapidly create or update system program libraries. Cross-reference listings (in any language, and PL360 in particular) are produced by another library program. Finally, a syntax processor is included to facilitate the construction of compilers based on the principle of analysis of precedence grammars.

A further program available under the system is the Algol W compiler. The Algol W language and the use of that compiler are described in the companion report CS 89 [2].

Development of the two systems was directed by Professor Niklaus Wirth. The stand-alone system and most of the library programs were implemented by Mr. J. Wells. Mr. E. Satterthwaite developed the OS subsystem, and Mrs. J. Keckler wrote the cross-referencing library program.

This project was supported in part by the National Science Foundation grant GP 6844.

2. Job Control Instructions

Jobs in the input batch are separated by control cards to be interpreted by the job control routine. These cards are characterized by a 0-4-8 punch (denoted by "%", the 029 keypunch graphic) in column 1 and, if encountered by the READ routine, give rise to an end-of-file indication (see section 3.1.2.4). Information contained in columns 2-9 (left adjusted) of control cards is inspected and interpreted. With the exception of an EOF card, such a control card is assumed to mark the beginning of a new job.

EOF — This control card merely causes an end-of-file indication to be given. It is to be used at the end of an object deck or of a compiler source deck.

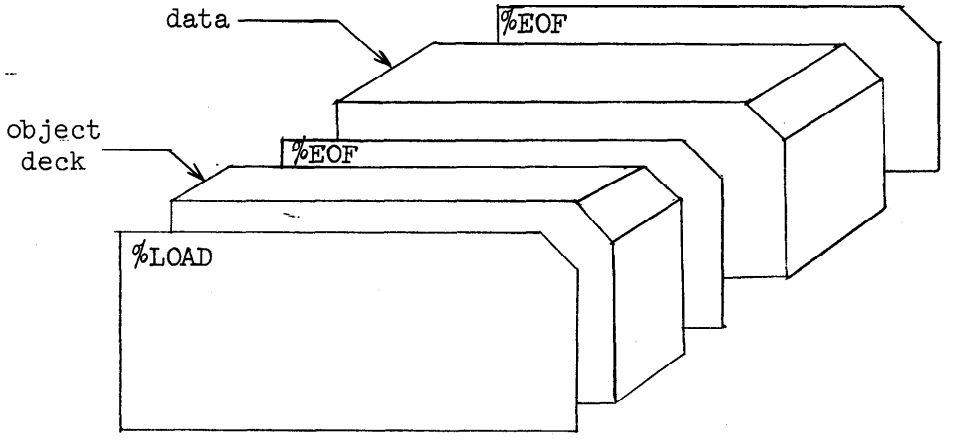
A time limit for each job may optionally be specified on the corresponding control card. Card columns 10-17 are used for such specification. Within that field, the following forms of time limit specification are allowed:

```
<time limit>  ::= <minutes> | <minutes> : <seconds>
<minutes>     ::= <unsigned integer> | {empty}
<seconds>     ::= <unsigned integer> | {empty}
```

An empty field is given the value zero. If the time limit field is blank or has a value of zero, the job is allowed unlimited time. Otherwise, execution of the job is automatically terminated upon expiration of the designated time interval if necessary. Any information in columns 18-80 of control cards is ignored.

Previously produced object decks may be loaded and executed by use of the LOAD control card.

LOAD — The subsequent cards should be an object deck punched by a compiler (see section 3.1.6). They are loaded and execution of the loaded program is initiated. An input deck using the LOAD instruction has the following composition.



Note: The second EOF card is not required if there is no data.

Any other text contained in columns 2-9 is interpreted as the name of a program to be loaded from the system library. The following standard library programs are described in the following chapter:

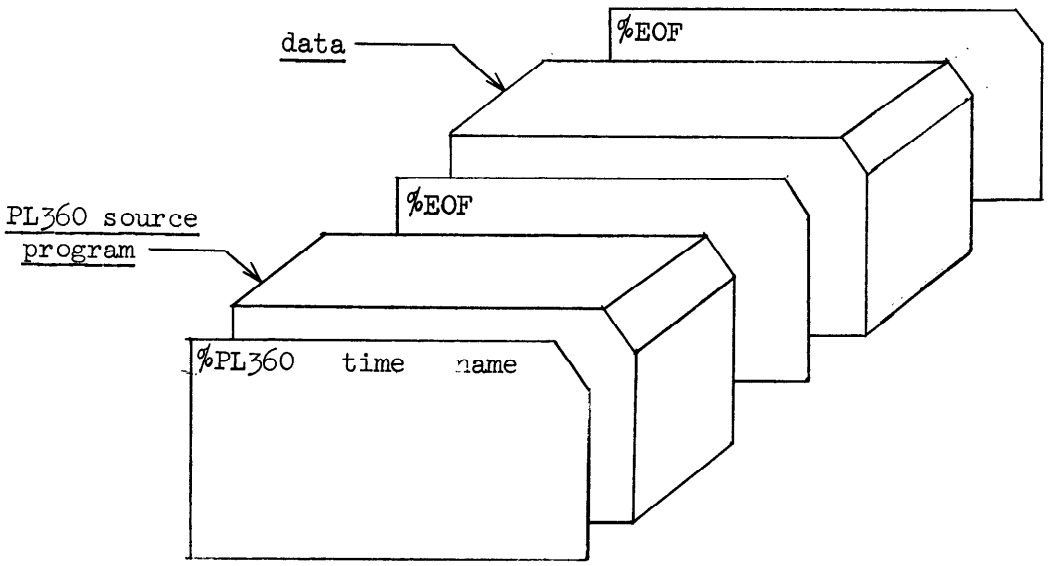
PL360	PL360 compiler
DUPDECK	Program to duplicate card decks
LISTER	Program to list card decks
TUP	Tape updating program
SYSTUP	System tape updating program
SFTUP	Source file tape updating program
SYNPROC	Syntax processor program
XREF	Identifier cross-referencing program

Use of the ALGOL W compiler is described in the companion report CS 89 [2].

3. Description of System Programs

3.1. The PL360 Compiler (PL360)

3.1.1. The Form of Input Decks



Note:
1. If there is no data, only one %EOF card is needed.
2. If a PL360 object deck is inserted between the source deck and the first %EOF card, then after the compiled program has been loaded, that object program is loaded up to the first %EOF card. The combined programs are then executed. This facility can be used in connection with the external and global segment facilities of PL360 (see the Appendix).

3.1.2. The Language

The PL360 programming language is described in a companion report [1]. Revisions and additions to the language are described in the Appendix of this report. Details pertinent to the present implementation (i.e., symbol representations, standard identifiers, and specific limi-

tations) are contained in the subsequent paragraphs.

3.1.2.1. Symbol Representation

Only capital letters are available. Basic symbols which consist of underlined letter sequences in the report [1] are denoted by the same letter sequences without further distinction. As a consequence, they cannot be used as identifiers. The basic symbols are:

```
+    -    *    /    (    )    =    <    >
,    ;    .    :    @    #    _    "    '
:=   <=   >=   ¬=
DO   IF   OF   OR
ABS  AND  END  FOR  NEG  SYN  XOR
BASE BYTE CASE ELSE GOTO LONG NULL
REAL SHLA SHLL SHRA SHRL STEP THEN
ARRAY BEGIN SHORT UNTIL WHILE
GLOBAL
COMMENT INTEGER LOGICAL SEGMENT
EXTERNAL FUNCTION OVERFLOW REGISTER
CHARACTER PROCEDURE
```

3.1.2.2. Standard Identifiers

The following identifiers are predeclared in the language, but may be redeclared due to block structure. Their predefined meaning is specified in the language report [1], in section 3.1.2.4, or in the Appendix.

```
MEM
B1   B2   B3   B4   B5   B6   B7   B8   B9
B10  B11  B12  B13
R0   R1   R2   R3   R4   R5   R6   R7   R8
R9   R10  R11  R12  R13  R14  R15
F0   F2   F4   F6
F01  F23  F45  F67
```

LA MVI MVC CLI CLC LM STM
SLDL SRDL IC STC CVD UNPK
ED EX TR SET RESET TEST
READ WRITE PUNCH PAGE
READTAPE WRITETAPE REWIND MARKTAPE
FSPTM FSPREC BSPTM BSPREC
READTYPE WRITETYPE
SETPMASK SETDUMP DUMP
GETTIME WRITETIME

3.1.2.3. Restrictions

The implementation imposes the following restrictions upon the language:

a. Only the first 10 characters of identifiers are recognized as significant.

b. No goto statement may refer to a label defined in a segment different from the one in which the goto statement occurs.

3.1.2.4. Supervisor Functions and Standard Procedures

A set of standard functions is defined for elementary input and output operations. The referenced supervisor routines make use of parameter registers as specified below. They set the condition code to 0, unless otherwise specified. Input-output devices are designated by logical unit numbers (see section 5.1 or 6.1).

READ	Read a card, assign the 80 character record to the memory area designated by the address in register R0. Set the condition code to 1 if a control card is encountered.
WRITE	Write the record of 132 characters designated by the address in register R0 on the line printer.

	Set the condition code to 1 if the next line to be printed appears on the top of a new page.
PUNCH	Punch the record of 80 characters designated by the address in register R0 on the card punch.
READTAPE	Read a record from the tape unit specified by the logical unit number in register R2. The length of the record in bytes is specified by register R1, and it is assigned to the memory area designated by the address in register R0. Set register R1 to the actual number of bytes read. Set the condition code to 1 if a tape mark is encountered.
WRITETAPE	Write a record on the tape unit specified by the logical unit number in register R2. The length of the record in bytes is specified by register R1; the record is designated by the address in register R0.
PAGE	Begin a new page with the next record written on the line printer.
READTYPE*	Read a record from the operator console typewriter. The length of the record in bytes is specified by register R1, and it is assigned to the memory area designated by the address in register R0. Set register R1 to the actual number of bytes read.
WRITETYPE*	Write a record onto the operator console typewriter. The length of the record in bytes is specified by register R1; the record is designated by the address in register R0.

* Available only in the stand-alone system.

The following are tape handling functions. They affect the tape unit specified by the logical unit number in register R2.

MARKTAPE	Write a tape mark.
REWIND	Rewind the tape.
BSPREC*	Backspace one record.
FSPREC*	Forwardspace one record.
BSPTM	Backspace past the previous tape mark.
FSPTM	Forwardspace past the next tape mark.

The system also provides a set of standard procedures.

DUMP	Print a specified area of memory in hexadecimal form. The starting address of the area is specified by register R0. Its length in bytes is specified by the register R1. The values of register R2 and the condition code are altered by a call of the dump routine.
WRITETIME	Print the time elapsed since the beginning of execution of the present program (in minutes, seconds, and sixtieths of a second). The values of registers R0, R1, and R2 and of the condition code are altered.
GETTIME	Set register R1 to the time elapsed since the beginning of execution of the present program (in sixtieths of a second). The values of registers R0 and R2 and of the condition code are altered.

In addition, the standard procedures SETPMASK and SETDUMP give the user some control over program interruptions; they are described in section 4.2.

---

* Available only in the stand-alone system.

```
%PL360     :10

01 0004   00 0170                BEGIN COMMENT MAGIC SQUARE GENERATOR;
01 0004   00 0170                   ARRAY 132 BYTE LINE = 132(" ");
01 0004   00 01F4                   ARRAY 8 BYTE PATTERN = (" ",3(#20),#21,#20);
01 0004   00 01FC                   LONG REAL DEC;
01 0004   00 0208                   ARRAY 256 INTEGER X;
01 0004   00 0608
01 0004   00 0608                PROCEDURE MAGICSQUARE (R6);
01 0008   00 0608                   COMMENT THIS PROCEDURE ESTABLISHES A MAGIC SQUARE OF ORDER N,
01 0008   00 0608                      IF N IS ODD AND 1 < N < 16. X IS THE MATRIX IN LINEAR FORM.
01 0008   00 0608                      REGISTERS R0 ... R6 ARE USED, AND R0 CONTAINS N AS PARAMETER.
01 0008   00 0608                      ALGORITHM 118 (COMM.ACM, AUG.1962);
01 0008   00 0608                   BEGIN SHORT INTEGER NSQR;
01 0008   00 060A                      INTEGER REGISTER N SYN R0, I SYN R1, J SYN R2, Y SYN R3;
01 0008   00 060A                      INTEGER REGISTER IJ SYN R4, K SYN R5;
01 0008   00 060A                      NSQR := N; R1 := N * NSQR; NSQR := R1;
01 0016   00 060A                      I := N+1 SHRL 1; J := N;
01 0022   00 060A                      FOR K := 1 STEP 1 UNTIL NSQR DO
01 0026   00 060A                      BEGIN Y := I SHLL 6; IJ := J SHLL 2 + Y; Y := X(IJ);
01 003C   00 060A                         IF Y ¬= 0 THEN
01 0042   00 060A                         BEGIN I := I-1; J := J-2;
01 004A   00 060A                            IF I < 1 THEN I := I+N;
01 0054   00 060A                            IF J < 1 THEN J := J+N;
01 005E   00 060A                            Y := I SHLL 6; IJ := J SHLL 2 + Y;
01 006C   00 060A                         END ;
01 006C   00 060A                         X(IJ) := K;
01 0070   00 060A                         I := I+1; IF I > N THEN I:= I-N;
01 007C   00 060A                         J := J+1; IF J > N THEN J := J-N;
01 0088   00 060A                      END ;
01 0094   00 060A                   END ;
01 0096   00 060A
```

```
01 0096   00 060A                PROCEDURE GENERATE (R8);
01 0096   00 060A                   BEGIN INTEGER REGISTER I SYN R1, J SYN R2, IJ SYN R4, N SYN R6;
01 0096   00 060A                      J := 0; FOR I := 0 STEP 4 UNTIL 1020 DO X(I) := J;
01 00B2   00 060A                      MAGICSQUARE;
01 00B6   00 060A                      N := R0;
01 00B8   00 060A                      FOR I := 1 STEP 1 UNTIL N DO
01 00BC   00 060A                      BEGIN IJ := I SHLL 6 +4; R5 := @LINE(4);
01 00CE   00 060A                         FOR J := 1 STEP 1 UNTIL N DO
01 00D2   00 060A                         BEGIN MVC(5, B5, PATTERN); R3 := X(IJ); CVD(R3, DEC);
01 00E4   00 060A                            ED(5, B5, DEC(5)); IJ := IJ+4; R5 := R5+7;
01 00F2   00 060A                         END ;
01 00FC   00 060A                         R0 := @LINE; WRITE;
01 010C   00 060A                      END ;
01 0116   00 060A                      MVC(130, LINE(1), LINE(0)); WRITE;
01 0128   00 060A                   END ;
01 012A   00 060A
01 012A   00 060A                   R0 := 3; GENERATE;
01 0132   00 060A                   R0 := 5; GENERATE;
01 013A   00 060A                   R0 := 9; GENERATE;
01 0142   00 060A                END .

SEGMENT 00 STARTS AT 07C810
SEGMENT 01 STARTS AT 038920

ELAPSED TIME IS 00:02:00
```

```
 4     3     8
 9     5     1
 2     7     6

11    10     4    23    17
18    12     6     5    24
25    19    13     7     1
 2    21    20    14     8
 9     3    22    16    15

37    36    26    16     6    77    67    57    47
48    38    28    27    17     7    78    68    58
59    49    39    29    19    18     8    79    69
70    60    50    40    30    20    10     9    80
81    71    61    51    41    31    21    11     1
 2    73    72    62    52    42    32    22    12
13     3    74    64    63    53    43    33    23
24    14     4    75    65    55    54    44    34
35    25    15     5    76    66    56    46    45
```

ELAPSED TIME IS 00:02:01

3.1.3. Instructions to the Compiler

The compiler accepts instructions inserted anywhere in the sequence of input records. These instructions affect subsequent records. A compiler instruction card is marked by a $ character in column 1, and an instruction in columns 2-20. Columns 21-80 of such a record are ignored.

$NOGO	Compile, but do not attempt execution.
$LIST	List source records on the printer (initial option).
$NOLIST	Do not list source records.
$PUNCH	Punch compiled program and data segments on cards.
$NOPUNCH	Do not punch compiled program and data segments (initial option).
$PAGE	Print the next record of the listing on a new page.
$0	Print the source text only (initial option).
$1	Indicate the addresses of all variables and procedures upon their declaration.
$2	List addresses as for $1. Also list the produced machine code in hexadecimal notation.
$TAPEn	Read subsequent source records from the tape unit with logical number n. If n is omitted, device 7 is assumed. Columns 10-17 of the card specify the program name. If the name field is not blank, the tape is searched for that program from the starting point to the end. Column 19 specifies the rewind option (see section 3.6.1). $TAPEn is the last card read by PL360 from the card deck.

$OUTPUTn	Place the compiled program on logical device n, and do not attempt execution. If n is omitted or no $OUTPUT card is used, the output is put on device 5. Column 19 specifies the rewind option. Note that PL360 produces an unlabeled object program (see section 3.5.1).

3.1.4. Compiler Output Listing

If listing has been specified, the compiler lists each source card as it is read. Source card images read from tape also include the sequence number used by TUP and SFTUP (see sections 3.4 and 3.6). At the left edge of the page, the compiler lists two sets of numbers. The first set consists of the current program segment number (in decimal) followed by the current object code relative address (in hexadecimal); the second set, of the current data segment number followed by the current data relative address.

3.1.5. Error Messages of the Compiler

Errors detected by the compiler are indicated by a message and a bar below the character which was last read. After 51 errors in any program, compilation is terminated. If listing has been specified, the remainder of the program is simply listed.

Error No.	Message	Meaning
00	SYNTAX	The source program violates the PL360 syntax. Analysis continues with the next statement.
01	VAR ASS TYPES	The type of operands in a variable assignment are incompatible.
02	FOR PARAMETER	In a for clause, the register is not an integer register, the step is not an integer or short integer number, or the limit is not an integer register, cell, or number or short integer cell or number.
03	REG ASS TYPES	The types of operands in a register assignment are incompatible.
04	BIN OP TYPES	The types of operands of an arithmetic or logical operator are incompatible.
05	SHIFT OP	A real instead of an integer register or number is specified in a shift operation.
06	COMPARE TYPES	The types of operands in a compare are incompatible.
07	REG TYPE OR #	Either the type or the number of the register used is incorrect.

Error No.	Message	Meaning
08	UNDEFINED ID	An undeclared identifier is encountered. The identifier is treated as if it were "R1". This may generate other errors.
09	MULT LAB DEF	The same identifier is defined as a label more than once in the same block.
10	EXC INI VALUE	The number of initializing values exceeds the number of elements in the array.
11	NOT INDEXABLE	The function argument does not allow for an index register.
12	DATA OVERFLOW	The address of the declared variable in the data segment exceeds 4095.
13	NO OF ARGS	An incorrect number of arguments is used for a function.
14	ILLEGAL CHAR	An illegal character was encountered; it is skipped.
15	MULTIPLE ID	The same identifier is declared more than once in the same block. This occurrence of the identifier is ignored.
16	PROGRAM OFLOW	The current program segment is too large. It must be resegmented.
17	INITIAL OFLOW	The area of initializing data in the compiler is full. This can usually be circumvented by suitable data segmentation or by reordering initialized data within the segment.

Error No.	Message	Meaning
18	ADDRESS OFLOW	The number used as index is such that the resulting relative address is less than 0 or greater than 4095.
19	NUMBER OFLOW	The integer number is too large in magnitude.
20	MISSING	An end-of-file has been read before a program terminating "." was encountered. The problem may be a missing string quote.
21	STRING LENGTH	The length of a string is either 0 or greater than 256.
22	AND/OR MIX	A compound condition must not contain both ANDs and ORs.
23	FUNC DEF NO.	The format number in a function declaration is illegal (see the Appendix).
24	ILLEGAL PARAM	A parameter incompatible with the specifications of the function is used (see the Appendix).
25	NUMBER	A number has been used that has an illegal type or value.
26	SYN MIX	Synonym declarations cannot mix cell and register declarations.

At the end of each program segment, all occurrences of undefined labels are listed with an indication of where they occurred.

3.1.6. The Format of "Binary" Cards

The compiler produces four types of "binary" cards if requested through the $PUNCH option. The card formats are:

Col 1	This column identifies the type of object card S = procedure segment header D = data segment header E = external procedure or data segment header P = object program card
Col 2	Segment number in hexadecimal.
Col 3-6	Length of segment if S, D, or E card. Relative address of first byte of object program on the card if P card.
Col 7-8	Count of object program bytes on card if P card. Blank if S, D, or E card.
Col 9-72	Object program bytes if P card. Date is in Col 40-47 if S, D, or E card.
Col 73-74	Segment number in decimal.
Col 75	Type of segment (E, D or S).
Col 76-80	Sequence number in decimal. For each segment sequence numbers start with 00001 and are incremented by 1.

Note: Columns 73-80 are ignored by the loader and are punched for identification purposes only.

3.2. Program to Duplicate Card Decks (DUPDECK)

This program duplicates the cards following the DUPDECK card up to the next control card. There are two option cards which are not punched and can appear anywhere in the deck.

$SEQUENCE	The following cards are sequenced in columns 76-80. Sequence numbers start with 00001 and are incremented by 1.
$NOSEQUENCE	No sequence numbers are provided. This is the initial option.

3.3. Program to List Card Decks (LISTER)

This program lists the cards following the LISTER card up to the next control card. There are three option cards which are not listed and can appear anywhere in the deck.

$PAGE	Start a new page with the next listed card.
$SEQUENCE	List subsequent cards with a card count, which starts at 1 and is incremented by 1. This is the initial option.
$NOSEQUENCE	List subsequent cards without card counts.

3.4. Tape Updating Utility Program (TUP)

TUP can be used to create, list, punch, or update a card-image tape file. Starting in the command mode, TUP reads and interprets a sequence of commands on cards, each of which is punched beginning in column one. If on any command card, except $LISTER or $PUNCHSEL, "LISTER" is punched in columns 12-17, the output produced during the interpretation of the command is also listed on the printer. Two cards, recognized in the command mode, are used to indicate the input and output units.

$INPUTn	Unit A is assigned the logical device number n. If n is omitted or if no $INPUTn is used, then unit A is logical device 6. Columns 10-17 specify the program name. If the name field is not blank, the input tape is searched for the program after the tape has been opened by a command. Column 19 specifies the rewind option (see section 3.6.1).
$OUTPUTn	Unit B is assigned the logical device number n. If n is omitted or if no $OUTPUTn is used, then unit B is logical device 7. Column 19 specifies the rewind option. Note that TUP produces an unlabeled source program as output (see section 3.6.1).

The appropriate $INPUTn and $OUTPUTn cards must precede the following command cards.

Note: 1) m and n as used below are five digit sequence numbers. m must be punched in columns 12-16, n in columns 20-24. Leading zeroes must be punched.

2) A TUP deck is ended by the first PL360 control card read. This card is processed as a $END card by TUP.

$NEWTAPE	Tape Unit B is opened. The subsequent card deck is read and put onto unit B. Every record is provided with a sequence number. Sequence numbers begin with 00010 and are incremented by 10. All 80 columns of the card can be used. A $END card completes the deck. Unit B is closed, and TUP returns to the command mode.
$LISTER m n	Unit A is opened. The records with sequence numbers m through n are listed. If n is omitted, listing is continued to the end of the program. If m is omitted, listing starts with the first record. Unit A is closed only if the program is listed to the end. After listing, TUP returns to the command mode.
$PUNCH	Unit A is opened. If a labeled program is to be punched, a program identification card is punched (see section 3.6.2). The entire program is punched, unit A is closed, a %EOF card is punched, and TUP returns to the command mode.
$PUNCHSEQ	This is identical to $PUNCH above, except that the tape sequence numbers are moved to columns 76-80 before the cards are punched.
$PUNCHSEL m n	Unit A is opened. The records with sequence numbers m through n are punched. If n is omitted, punching continues to the end of the deck. If m is omitted, punching starts with the first record. Unit A is closed only if the program is punched to the end. A %EOF card is punched and TUP returns to the command mode.

$RESEQUENCE

Unit A and unit B are both opened. The records on unit A are read, provided with new sequence numbers (starting with 00010 and incremented by 10), and written onto unit B. At the end of the program, unit A and unit B are closed, and TUP returns to the command mode.

$UPDATE

Unit A and unit B are both opened. TUP enters the update mode in which it updates the information on unit A with information read from cards. The updated information is written onto unit B. The following instructions are obeyed in the update mode:

$DELETE m n

Records with sequence numbers m through n are deleted. If n is missing, only one card is deleted.

$INSERT m

TUP enters the insert mode. The subsequent card records are inserted after the record with sequence number m. They are assigned sequence numbers beginning with m + 1 and incremented by 1. All cards (preceding the first $END update card) are treated as data to be inserted. If an inserted record is given a sequence number identical to that of an existing record on input unit A, that existing record is replaced by the new record. All 80 columns on data cards may be used.

$END

If TUP is in the insert mode, it returns to the update mode. Otherwise, the remainder of the program on unit A is copied onto unit B, units A and B are both closed, and TUP returns to the command mode.

$LISTER m	Listing starts (or resumes) at the record with sequence number m.
$NOLISTER m	Listing stops at the record with sequence number m. Note: Even when no listing is selected, all command or update mode cards and all inserted and replaced records are listed.
Other cards:	Any other card is treated as a data card, and it must be provided with a sequence number in columns 76-80 (leading zeroes must be punched). If its sequence number coincides with the sequence number of a record on the input tape, then that record is replaced by the one read from cards; otherwise, the card record is inserted at the appropriate place.
Note:	Cards in the update deck must be properly sequenced, i.e., the sequence numbers on "other cards" and the parameter m on update command cards must be in increasing sequence. If there are no cards in the update deck, then the program on unit A is simply copied onto unit B. If a card is read with a sequence number larger than any number in the program on unit A, unit A and unit B are closed and TUP returns to the command mode.

3.4.1. Opening and Closing Tape Units

Unit A is opened by performing the specified rewind option. If the program name field is not blank, the tape is searched from the starting point to the end of the tape for a program with that name. Unit A is closed by either rewinding or positioning at the next program on the tape, depending upon the rewind option (see $INPUTn and section 3.6.1).

Unit B is opened by performing the specified rewind option. It is closed by writing a program separator and then performing the specified rewind option (see $OUTPUTn and section 3.6.1).

3.4.2. TUP Error Messages

TUP always produces an output tape with sequence numbers in an increasing order. Each sequence number is checked before a record is written. If the sequence number is less than or equal to the last record written then the out-of-sequence record is not written. The record is listed on the printer with a message that it was deleted.

The m field of each update command card and the sequence field of each "other card" in an update run are checked for a valid 5 digit number. If the sequence number is not valid, then the card is ignored by TUP. All cards with invalid sequence numbers are listed with an indication that they were ignored.

3.5. System Tape Updating Program (SYSTUP)

SYSTUP can be used to list, copy, update, or punch the contents of system tapes. The program assumes the update mode unless a control card changes the mode. All mode control cards must occur before the first program identification card, $INSERT card or $RENAME card. The control cards can occur in any order because all system control cards are read before any action is taken.

3.5.1. Object Program Formats on Tape

Each program or data segment is represented on tape by at least two records. The first record is a header record describing the records comprising the given segment. These records immediately follow the header record. Normally, the program or data segment object code is written as a single record. However, in the OS system it is sometimes necessary to divide the segment to avoid exceeding the maximum record length allowable for the actual physical device being used (see section 5.2.2). This is done automatically by all the PL360 system programs using or producing object programs. The end of the object program is signalled by reading (or writing) a tape mark. Each program on a system tape is preceded by a header record with the format of a program identification card (see 3.5.2). This record gives an eight character name or label to the program. It is this name that is used by SYSTUP or by the system loader to recognize the program. On all "$" control cards referring to a labeled object program, the program name is punched in columns 10-17. Unlabeled object programs do not have a header record. The end of a system tape is indicated by the

occurrence of two successive tape marks. The first one indicates the end of the previous object program; the second indicates the end of the system tape.

At times it is desirable to control the rewinding of tapes containing system programs so that unnecessary tape positioning can be avoided when accessing more than one object program on a tape. Therefore, a rewind option exists on most "$" control cards. Column 19 is used to specify the option. "B" causes the tape to be rewound only before use; "A" causes the tape to be rewound only after use; "N" causes no rewinding; any other punch in column 19 (including blank) causes rewinding both before and after using the tape.

3.5.2. Program Identification Card

The copy, punch, and update modes of SYSTUP are controlled by program identification cards. Columns 2-9 of each card contain the program name by which each program is to be recognized by the system loader as well as by SYSTUP, columns 10-72 form a comment field to be used for version identification, and columns 73-80 constitute the version date field. If the date field is blank then the current date is copied into it. Column one is ignored on the card.

3.5.3. Mode Control Cards

$INPUTn	The input tape is identified with the logical device n (in decimal). Device 4 is the standard input unit if $INPUTn is not used or n is omitted. If n is zero, it is assumed that there is no input tape. Column 19 specifies the rewind option.

$OUTPUTn

The output tape is identified with logical device number n. Device 9 is the standard output unit if $OUTPUTn is not used or n is omitted. Column 19 specifies the rewind option.

$PUNCH

The initial input tape rewind option is performed. All the programs specified by program identification cards are punched. Each punched deck consists of an initial program identification card, the object deck, and finally a %EOF control card. Thus the deck is in the form used by SYSTUP to load a program. For each program identification card in the deck, the input tape is searched for the specified program. The search is started without rewinding and, if necessary, continued to the starting point after rewinding. Therefore, no specific order is required for program identification cards, but it is more efficient to punch decks in the same order as the programs appear on the tape. The first "%" control card read ends the punch run. The closing input tape rewind option is performed. $COPY or $LIST should not be used with $PUNCH.

$COPY

The initial input and output tape rewind options are performed. In the stand-alone system, a monitor is first put on the output tape as described in section 6.2.2. All the programs specified by program identification cards are copied from the input tape onto the output tape. Each program is located on the input tape in the same manner as described for $PUNCH. Since no specific order is required for program identification cards, a copy run can be used to reorder a system tape. The first "%" control card read ends the copy run.

A final tape mark is written on the output tape. The closing input and output tape rewind options are performed. $PUNCH or $LIST should not be used with $COPY.

$LIST — The initial input tape rewind option is performed; all the program identification records on the input tape are listed on the printer; then the closing input tape rewind option is performed. $PUNCH or $COPY should not be used with $LIST.

If a $PUNCH, $COPY, or $LIST card is not encountered among the mode control cards, then an update run is performed. In the stand-alone system, a monitor is written first on the tape followed by a tape mark (see section 6.2.2). In the OS system only a tape date record and a logical tape mark are written.

An update run is a sequential merge between the input tape and the changes specified by the card deck. Neither the input nor the output tape is rewound during the update. Therefore, the order of updating must correspond to the order of the programs on the input tape. The first "%" control card that is not %EOF or the first %EOF card that does not pair with a program identification card ends the input card deck. The update function operates as follows:

1. If the next card to be processed is $INSERT then one of the following occurs:

   a. If the end of the input tape has been read or if columns 10-17 are blank, then the deck set following the $INSERT card is immediately loaded onto the output tape without reference to the input tape. A deck set consists of a pro-

gram identification card followed by a deck indication (see section 3.5.4).

b. If columns 10-17 are not blank, then the input tape is copied onto the output tape up to and including the program named in columns 10-17. Then the following deck set is loaded as above.

2. If the next card to be processed is $RENAME then one of the following occurs:

a. If the end of the input tape has been read, then the card is ignored and a skip is made to the next "%" control card.

b. Otherwise, the input tape is copied onto the output tape up to the program named in columns 10-17. The next card in the deck replaces the old program identification record on the output tape, the program is copied from the input tape onto the output tape, and a skip is made to the next "%" control card.

3. If the next card to be processed is a program identification card then one of the following occurs:

a. If the end of the input tape has been read, the program is simply loaded onto the output tape.

b. Otherwise, the input tape is copied onto the output tape up to the program named on the program identification card. If a "%" control card follows the identification card, the named program is not placed on the output tape; otherwise a deck indication (see section 3.5.4) must follow and the new ver-

sion of the program is loaded onto the output tape. The input tape is moved forward to the next program.

4. If the end of the card deck has been reached, then the rest of the input tape is copied onto the output tape. An update run with no update deck copies all the input tape programs to the output tape.

Thus, the update mode has the following general features:

1. All programs on the input tape that are not specified in the update deck are sequentially copied to the output tape.
2. Programs can be inserted, changed, deleted, or renamed. The rename feature can change the program name, the version identification, or both.
3. The %EOF card separates each update step in the program, and each %EOF card signifies a return to the normal updating mode. Each update step is dependent on previous steps only because of order.
4. The end of both the card deck and the input tape must be reached before the update is completed. If the end of the input tape is reached while searching the input tape for a program name, then the current update step is completed as if the end of the input tape had been read before starting that step.

3.5.4. Deck Indications

There are three ways to specify the location of the object program associated with a given program identification card.

1. A $TAPEn card with columns 10-17 blank immediately follows. It indicates that the program is unlabeled and can be found on device n.

However, n cannot be the same as the input or the output device. Column 19 specifies the rewind option. The initial rewind option is performed, the program is loaded, and then the closing rewind option is performed. If the card following $TAPEn is not a "%" control card, then a card object deck must follow. It is also loaded and the combined program is written onto the output tape. If n is omitted, then device 5 is used.

2. A $TAPEn card with columns 10-17 not blank immediately follows. It indicates that the program is to be found labeled on another system tape on device n. However, n cannot be the same as the input or output device. Column 19 specifies the rewind option. The initial rewind option is performed and the tape is searched for the program named in columns 10-17. The search is the same as that described above for $PUNCH. That program is loaded and the closing rewind option is performed. If the card following $TAPEn is not a "%" control card, then a card object deck must follow. It is also loaded and the combined program is written onto the output tape. If n is omitted then device 5 is used. The program name on the $TAPEn card need not be the same as the one on the program identification card. Therefore, this feature can be used to rename a program.

3. Otherwise, a card object deck must immediately follow in the card reader. It is loaded until the next "%" control card (usually a %EOF card).

3.5.5. Examples of Usage

1. Copy the system tape from device 4 to device 9.

```
%SYSTUP
%EOF
```

2. List the program headers on the system tape on device 6.

```
%SYSTUP
$INPUT6
$LIST
%EOF
```

3. Punch object decks of PL360 and TUP from system tape on device 4.

```
%SYSTUP
$PUNCH
 PL360
 TUP
%EOF
```

4. Compile LISTER program and update the system tape from device 6 to device 8 with the new version of LISTER. Also insert an object deck of PNAME immediately after LISTER, and delete PNAME2 from the new system tape.

```
%PL360
$NOGO
                    {LISTER source deck}
%EOF
%SYSTUP
$INPUT6
$OUTPUT8
 LISTER
$TAPE
%EOF
$INSERT
 PNAME
                    {object deck}
%EOF
 PNAME2
%EOF
%EOF
```

Note: The final card of any SYSTUP deck may be a "%" control card for the next job in place of the final %EOF card shown in the examples above.

3.6. Source File Tape Updating Program (SFTUP)

SFTUP can be used to list, copy, or update the contents of source program file tapes. The mechanism is basically the same mechanism used by SYSTUP (see section 3.5). SFTUP does not have a punch option because TUP (see section 3.4) can be used whenever it is necessary to punch decks from a source tape. The program assumes the update mode unless a control card changes the mode. All mode control cards must occur before the first program identification card, $INSERT card, $RENAME card, $UPDATE card or $RESEQ card. The mode control cards can occur in any order because all mode cards are read before any action is taken.

3.6.1. Source Program Formats on Tape

Each source program is written on tape in a blocked format. Each tape record consists of eight source card images of eighty bytes each with an eight byte sequence field preceding each card image. The end of a source deck is indicated by a card image record with a special associated sequence field. Source programs are separated on tape by a special record. Source tapes for the stand-alone and OS systems are compatible. Each program on a source file tape is preceded by a header record with the format of a program identification card (see section 3.6.2). This record gives an eight character name or label to the program. It is this name that is used by SFTUP, PL360, TUP, XREF or SYNPROC to recognize the program. On all "$" control cards referring to a labeled source program, the program name is punched in columns 10-17. Unlabeled source programs do not have a header record. The end of a source file tape is indicated

by the occurrence of two successive special records. The first one indicates the end of the previous source program, the second indicates the end of the source file tape.

At times it is desirable to control the rewinding of tapes containing source programs so that unnecessary tape positioning can be avoided when accessing more than one source program on a tape. Therefore, a rewind option exists on most "$" control cards. Column 19 is used to specify the option. "B" causes the tape to be rewound only before use; "A" causes the tape to be rewound only after use; "N" causes no rewinding; any other punch in column 19 (including blank) causes rewinding both before and after using the tape.

3.6.2. Program Identification Card

The copy and update modes of SFTUP are controlled by program identification cards. Columns 2-9 of each card contain the program name by which each program is to be recognized by the various system programs; columns 10-72 form a comment field to be used for version identification; columns 73-80 constitute the version date field. If the date field is blank then the current date is copied into it. Column one is ignored on the card.

3.6.3. Mode Control Cards

$INPUTn	The input tape is identified with logical device number n. Device 6 is the standard input unit if $INPUTn is not used or n is omitted. If n is zero, it is assumed that there is no input tape. Column 19 specifies the rewind option.
$OUTPUTn	The output tape is identified with logical device number n. Device 7 is the standard output unit if $OUTPUTn is not used or n is omitted. Column 19 specifies the rewind option.
$COPY	The initial input and output tape rewind options are performed. All the programs specified by program identification cards are copied from the input tape onto the output tape. Each program is located by searching the input tape for the specified program. The search is started without rewinding and, if necessary, continued to the starting point after rewinding. Since no specific order is required for program identification cards, a copy run can be used to reorder a source file tape. The first "%" control card read ends the copy run. A final special record is written on the output tape. The closing input and output tape rewind options are performed. $LIST should not be used with $COPY.
$LIST	The initial input tape rewind option is performed; all the program identification records on the input tape are listed on the printer; then the closing input tape rewind option is performed. $COPY should not be used with $LIST.

If a $COPY or $LIST card is not encountered among the mode control cards, then an update run is performed. The initial input and output tape rewind options are performed. However, if the input unit is logical device zero, then no input tape is used, and the update run makes a new source file tape from the input card deck.

An update run is a sequential merge between the input tape and the changes specified by the card deck. Neither the input nor the output tape is rewound during the update. Therefore, the order of updating must correspond to the order of the programs on the input tape. The first "%" control card that is not %EOF or the first %EOF card that does not pair with a program identification card, $UPDATE card, or $RESEQ card ends the input card deck. The update function operates as follows:

1. If the next card to be processed is $UPDATE than one of the following occurs:
   a. If the end of the input tape has been read, the card is ignored and a skip is made to the next "%" control card.
   b. Otherwise, the input tape is copied onto the output tape up to the program named in columns 10-17. The deck following $UPDATE up to the next "%" control card is then used with the designated source program on the input tape to produce an updated source program on the output tape. The form of the update deck is the same as that used in TUP (see TUP update mode description in section 3.4).
2. If the next card to be processed is $RESEQ then one of the following occurs:

a. If the end of the input tape has been read, the card is ignored and a skip is made to the next "%" control card.

b. Otherwise, the input tape is copied onto the output tape up to the program named in columns 10-17. Each source record of the designated program is read from the input tape, provided with a new sequence number, and written onto the output tape. The sequence numbers start with 00010 and are incremented by 10.

3. If the next card to be processed is $INSERT then one of the following occurs:

a. If the end of the input tape has been read or if columns 10-17 are blank, the deck set following the $INSERT card is loaded onto the output tape without reference to the input tape. A deck set consists of a program identification card followed by a deck indication (see section 3.6.4).

b. If columns 10-17 are not blank, the input tape is copied onto the output tape up to and including the program named in columns 10-17. Then the following deck set is loaded as above.

4. If the next card to be processed is $RENAME then one of the following occurs:

a. If the end of the input tape has been read, the card is ignored and a skip is made to the next "%" control card.

b. Otherwise, the input tape is copied onto the output tape up to the program named in columns 10-17. The next card in the deck becomes the program identification record for the

source program, the source program is copied from the input tape onto the output tape, and a skip is made to the next "%" control card.

5. If the next card to be processed is a program identification card then one of the following occurs:
   a. If the end of the input tape has been read, the deck set is loaded onto the output tape.
   b. Otherwise, the input tape is copied onto the output tape up to the program named on the program identification card. If a "%" control card follows the program identification card, the named program is not placed on the output tape; otherwise, a deck indication (see section 3.6.4) must follow, and the new version of the source program is loaded onto the output tape. The input tape is moved forward to the next program.
6. If the end of the card deck has been reached, then the rest of the input tape is copied onto the output tape. An update run with no update deck copies all the input tape programs to the output tape.

Thus, the update mode has the following general features:

1. All programs on the input tape that are not specified in the update deck are sequentially copied to the output tape.
2. Programs can be inserted, completely changed, deleted, renamed, updated or resequenced. The rename feature can change the program name, the version identification, or both.
3. The %EOF card separates each file update step in the program, and

each %EOF card signifies a return to the normal updating mode. Each file update step is dependent on previous steps only because of order.

4. The end of the card deck and the input tape must be reached before the file update is completed. If the end of the input tape is reached while searching the input tape for a program name, then the current file update step is completed as if the end of the input tape had been read before starting that step.

3.6.4. Deck Indications

There are three ways to specify the location of the source program associated with a given program identification card.

1. A $TAPEn card with columns 10-17 blank immediately follows. It indicates that the program is unlabeled and can be found on device n. However, n cannot be the same as the input or output device. Column 19 specifies the rewind option. The initial rewind option is performed for device n, the source program is loaded onto the output tape, the closing rewind option is performed, and a skip is made to the next "%" control card. If n is omitted then device 5 is used. The source program is copied without changing sequence numbers.

2. A $TAPEn card immediately follows, with columns 10-17 not blank. It indicates that the program is to be found labeled on another source file tape on device n. However, n cannot be the same as the input or output device. Column 19 specifies the rewind option. The initial rewind option for device n is performed and the tape

is searched for the program named in columns 10-17. The search is the same as that described above for $COPY. That program is loaded onto the output tape, the closing rewind option for device n is performed, and a skip is made to the next "%" control card. The program name on the $TAPEn card need not be the same as the one on the program identification card. Therefore, this feature also can be used to rename a program. If n is omitted device 5 is used. The source program is copied without changing sequence numbers.

3. Otherwise, a source deck must immediately follow in the card reader. It is loaded onto the output tape until the next "%" control card. Each source record is given a sequence number (starting with 00010 and incremented by 10).

3.6.5. Examples of Usage

1. Copy the source file tape from device 6 to device 7.

```
%SFTUP
%EOF
```

2. List the program headers on the source tape on device 7.

```
%SFTUP
$INPUT7
$LIST
%EOF
```

3. Produce a new source file tape on device 7 from device 6. Update the source program TUP, resequence the source program XREF, insert the source program LISTER immediately after XREF and delete the source program PNAME. Assemble TUP and LISTER putting both object programs on device 5. Finally, make a new system tape from device 4 to device 9 with the new object versions of TUP and LISTER (inserted after XREF) and rename the object program PNAME to be PNAME2.

```
%SFTUP
$UPDATE       TUP
          {update deck}
%EOF
$RESEQ        XREF
%EOF
$INSERT
 LISTER
          {source deck}
%EOF
 PNAME
%EOF
%EOF
%PL360
$OUTPUT                   B
$TAPE         TUP         B
%EOF
%PL360
$OUTPUT                   A
$TAPE         LISTER      A
%EOF
%SYSTUP
 TUP
$TAPE                     B
```

```
%EOF
$INSERT        XREF
 LISTER
$TAPE                          A
%EOF
$RENAME        PNAME
 PNAME2
%EOF
%EOF
```

Note: The final card of any SFTUP or SYSTUP deck may be a "%" control card for the next job in place of the %EOF card shown in the examples above.

3.7. Syntax Processor (SYNPROC)

The syntax processor program can be used to process simple precedence grammars in order to determine the precedence matrix and the f and g functions as described by Wirth and Weber [3]. The main input to the processor consists of the productions of the language. A maximum of 499 productions containing 255 distinct symbols can be processed. Each production is punched on one card. Columns 1-72 of the card are used for the production and divided into six 12-character symbol fields. The left part symbol of the production occurs in columns 1-12. (If columns 1-12 are blank, then the left part of the previous production is used as the left part of the current production). The right part consists of 1 to 5 symbols punched in columns 13-24, 25-36, 37-48, 49-60, and 61-72 respectively. (Note that blank spaces are significant.)

As standard procedure, the syntax processor reads and lists all of the productions, constructs a symbol table in two parts (nonterminal and terminal), assigns each symbol a number, and finally determines the precedence matrix if it exists or prints out the conflicts that make the matrix not exist.

The following option cards are recognized by SYNPROC:

$SYMBOLS	The symbol table is to be read in before reading the productions. Each symbol must be on a separate card in columns 1-12. The nonterminal symbols are read first. A "$$" card signifies the end of the nonterminals and the start of the terminals. A second "$$" card is used to separate the terminal symbols from the productions. Every symbol in the language must occur on a card. The terminal and nonterminal symbol groups can be ordered in any desired fashion. In this way the user can specify his own symbol numbers.

$SYMPUNCH	This causes the symbol table to be punched in the form used by SYNPROC (including the "$SYMBOLS" and two "$$" cards).
$CHECK	After the symbol table has been listed, a check is made for any productions having identical right parts. All such occurrences are listed. No check is made if the card is omitted.
$MATRIX	If the precedence matrix exists then it is printed out in blocks 100 symbols wide. If the grammar has more than 99 symbols, then the matrix will be printed in two parts or in three parts if there are more than 199 symbols.
$LEFT	In general it is tedious to look up relations using the precedence matrix. If the precedence matrix exists, the $LEFT will cause each symbol to be listed along with the relation and symbol of all symbols that have a relation to the right of the symbol. Five relations are printed per line in order to condense output.
$RIGHT	This is entirely analogous to $LEFT.
$FUNCTIONS	If the precedence matrix exists, then the f and g precedence functions are calculated. If they exist then they are listed; otherwise the precedence chain that makes them not exist is printed.
$TAPEn	Subsequent source records are read from the tape unit with logical number n. If n is omitted, device 7 is used. Columns 10-17 of the card specify the program name. If the name field is not blank, the tape is searched for that program from the starting point to the end. Column 19 specifies the rewind option (see section 3.6.1). $TAPEn is the last card read by SYNPROC from the card deck.

\$OUTPUTn — The results of the syntax processor are put on device n. If n is omitted device 5 is used. Column 19 specifies the rewind option. If the matrix does not exist then only a closing tape mark is written. If the functions are requested then they are written on tape if they exist. The output consists of an 80-character control record followed in order by the symbol table, production table, matrix, and functions (if calculated), described as follows:

1. Control record

Cols 1- 4	number of nonterminal symbols
Cols 5- 8	total number of symbols, M
Cols 9-12	total number of productions, N
Cols 13-16	length of symbol table in bytes, 12(M + 1)
Cols 17-20	length of production table in bytes, 12(N + 1)
Cols 21-24	length of matrix in bytes, 64(M + 1)
Cols 25-28	length of functions in bytes, 2(M + 1)

2. Symbol table

The symbol table consists of twelve byte entries which are the nonterminal and terminal alphabetic symbols ordered by number. Symbol 0 is the blank symbol; thus the symbol table has M + 1 elements and is a 12(M + 1) byte record.

3. Production table

Each production is represented by six short integers that contain the symbol number for each part of the rule. All symbol numbers are doubled in the table in order to facilitate half-word indexing in function calculations. The symbol number 0 fills out all right

parts that consisted of less than five symbols. Since each rule takes 12 bytes and the first entry is not used, there are 12(N + 1) bytes in the record. The processor can handle a maximum of 499 productions.

4. Matrix

The matrix is written in a completely packed form with 2 bits used for each relation (00 for no relation, 01 for $\lessdot$ relation, 10 for $\gtrdot$ relation, 11 for $\doteq$ relation). The processor can handle a maximum of 255 symbols so there are 64 bytes for each row. The first row corresponds to symbol 0 so there are 64(M + 1) bytes in the record.

5. Functions

The f and g functions are respectively the last two records on the tape. The function values are short integers so each record is 2(M + 1) bytes long, including a functional value of 0 for symbol 0.

The control cards can appear in any order and at any place in the deck, except that $SYMBOLS and the symbol cards must obviously be placed before the first production.

3.8. Program to Generate Cross-Reference Tables of Identifiers (XREF)

The cross-reference program will list alphabetically all identifiers in a source program with the numbers of the lines on which they occur. An identifier is defined as a string of one or more letters and digits, with the first character a letter. According to various input options, one may request that certain identifiers not be referenced (e.g. reserved words) or that only specified identifiers be cross-referenced. The input program may be on cards or tape, and a listing of it may be suppressed.

3.8.1. Control Cards

1. %XREF

2. $PL360 — Ignore all PL360 basic tokens, standard identifiers and identifiers on card(s) 3, cross-referencing all others.

   or

   $IGNORE — Ignore the identifiers on card(s) 3, monitoring all others.

   or

   $MONITOR — Cross-reference only the identifiers on card(s) 3, ignoring all others.

   ($IGNORE is assumed if card 2 is omitted.)

3. Specify a list of identifiers in free field, taking as many cards as necessary.

   (If omitted, the list is assumed to be empty.)

4. $NOLIST — Suppress the listing of the input program.

or

$LIST — List the input program.

($LIST is assumed if 4 is omitted. These cards may occur anywhere within the input deck to obtain appropriate listing action.)

5. $CARD — The input program is on cards. Each card is provided with a sequence number (starting with 00010 and incremented by 10).

or

$TAPEn — The input program is on tape n. If n is omitted device 7 is used. Columns 10-17 of the card specify the program name. If the name field is not blank, the tape is searched for that program from the starting point to the end. Column 19 specifies the rewind option (see section 3.6.1). $TAPEn is the last card read by XREF. The sequence number associated with each tape record is used by XREF.

(Card 5 must not be omitted.)

3.8.2. Program Size Limitations

500	unique identifiers
3000	total characters of identifiers
8000	total references to the identifiers
200	unique special identifiers to be monitored or ignored
1000	total characters of special identifiers

3.8.3. Sample Deck Setups

1. Cross-reference all identifiers. List the card program.

```
%XREF
$CARD
      {input deck}
%EOF
```

2. Cross-reference all but the standard PL360 identifiers. List the card program.

```
%XREF
$PL360
$CARD
      {input deck}
%EOF
```

3. Cross-reference all but the standard PL360 identifiers and the identifiers ALFA BETA GAMMA. The unlabeled program is on device 6. Do not list the program.

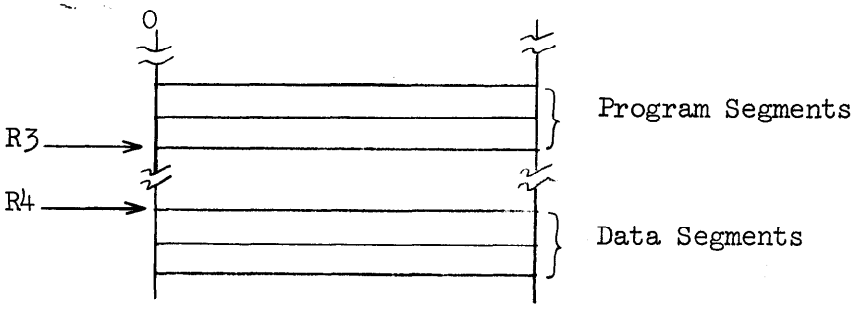
```
%XREF
$PL360
 ALFA BETA GAMMA
$NOLIST
$TAPE6
```

4. The PL360 Environment

This chapter presents some details of the provided environment that are of importance to the PL360 programmer. In particular, storage organization and the reaction of the system to unintended situations are described. Further information about the input/output facilities provided by the two systems is included in Chapters 5 and 6.

4.1. Storage Organization

The storage available to PL360 programs is organized as indicated in the following diagram.



Before each program is loaded, the entire storage area is set to zero. During the loading process, the segment reference table (see [1], Chapter 5) is completed. This table consists of 80 entries, 64 of which are available for user program and data segments, and is either the first entry in data segment 0 (OS system) or included in the monitor area (stand-alone system). Each program is then treated as a procedure and called with the following information in registers:

R2	return address
R3	address of the first byte following the "last" program segment
R4	address of the first byte of the "first" data segment

R6 — record length parameter (OS system only, see 5.2.1) or 32767 (stand-alone system).

PL360 programs are compiled as if they were declared as segment procedures. Additionally, the return address is saved upon entry and restored upon exit.

4.2. Program Execution (Run-Time) Errors

PL360 program execution is terminated by the detection of input/output errors, by expiration of a specified time interval, by external intervention (stand-alone system only), and conditionally by the detection of program interrupts. System action in the last case is determined by interrogation of an extended program check mask. If the bit of that mask selected by the interrupt code (see [4]) is a one, the program is terminated; otherwise, the condition code is set to 3 (overflow) and execution continues. Part of the mask may be set by the standard function SETPMASK.

SETPMASK — Use the low-order byte of register R0 to set the extended program check mask according to the following table.

Bit	Interrupt Condition
24	fixed-point overflow
25	fixed-point divide
26	decimal overflow
27	decimal divide
28	exponent overflow
29	exponent underflow
30	significance
31	floating-point divide

Mask bits for all other program interrupts are always set to one. Counts of the maskable interrupts listed above are maintained and printed (if non-zero) after termination of program execution.

Every abnormal program termination results in the printing of an error message and a dump of the data area of the interrupted program. The standard procedure SETDUMP provides for a selective dump.

SETDUMP	Set the beginning of the area to be dumped upon abnormal termination to the address contained in register R0; set the length of that area (in bytes) to the value of register R1. Register R2 and the condition code are altered.

The message provides an indication of the nature of the error. The PSW at the point of detection of that error is printed, and, if appropriate, the address is also given as a (decimal) segment number and (hexadecimal) displacement. For errors related to input/output, the PSW displayed by the OS system is a pseudo-PSW in which the high-order word contains the logical device number. The format of these messages may differ somewhat between the two systems. The following conditions are detected.

1. PRG CHK
   A program check interrupt was detected, and the corresponding mask bit was a one.
2. END DS or END CRD
   An attempt was made to read beyond the next PL360 system job card on logical device 2 or (OS system only) beyond the end of the data set on a logical tape unit.

3. DEV NOP

An input/output device was addressed which either does not exist or was not operable. In the OS system, a device is considered not operable if the associated data set cannot be successfully opened (usually because of a missing DD card). See section 5.3.

4. I/O ERR

An input/output error occurred after initiation of the corresponding channel program. Channel status and device sense information, which must be interpreted according to the particular physical device, is also provided with this message. In the OS system, if this message references logical device 5 and that device is implemented in main storage (see section 5.2.1), it indicates that insufficient storage was available, and the status and sense information is not relevant.

5. I/O CHK

An illegal input/output operation, such as one specifying an illegal address or record length, was attempted. Channel status information may be provided.

6. XS TIME or EXT INT

The job was terminated by expiration of the specified time interval or (stand-alone system only) by operator intervention.

After program termination due to an error, an attempt will be made to continue processing with the next PL360 system job. In the OS system, however, certain serious input/output errors will not be accepted by OS but will instead cause termination of processing of the PL360 batch. In addition, PL360 programs use the same memory protection key as the PL360/OS subsystem monitor and thus can cause its failure.

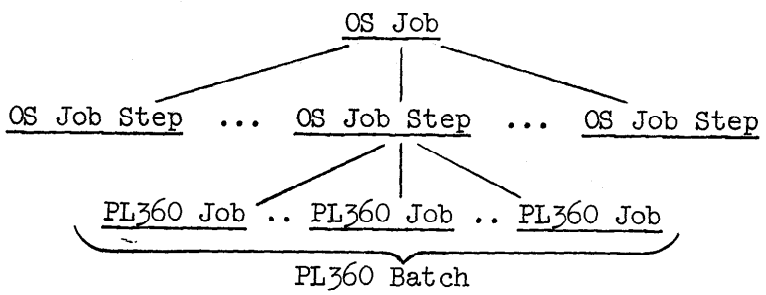
5. Use of the PL360/OS System

This chapter describes the use of the PL360 system as a subsystem of IBM's Operating System/360 (OS). Section 5.1 contains a very brief summary of some concepts of OS job and data management important for programming in PL360 under OS. Section 5.2 provides further information about the use of previously described PL360 input/output functions under OS and also summarizes points of incompatibility between the OS and stand-alone systems. Section 5.3 describes the OS deck setup and Job Control Language statements required for the use of the system. Finally, section 5.4 presents some sample deck setups for typical tasks and system configurations. In general, the OS control statements shown in these examples will need some modification to reflect the conventions for naming device classes, storage volumes, and data sets used by a particular installation. Thus normally some familiarity with OS concepts and terminology will be required and in parts of Section 5.3 is assumed. The publication IBM System/360 Operating System Concepts and Facilities [5] contains an introduction to OS and further references.

5.1. Background and General Organization

Both OS/360 and the PL360/OS subsystem process sequences of jobs under the direction of control statements. The relationship between them, as well as the terminology to be used in the remainder of this chapter, is summarized as follows: An OS job, which is the OS unit of accounting and scope of certain names, consists of one or more OS job steps, each delimited by OS Job Control Language (JCL) statements as described in section 5.3. The PL360/OS system is processed as a

single OS job step and, in turn, processes a PL360 batch, which consists of one or more PL360 system jobs, each delimited by PL360 system control cards as described in Chapter 2. Thus a sequence of PL360 system jobs can be processed in a single OS job step. These relations are shown schematically in the following diagram:



PL360/OS is a closed subsystem; output from its language processors cannot be used as input to the OS linkage editor, nor can the output of IBM language processors be loaded by the PL360/OS system.

Input and output for the PL360/OS system are directed to OS data sets (i.e., delimited collections of associated records) rather than to physical input/output devices. These data sets are referenced in PL360 programs by logical device numbers. The association of data sets and logical devices is determined by OS Job Control Language Data Definition (DD) statements for each PL360 batch. The association is fixed for each batch but may vary between batches. The PL360/OS system provides for two data set organizations. In the first, logical records have a fixed length and must be processed sequentially. These data sets correspond to physical unit record equipment and for compatibility are referenced by the unit record input/output functions of the stand-alone system (READ, WRITE, PAGE, PUNCH). The second type of data set

consists of arbitrary length records. This type of data set must be sequentially organized but may be accessed in a non-sequential manner by the use of certain marking and positioning functions. Such data sets correspond to physical tape units and for compatibility are referenced by the tape input/output functions of the stand-alone system.

The standard PL360/OS system supports nine logical devices. These devices and associated logical characteristics are as follows:

Device Number	Logical Device Type
1	line printer
2	card reader
3	card punch
4	tape (system)
5	tape (scratch)
6	tape
7	tape
8	tape
9	tape

5.2. PL360 System Programs Under OS

5.2.1. Input/Output Considerations

Supervisor input/output function statements are written in PL360 as described in section 3.1.2.4; however, the following restrictions are made:

1. READ

   Attempting to read past the end of the data set associated with the reader (logical device 2) will cause immediate termination of the PL360 system. The message

   ```
   * END OF READER  DS
   ```

will be printed at the top of the next (logical) page. In many cases, the reader data set will be placed in an OS system input stream (i.e., an input stream from which OS Job Control Language statements are being read and interpreted). In such cases, any logical record with a "/*" in columns 1 and 2 will be interpreted by OS to mean that the preceding record terminated the data set.

2. Tape Functions

a. Logical tape marks are written by the PL360/OS system as special records. Such records are 18 bytes long; the first 14 bytes are #E0 (corresponding to 0-2-8 punches). Since such records are recognized as special marks by the system, they should not be written using the WRITETAPE function.

b. The functions FSPREC, BSPREC, READTYPE, and WRITETYPE are not supported by the PL360/OS system. The corresponding identifiers are not predeclared in the OS PL360 compiler.

c. OS requires that the length of each record in a data set be in the range of 18 bytes to 32760 bytes inclusive. In addition, for data sets on direct access devices, record lengths cannot exceed the track capacity of the device unless the track overflow feature is available and specified (see section 5.3.5). These track capacities are as follows:

Physical Device	Track Capacity (bytes)
2311 disk	3625
2314 disk	7294
2302 disk	4984
2301 drum	20483
2303 drum	4892
2321 data cell	2000

Attempts to write records longer than those allowed for the physical device will cause termination of the OS job step with

an OS-supplied error message. For each PL360 batch, a record length parameter may optionally be specified (see section 5.3.4); its default value is 3520. This parameter is passed to each program in that batch in register 6. In addition, it is used by the PL360 compiler and system tape update programs as described in section 5.2.2.

d. For installations requiring rapid processing of PL360 jobs and with large amounts of core storage available, an option has been provided to use a core storage area for logical device 5. The size of this area is adjusted to the total amount of storage available and in the standard system will lie in the range of 32K bytes to 128K bytes (K=1024). Track size considerations do not apply to device 5 when so implemented. This option is selected by an OS job step parameter (see section 5.3.4); if it is not specifically selected, transfer to or from an OS data set is assumed.

5.2.2. System Program Considerations

In addition to the input/output considerations above, the following points concerning the use of the system programs in the PL360/OS system should be noted.

5.2.2.1. The PL360 Compiler

Some programs which can be compiled by the stand-alone system will cause segment overflow errors in the OS version, since the first 368 bytes of data segment 0 are unavailable for data and since supervisor function statements generate twelve bytes of code instead of two. Compiled segments will be written on the scratch data set as multiple records, if necessary, to limit maximum record length to that specified by the record length parameter.

5.2.2.2. The System Tape Updating Program

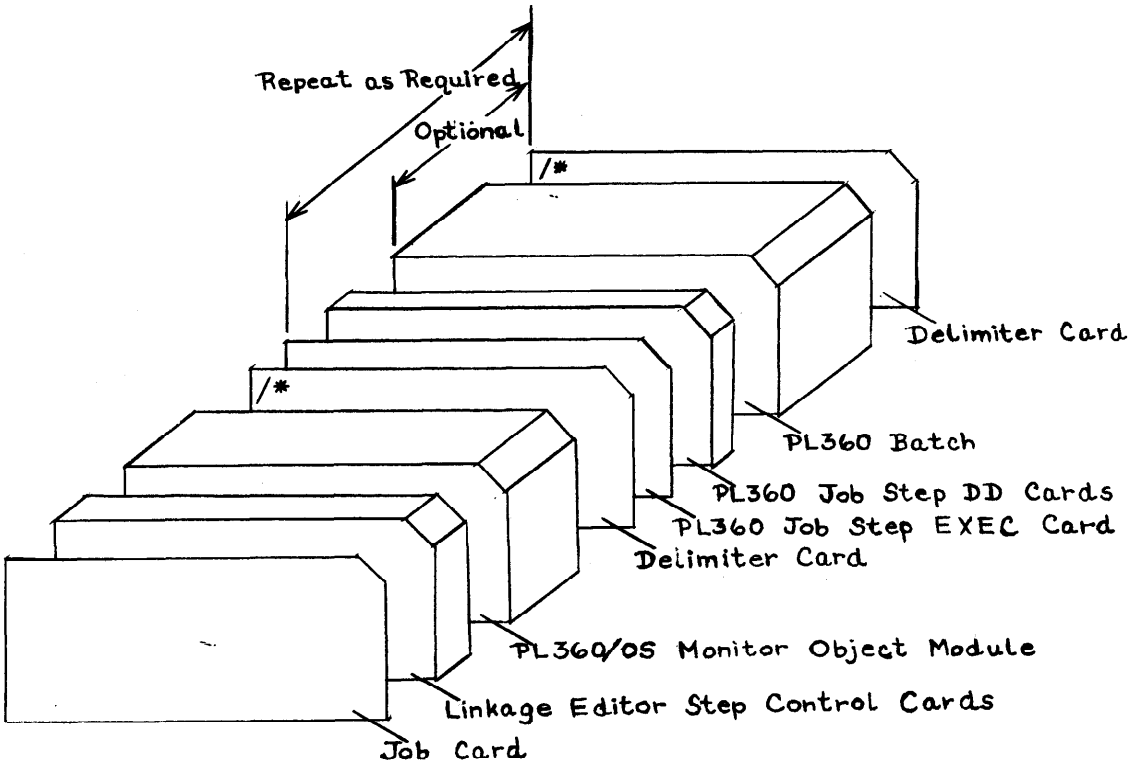
Maximum record length on any new system tape generated will not exceed that specified by the record length parameter.

5.3. OS Job Control Language Requirements

In parts of this section, the reader is assumed to be familiar with OS concepts and terminology and is referred to the publication IBM System/360 Operating System Job Control Language [6] for further information and explanation. Use of the system with a card object module and complete set of Job Control Language (JCL) statements will be described. The number of non-PL360 system cards required is greatly reduced if the installation provides a load module of the monitor in a private or system library and provides an appropriate catalogued procedure. Documentation of the use of such facilities is considered an installation responsibility.

5.3.1. OS Job Organization and Deck Setup

A PL360/OS job consists of two or more OS job steps. In the first step, the OS linkage editor is used to produce an executable load module of the monitor. In each subsequent step, a batch of PL360 jobs is processed. The card deck organization required is shown schematically below.



Note: A delimiter card contains a "/*" in columns 1 and 2.

The OS JCL statements required are described below. Information concerning the syntax and format of cards containing these statements may be found in the IBM JCL manual [6]. The PL360 batch must contain PL360 system control cards as described in Chapter 2. In particular, it should be noted that the OS delimiter card is not a substitute for a PL360 %EOF card (see section 5.2.1).

5.3.2. The Job Card

This card is prepared according to individual installation standards.

5.3.3. The Linkage Editor Job Step Control Cards

If available, an installation catalogued procedure may be used for the linkage editing step. Otherwise, the JCL statements for this step should be copied from the installation's standard catalogued procedure ASMFCLG, with any DD statements for SYSLIB omitted and with SYSLIN naming the following object module cards. A typical set of statements follows.

```
//LKED  EXEC     PGM=IEWL
//SYSUT1   DD     SPACE=(3076,(5,5)),UNIT=SYSDA
//SYSPRINT DD     SYSOUT=A
//SYSLMOD  DD     DSNAME=&GOSET(GO),UNIT=SYSDA,SPACE=(3076,(2,1,1)),   X
//               DCB=(RECFM=U,BLKSIZE=3076),DISP=(NEW,PASS)
//SYSLIN   DD     *
```

5.3.4. The EXEC Statement

Execution of the load module containing the monitor and produced in a linkage editing step named LKED is specified by a statement of one of the following forms:

```
//stepname  EXEC     PGM=*.LKED.SYSLMOD
```

or

```
//stepname  EXEC     PGM=*.LKED.SYSLMOD,PARM=parameter
```

The name of each step within a job should be unique. The parameter is of the form given by

parameter ::= C | unsigned-integer | C unsigned-integer

The C specifies that logical device 5 is to be implemented in main core storage; the unsigned integer specifies the record length parameter (see section 5.2.1).

5.3.5. The DD Statements

DD statements serve to associate PL360 system logical device numbers with OS data sets. The DD statement for logical device n is named DEVICEn, and its operand field describes the corresponding data set. All data sets processed by the PL360/OS system must be sequentially organized. The table below summarizes the logical characteristics required of the data sets to be associated with each device.

Device	Type of Access	Record Format	OS Format Code	Logical Record Length (bytes)
1	output	fixed	FBA	133 *
2	input	fixed	FB	80
3	output	fixed	FB	80
4	input	undefined	U	variable
5	input/output	undefined	U	variable
6	input/output	undefined	U	variable
7	input/output	undefined	U	variable
8	input/output	undefined	U	variable
9	input/output	undefined	U	variable

* Includes a USASI carriage control character supplied by the monitor.

Detailed information about DD statements will be found in the IBM JCL manual [6]; section 2 of that manual contains model DD statements for many common applications. In general, each DD statement contains sufficient information to name, locate, and indicate the status of each data set. In addition, by use of the DCB operand it is possible to supply information that describes the organization of the data set and is used to complete internal control blocks. The following DCB informa-

tion is required for PL360 data sets:

1. For devices 1, 2, and 3, appropriate values for the physical block size (BLKSIZE) and number of buffers (BUFNO) must be specified.

2. For devices 4 through 9, the record format (RECFM) must be specified as UT if the track overflow feature is to be used for direct access data sets. Otherwise, the record format is assumed to be U (which may be explicitly specified).

In addition, DCB information specifying device-dependent options (recording density, stacker selection, etc.) may also be supplied.

DD statements for devices 1, 2, and 4 are always required. A DD statement for device 5 is required unless device 5 is implemented in main storage (see section 5.2.1). DD statements for device 3 and for devices 6 through 9 are optional. If a device in the latter set is referenced and the corresponding DD statement is missing, the PL360 job will be terminated with an I/O error message.

5.4. Examples

In this section, examples are given of both individual DD statements and appropriate sets of JCL statements for PL360 job steps in some typical situations.

5.4.1. DD Statement Examples

5.4.1.1. Fixed Format Data Sets

1. `//DEVICE1 DD SYSOUT=A,DCB=(BLKSIZE=133,BUFNO=1)`

Output records are directed to the system printer output stream.

2. //DEVICE2 DD *

The data set follows. This must be the last DD statement for the job step, and the reader data set must immediately follow in the system input stream. Appropriate DCB information is supplied by the system.

3. //DEVICE2 DD DDNAME=SYSIN

The data set is identified with the data set specified by the DD statement named SYSIN. This form is primarily used in catalogued procedures.

4. //DEVICE3 DD UNIT=SYSCP,DCB=(BLKSIZE=80,BUFNO=2)

Output records are punched on a system card punch.

5. //DEVICE3 DD DSNAME=A000.CARDS,UNIT=2314,VOLUME=SER=SYS07, X
// DCB=(BLKSIZE=1600,BUFNO=1),SPACE=(1600,(20,10)), X
// DISP=(NEW,KEEP)

Card-image output records are blocked and placed in a newly created direct access data set for later processing, such as editing.

6. //DEVICE2 DD DSNAME=A000.CARDS,UNIT=2314,VOLUME=SER=SYS07, X
// DCB=(BLKSIZE=1600,BUFNO=1),DISP=(OLD,KEEP)

A previously created card-image data set on a direct access volume is processed as reader input. Note that such a data set must contain PL360 control cards but no OS control statements.

5.4.1.2. Undefined Format Data Sets

Tapes

1. //DEVICE4 DD DSNAME=DUMMY1,UNIT=TAPE9,VOLUME=SER=SS0050, X
// LABEL=(,NL),DISP=(OLD,KEEP)

The system library is identified as a data set on unlabeled tape. A dummy data set name is required because a disposition of KEEP is specified.

2. `//DEVICE4 DD  DSNAME=A000.TAPESYS,UNIT=TAPE9,VOLUME=SER=SG0001, X`
`//  DISP=(OLD,KEEP)`

The system library is identified as a data set on labeled tape.

3. `//DEVICE5 DD  UNIT=TAPE9,LABEL=(,NL)`

An unlabeled scratch tape is specified.

4. `//DEVICE7 DD  UNIT=(TAPE9,,DEFER),LABEL=(,NL)`

As above, but the tape need not be mounted until (and unless) needed.

5. `//DEVICE9 DD  DSNAME=A000.PL360SYS,UNIT=282,VOLUME=SER=GSG140, X`
`//  DISP=(NEW,KEEP)`

A new data set is to be created on a labeled tape mounted on unit 282.

Direct Access Devices

6. `//DEVICE4 DD  DSNAME=A000.PL360SYS,UNIT=2311,VOLUME=SER=SLACA0, X`
`//  DISP=(OLD,KEEP)`

The system library is identified as a 2311 data set named A000.PL360SYS.

7. `//DEVICE4 DD  DSNAME=A000.PL360SYS,UNIT=2314,VOLUME=SER=SYS01, X`
`//  DCB=(RECFM=UT),DISP=(OLD,KEEP)`

The system library is identified as a 2314 data set written using the track overflow feature.

8. `//DEVICE5 DD  UNIT=SYSDA,SPACE=(3500,(10,5))`

A scratch data set on a direct access volume is specified.

9. `//DEVICE5 DD  SPACE=(TRK,(10,5)),DCB=(RECFM=UT), X`
`//  VOLUME=REF=SYS1.UT2`

A scratch data set on a system utility volume is specified.

10. `//DEVICE6 DD  DSNAME=A000.SOURCE,DCB=(RECFM=UT),DISP=(OLD,KEEP)`

An existing catalogued data set is specified.

11. `//DEVICE7 DD  DSNAME=A000.SFILE,UNIT=2321,VOLUME=SER=CAMP38,  X`
`//  SPACE=(CYL,(50,20),RLSE),DISP=(NEW,KEEP)`

A new data set is created on a 2321 volume. Any allocated but unused space is to be released at the end of the job step.

12. `//DEVICE9 DD  DSNAME=A000.SYSTEM.MOD,UNIT=2314,VOLUME=SER=PUB001,X`
`//  DCB=(RECFM=UT),SPACE=(CYL,(2,1)),DISP=(NEW,KEEP)`

A new data set is created on the 2314 volume PUB001.

Other Devices

13. `//DEVICE8 DD  UNIT=SYSCP,DCB=(RECFM=U,MODE=C)`

Output is directed to the system card punch, operating in column binary mode.

5.4.2. Sample Job Steps

The following examples illustrate sample JCL and, when appropriate, PL360 deck setups for OS job steps. The linkage editing step is not included but is assumed to be named LKED.

1.
```
//PL360   EXEC PGM=*.LKED.SYSLMOD,PARM=3625
//DEVICE1 DD   SYSOUT=A,DCB=(BLKSIZE=133,BUFNO=1)
//DEVICE4 DD   DSNAME=DUMMY,UNIT=283,VOLUME=SER=GSG140,          X
//            LABEL=(,NL),DISP=(OLD,KEEP)
//DEVICE5 DD   UNIT=2311,SPACE=(TRK,(10,5))
//DEVICE2 DD   *
 {PL360 Job Batch}
/*
```

This example illustrates typical minimal JCL for a smaller machine installation. The system is on an unlabeled tape volume named GSG140; the scratch area is on a 2311.

2.
```
//PL360   EXEC PGM=*.LKED.SYSLMOD,PARM=C32760
//DEVICE1 DD   SYSOUT=A,DCB=(BLKSIZE=133,BUFNO=1)
//DEVICE4 DD   DSNAME=A000.PL360SYS,UNIT=2314,VOLUME=SER=PUB001,  X
//            DCB=(RECFM=UT),DISP=(OLD,KEEP)
//DEVICE2 DD   *
 {PL360 Job Batch}
/*
```

This example illustrates typical minimal JCL for a larger machine installation. The system is on the disk volume PUB001; main storage is used as the scratch area.

3.
```
//STEP1   EXEC PGM=*.LKED.SYSLMOD,PARM=32760
//DEVICE1 DD   SYSOUT=A,DCB=(BLKSIZE=133,BUFNO=1)
//DEVICE4 DD   DSNAME=A000.PL360SYS,UNIT=2314,VOLUME=SER=PUB001,     X
//            DCB=(RECFM=UT),DISP=(OLD,KEEP)
//DEVICE5 DD   DSNAME=SYS1.UT2,DCB=(RECFM=UT),DISP=(OLD,KEEP)
//DEVICE6 DD   DSNAME=A000.SFILE,UNIT=2321,VOLUME=SER=CAMP38,        X
//            DISP=(OLD,KEEP)
//DEVICE7 DD   DSNAME=SYS1.UT3,DCB=(RECFM=UT),DISP=(OLD,KEEP)
//DEVICE9 DD   DSNAME=A000.TEMPSYS,UNIT=2314,VOLUME=SER=PUB002,      X
//            DCB=(RECFM=UT),SPACE=(CYL,(2,1),RLSE),DISP=(NEW,PASS)
//DEVICE2 DD   *
 {Update Decks}
%SYSTUP
 {Update Specifications}
%EOF
/*
//STEP2   EXEC PGM=*.LKED.SYSLMOD,PARM=C32760
//DEVICE1 DD   SYSOUT=A,DCB=(BLKSIZE=133,BUFNO=1)
//DEVICE4 DD   DSNAME=*.STEP1.DEVICE9,DCB=(RECFM=UT),                X
//            DISP=(OLD,KEEP)
//DEVICE2 DD   *
 {Test Programs}
/*
```

See section 3.6.5, example 3, for a possible update deck.

In this example, a system program source file is updated and used to create a new copy of the system on device 9. The new system is passed to a second job step as device 4, which tests the update and saves it. Previously allocated and catalogued scratch data sets are used for devices 5 and 7.

6. Use of the PL360 Stand-Alone System

This chapter describes the use of the PL360 system as a completely independent, self-loading program. Section 6.1 contains information about the input/output facilities provided. Section 6.2 describes the PAUSE control card and system program facilities unique to the stand-alone system. Finally, section 6.3 contains information about loading and operating the system and serves as an operator's guide.

6.1. Input/Output Facilities

Input and output for the stand-alone system are directed to an appropriate physical device. These devices are designated by logical device numbers. In the standard system, ten logical devices are provided, with the following characteristics.

Logical Device Number	Device Type
0	operator's console
1	printer
2	card reader
3	card punch
4	system tape
5	tape
6	tape
7	tape
8	tape
9	tape

The correspondence between logical device numbers and actual device addresses is established by a device table, which may be examined and altered from the operator's console (see section 6.3).

6.2. Special System Facilities

6.2.1. The PAUSE Card

The following additional control card (see Chapter 2) is provided:

PAUSE — Columns 1 through 50 of the card are typed on the operator's console as a message to the operator, and the system waits for instructions to be entered at the console (see section 6.3).

6.2.2. The System Tape Update Program (SYSTUP)

In the stand-alone system, a self-loading monitor is included on the system tape. In either an update or copy run of the program SYSTUP, it is necessary to first put a copy of the monitor on the new tape. Normally the monitor can be copied from the input tape, and that is the default option. However, two mode control cards are provided to specify alternate sources, as follows:

a. $LOAD signifies that the monitor is to be loaded from the cards following in the card reader. The object deck is assumed to be an absolute 360 assembly language object deck. The transfer address on the END card must specify the initial program status word. The length of the monitor is determined from the ESD card and is aligned to a half-segment address. The first PL360 control card signifies the end of the monitor deck.

b. $MONITOR signifies that the monitor is to be copied from low core. The length of the monitor must be the same as the one on the input tape (aligned to a half-segment address). The address at which execution will start must be in the half-word starting at memory location 20 (decimal). This feature is intended mainly to facilitate the creation of system tapes with different device assignments.

After these options have been processed, a normal copy or update is performed.

6.3. Initial Loading and Operating the System

Initial loading of the PL360 system is accomplished by the following procedure:

a. Reset the system.

b. Mount the system tape on an appropriate unit.

c. Stack jobs, including the control cards of Chapter 2, on the card reader.

d. Make the card reader, line printer, and logical tape 5 (scratch tape) ready.

e. Select the unit carrying the system tape on the load unit rotary switches.

f. Press the load key.

g. Enter the date (8 characters) from the console.

Operation of the system requires that the device addresses for the operator's console, card reader, and line printer be correctly loaded. If any of these addresses must be changed, the following additional steps should be executed when the processor enters the wait state following step f. They use the storage select switch and address keys, the display and store keys, and the storage data switches according to the detailed procedures for each processor model.

f1. Press the stop key.

f2. Examine the half-word at hexadecimal location 16 (decimal 22) to determine the device table address.

f3. Modify the half-word device address entries in the device table, as follows:

Device	Displacement in Device Table (hexadecimal)	Standard Device Address (hexadecimal)
console	0	0009
line printer	2	000E
card reader	4	000C

f4. Press the start key.

f5. If the console address required modification, press the attention key on the console.

Addresses of other devices may be changed from the console if an initial PAUSE card is included in the job sequence.

After initial loading, execution of the job sequence stacked on the card reader is immediately started. Pressing the external interrupt key causes the job currently being executed to be terminated and the next job in sequence to be started. Control is returned to the operator when a PAUSE control card is encountered. The computer then accepts instructions from the operator via the console. Each message must be terminated with an EOB (end of block) character. The following free-field instructions are accepted:

a. dump XXXXXX, NNNNNN EOB

dump XXXXXX EOB

dump EOB

The values of the registers and of the NNNNNN byte cells starting from the initial address XXXXXX are listed on the line printer in hexadecimal form. If the initial address is omitted, it is taken as the end of the user's program segment area, and if the count is omitted, the dump extends over the entire data segment area (see section 4.1).

b. device XX EOB

The address AAA of the device with logical unit number XX is typed out. Subsequent typing of the device address BBB causes that device to be assigned the logical unit number XX, and the device with address AAA to be given the logical unit number YY, which previously designated device BBB (if any). As a result, every device in the system will always be designated by at most one logical unit number.

before			after		
XX	:	AAA	XX	:	BBB
YY	:	BBB	YY	:	AAA

c. EOB

Processing resumes with the next job in sequence.

The operator is also informed about abnormal conditions encountered by the error analysis routines of the elementary input/output programs contained in the monitor. The following messages are typed:

a. XX YYY NOT RDY

b. XX YYY NOT OP

c. XX YYY I/O ERROR CCCC DDDD

d. XX YYY DEV END CCCC DDDD

XX represents the logical number of the affected device, YYY its physical address, CCCC the encountered channel status, and DDDD the device status.

Message a. is given when the device is not ready. Execution resumes when the device is put into the ready state. Messages b., c., and d. are respectively given when a device is not operating, when a malfunction is encountered, or when an error is discovered upon device end interrupt caused by the reader, punch, or printer. The operator must reply with one of the following messages:

a. ignore EOB

b. exit EOB (resume processing with next job)

c. EOB (retry the operation after I/O ERROR; ignore the DEV END condition).

If the operator response is not recognized by the system, then "RETRY" is typed out. In order to cancel a response, the CANCEL character must be typed before typing EOB. In either case, a correct response should then be typed by the operator.

7. PL360 System Organization and Storage Requirements

This chapter contains information concerning the internal organization and storage requirements of certain parts of the PL360 systems. Knowledge of this chapter is not required for normal use of the systems, and certain sections assume a familiarity with OS control program services.

7.1. General Organization

Both PL360 systems consist of a monitor, originally coded in IBM 360 assembly language, and a set of PL360 system programs, originally coded in PL360, on a system tape (or OS sequential data set). The monitors are core-resident; they process requests for various supervisor services and also perform initialization functions. Among the PL360 system programs is a job sequencing routine which is also core-resident and, in conjunction with the monitor, processes PL360 system control cards. Other system programs are loaded upon demand as required by these control cards.

7.2. The PL360/OS System

7.2.1. Resource Requirements

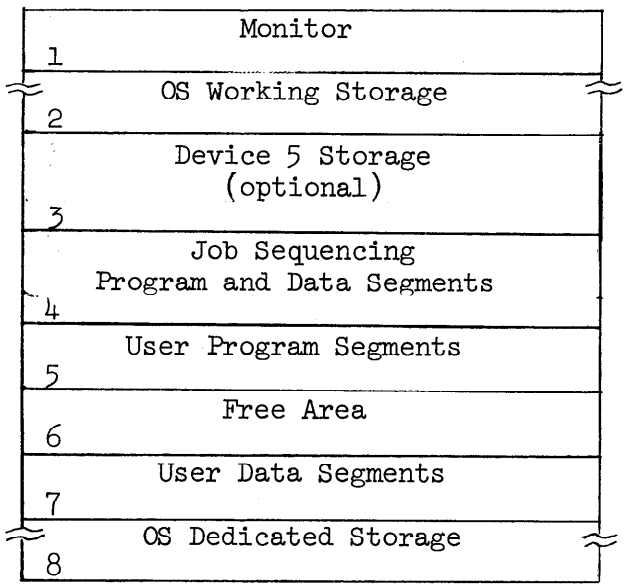
The PL360/OS system requires a machine configuration capable of supporting OS/360 and additionally providing

1. at least 64K bytes of storage for user programs, and
2. devices and space for at least two additional sequential data sets, one for the system library (about 140K bytes for a system consisting of the programs described in Chapter 3) and one for scratch use.

7.2.2. Storage Organization

During execution of a PL360 system job, storage is organized as indicated schematically in the following diagram.

OS Partition



The areas numbered 2 through 7 occupy a contiguous block of storage obtained by a single GETMAIN. Area 8 contains the buffers and access method routines for data sets opened unconditionally (see 7.2.3) as well as for certain OS control blocks allocated during the initialization process. Areas 4 through 6 are considered PL360 system storage areas. Area 2 is specifically released to OS to provide storage for buffers (device 3 only) and access routines required by data sets opened on demand (see 7.2.3) as well as for OS transient work areas. The monitor occupies approximately 5000 bytes. Sizes of the other areas are determined by the OS control program or by the values of certain symbolic quantities, defined in the monitor source code, as summarized

in the following table.

Description	Minimum	Maximum	Actual
Total Storage	COREMIN+SYSFREE	COREMAX+SYSFREE	x+SYSFREE
OS Working Storage	SYSFREE	SYSFREE	SYSFREE
Device 5 Storage (if specified)	COREMIN/2	BUFFMAX	min(x/2,BUFFMAX)
PL360 System Storage			
1. Device 5 in Core	COREMIN/2	max(COREMAX/2, COREMAX-BUFFMAX)	max(x/2, x-BUFFMAX)
2. Device 5 External	COREMIN	COREMAX	x

Standard values for the symbolic quantities are as follows:

COREMIN	64K	
COREMAX	512K	(K = 1024 bytes)
SYSFREE	4K	
BUFFMAX	128K	

7.2.3. Input/Output Routines

Characteristics of the input/output routines are summarized in the following table.

Logical Device	Access Method	Open Option	Open Selection
1	QSAM	INPUT	unconditional
2	QSAM	OUTPUT	unconditional
3	QSAM	OUTPUT	on demand
4	BSAM	INPUT	unconditional
5	BSAM	OUTIN	unconditional
6	BSAM	*	on demand
7	BSAM	*	on demand
8	BSAM	*	on demand
9	BSAM	*	on demand

* OUTIN if the function forcing opening implies write access; otherwise, INOUT.

7.3. The PL360 Stand-Alone System

7.3.1. Resource Requirements

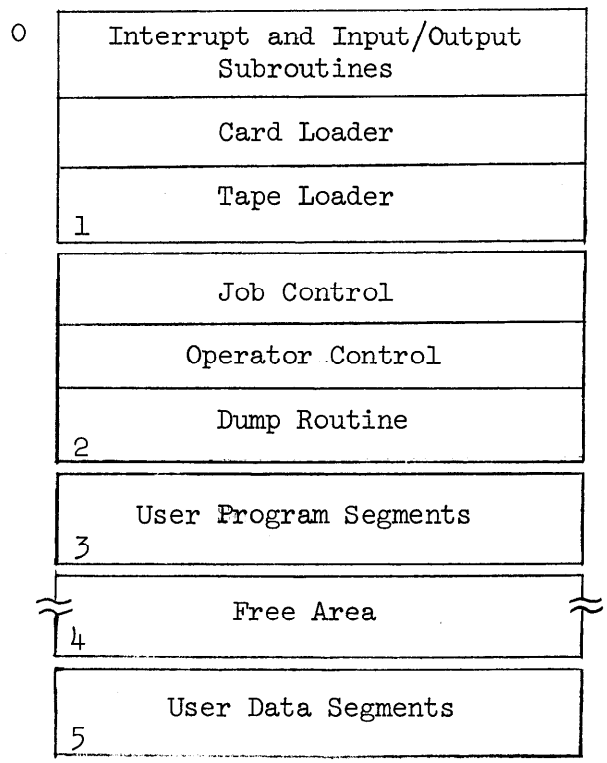
The minimal configuration requirements for the stand-alone version are as follows:

a. 64K bytes of core memory, with or without memory protection;

b. a line printer, a card reader, an operator's console typewriter, and two tape drives.

7.3.2. Storage Organization

During job execution, storage is organized as indicated schematically below:



Area 1 constitutes the monitor, which is a self-loading program placed at the beginning of the system tape. Areas 1 and 2 are memory protected (if possible) during the execution of each job.

7.3.3. Input/Output Routines

The input/output routines of the monitor generate channel programs for the following IBM devices.

Device Type	IBM Model Number
operator's console	1052
card reader	2540
line printer	1403
card punch	2540
tape drive	2400-2402

In many cases, such programs are suitable for similar devices with different model numbers.

Appendix: Additions and Changes to the PL360 Language

The PL360 language is described in a companion report [1]. The following additions and changes have been made to the language since that report was issued.

1. Function Format Code Extensions

Section 2.2.8 of [1] describes function declarations. The format code description given there does not provide for literal strings or numbers as parameters in function statements except as immediate data. Therefore, the function format codes have been extended to allow literal strings and numbers as parameters. The table on the following page describes the format and allowed parameters for each function code. An integer value parameter is considered to be illegal if the value is too large to fit into the specified field or if the value is negative. A cell designator used as a parameter for a single byte field is considered to be illegal if its relative address has a base register or if the displacement exceeds 255.

The following example shows the effective declaration of the standard function identifiers:

function		
	MVI(4,#9200),	CLI(4,#9500),
	MVC(5,#D200),	CLC(13,#D500),
	STM(3,#9000),	LM(3,#9800),
	SRDL(9,#8C00),	SLDL(9,#8D00),
	IC(2,#4300),	STC(12,#4200),
	LA(11,#4100),	RESET(8,#9200),
	SET(8,#92FF),	UNPK(10,#F300),
	CVD(12,#4E00),	EX(2,#4400),
	ED(5,#DE00),	TR(5,#DC00),
	TEST(8,#95FF)	

Format Code	No. of Parameter Fields in Function	Definition of Parameter Fields	R = ℵ register C = ℑ cell designator L = ℑ value or string (literal)** I = integer value* S = string*
0	0		
1	2	R R	
2	2	R LC	
3	3	R R C	
4	2	ICS C	
5	3	ICS C LC	
6	1	R	
7	1	ICS	
8	1	C	
9	2	R IC	
10	4	I I C LC	
11	2	R ICS	
12	2	R C	
13	3	ICS LC LC	
		0 8 16 32	

Table of Function Format Codes

---

* value used directly in instruction field.

** address of value used in instruction field.

2. Cell Declaration Additions

Section 2.2.4 of [1] describes cell declarations. Cell declarations have been modified to allow nested repetition of initialization elements in order to facilitate initializing arrays of cells to patterns of numbers or strings. The reserved word character has also been added to the language and is equivalent to the reserved word byte. The syntax for cell declarations becomes:

```
<simple byte type>  ::= byte | character
<simple short integer type>  ::= short integer
<simple integer type>  ::= integer | logical
<simple real type>  ::= real
<simple long real type>  ::= long real
<J type>  ::= <simple J type> | array <integer number> <simple J type>
<J cell declaration>  ::= <J type item> | <J cell declaration>,<item>
<item>  ::= <identifier> | <identifier> = <fill value>
<fill value>  ::= <J value> | <string> | @ <J cell designator> |
                  <repetition list> <fill value> )
<repetition list>  ::= ( | <integer number> ( |
                       <repetition list> <fill value> ,
```

a. Boundary alignment is performed only for the cell declaration; there is no further automatic alignment within the fill value. Care must be taken when initializing with combinations of numbers and strings in order to maintain the desired alignment for each number.

b. As a convenience to the programmer, the number of low order bytes appropriate for the type of the declared cell is taken for each J-value. Strings are never expanded or truncated.

c. short integer, integer, and logical cells can be initialized to the address of a $\mathcal{T}$ cell designator. Two or four bytes are filled with the relative address (i.e., index register number, base register number, and displacement) of the $\mathcal{T}$ cell designator. The index register must be zero when initializing short integer cells. Cells cannot be initialized to absolute addresses.

d. The repetition count is specified by the integer number in the repetition list. If the count is omitted, it is assumed to be one. The count must not be negative. If the count is zero, no initialization takes place for the list.

e. The total length of the fill value for each item must be less than or equal to the space allocated for the item.

Examples:

```
short integer  I = 10S, J = (5),  BADDR = @B5
array 132 byte line = 132 (" "),  buff = 4(33(" "))
array 15 integer X = 3 (@line, "ABCD",3(5))
```

3. Standard Integer Identifier Declarations

A set of fourteen standard integer identifiers have been declared. These identifiers are often useful in function statements. Their declaration is equivalent to the following:

```
integer   MEM syn 0;
integer   B1 syn MEM(R1),  B2 syn MEM(R2),
          B3 syn MEM(R3),  B4 syn MEM(R4),
          B5 syn MEM(R5),  B6 syn MEM(R6),
          B7 syn MEM(R7),  B8 syn MEM(R8),
          B9 syn MEM(R9),  B10 syn MEM(R10),
          B11 syn MEM(R11),  B12 syn MEM(R12),
          B13 syn MEM(R13);
```

4. Global and External Segment Procedures

A primitive method of defining global and external segments to allow linkage to separately compiled PL360 procedures has been provided. These facilities are primarily intended for system use; however, these features can be used by any programmer in order to avoid recompiling certain standard parts of his programs.

The following syntax updates section 2.2.11 (Segment Base Declarations) and section 2.3.7 (Procedure Declarations) of [1] in order to include the global and external features:

```
<segment head>              ::= segment | global <integer number> |
                                external <integer number>
<segment base declaration>  ::= <segment head> base <integer register>
<procedure heading>         ::= procedure <identifier> (<integer register>);|
                                <segment head> procedure <identifier>
                                (<integer register>);
<procedure declaration>     ::= <procedure heading> <statement>
```

Program and data segment numbers are assigned consecutively starting at 00 by the PL360 compiler to each new segment, not explicitly numbered by its declaration, when it is encountered. In order to use conveniently a compiled segment as an external segment for another program, the programmer can use the global segment facility to specify assignment of a particular segment number. The integer number in the segment head is taken as the segment number. The permissible range of segment numbers for user programs is 00 to 63. Use of such a segment in another program requires the declaration of an external data or program segment with the same segment number. References between

external segments are by number only. No identifiers are used.

The following comments apply to the use of global and external segments:

a. Global program and data segments are distinguished by the compiler only by the fact that segment numbers are assigned by the programmer instead of the compiler.

b. External program and data segments are compiled in exactly the same way as other segments. However, these segments produce no compiler object output. Thus any data initialization specified or any compiled code is ignored. (The body of an external procedure normally is the statement null.) Such segment declarations are used only to name and describe the corresponding segment. However, the programmer must insure that each use of a separately compiled procedure containing cell declarations has a valid base register for the declarations of that procedure. This could mean that the declarations must be repeated each time a corresponding external procedure declaration is used in another program or that the procedure contains its declarations in global or external data segments declared within the procedure. The problem occurs because declarations are static in PL360 and not dynamic.

c. Since the external base declaration loads the specified base register and since all variables declared in the segment will use that register as the base register for addressing, the base register used in the external declaration need not be the same as the register used when the segment was compiled using the global declaration. The segment numbers must be the same for both forms.

d. Since the return register is used in both the call and the return from a procedure, the external and global procedure declarations must specify the same return register as well as the same segment number.

e. Object card decks for previously compiled external segments can be used with PL360 (see Section 3.1.1) and with the $TAPEn deck indications for SYSTUP (see Section 3.5.4). External segments can be recognized in object decks by referring to the description of object deck formats (Section 3.1.6).

References

[1] N. Wirth, "A programming language for the 360 computers", Technical Report CS 53 (revised), Stanford University, June 1967 (also, J. ACM. 15, 37 (January 1968)).

[2] H. R. Bauer, S. Becker, and S. L. Graham, "ALGOL W" , Technical Report CS 89, Computer Science Department, Stanford University, January 1968.

[3] N. Wirth and H. Weber, "EULER, a generalization of ALGOL, and its formal definition", Part 1, Comm. ACM. 9, 1 (January 1966), pp. 13-23.

[4] IBM System/360 Principles of Operation, IBM Systems Reference Library, Form A22-6821.

[5] IBM System/360 Operating System Concepts and Facilities, IBM Systems Reference Library, Form C28-6535.

[6] IBM System/360 Operating System Job Control Language, IBM Systems Reference Library, Form C28-6539.