CS 68

THE PL360 SYSTEM

Edited
By

NIKLAUS WIRTH

TECHNICAL REPORT NO. CS 68
JUNE 5, 1967

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

THE PL360 SYSTEM

Edited

By

Niklaus Wirth

Computer Science Department

Stanford University

Stanford, California

THE PL360 SYSTEM

Table of Contents

1. Introduction and Survey

This report describes the use and the organization of the operating system which serves as the environment of the PL360 language defined in the Companion Report CS 53 [1].

The core of the system consists of 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 library programs is available to the system and can be accessed by its program loader. Their use is described in Chapter 3. Among other programs, the library contains the PL360 compiler. Paragraph 3.1. defines the symbol representations, the restrictions imposed on the language by this implementation, the various available system subroutines, particularly those for input-output, and the usage of the compiler.

Chapter 4 indicates the error messages provided by the resident supervisor routines upon the occurrence of program checks and other unintended situations.

While chapters 2-4 are intended to serve as a user's reference manual, chapters 5 and 6 outline the internal organization of the system for the reader interested in further detail. The PL360 system was first developed as a stand-alone, self-loading program, and its properties are therefore optimally adapted to the few requirements of the language. This system is described in Chapter 5. A second version was designed to operate as a subsystem under IBM's OS/360, and consists of a set of linkage routines, as described in Chapter 6.

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

1. Core memory, 64K bytes, with or without memory protection.
2. Printer (1403), card reader/punch (2540), 2 tape drives.

The entire system was designed with the aim of a convenient, efficient, and easy-to-use tool for the development of compilers and operating systems. The set of library routines reflects this fact clearly. Apart from the lister and card duplication programs, there exists a program to edit tape files, which the compiler is able to accept as input in place of cards. Moreover, a system generation program

is provided to perform the rapid creation of new systems or the inclusion of new or modified library programs. Listings of cross-references of identifiers (in any language, and PL360 in particular) can be produced by another library program to facilitate work with large scale source programs. And finally, the Syntax Processor is used in the development of compilers based on the principle of analysis of precedence grammars.

A further program to be available under the system is the Algol 66 compiler for whose development the PL360 project was undertaken. Since at this time the Algol compiler is not yet generally available, the description of its use is not incorporated in this report.

The implementation of the PL360 system and most of the library programs was conducted by Mr. J. Wells. Mr. E. Satterthwaite developed the linkage routines to OS/360 described in Chapter 6, and Mrs. J. Keckler wrote the cross-referencing library program (3.7).

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

2. Jobcontrol instructions

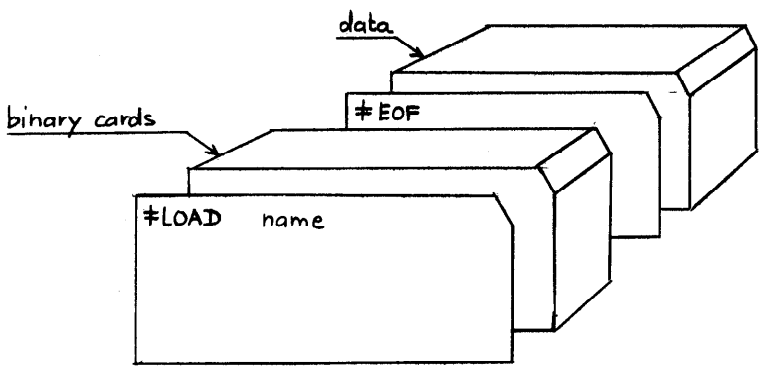
Jobs in the input batch are separated by control cards to be interpreted by the jobcontrol routine. They are characterized by a 0-2-8 punch in column 1, (denoted by ≠), and, if encountered by the READ routine, give rise to an end-of-file indication. Information contained in columns 2-9 (left adjusted) of control cards is inspected and interpreted as follows:

LOAD — The subsequent cards are supposed to be "binary" (punched by a compiler). They are loaded, and execution of the loaded program is initiated.

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

PAUSE — The operator is notified and the system waits for instructions to be entered at the console (cf. 5.2).

An input deck using the LOAD instruction has the following composition:



If any other text is contained in columns 2-9, it is interpreted as the name of a program to be loaded from the system library. Information in the remaining columns is ignored. The following programs are presently available from that library, and they are described in the following chapter:

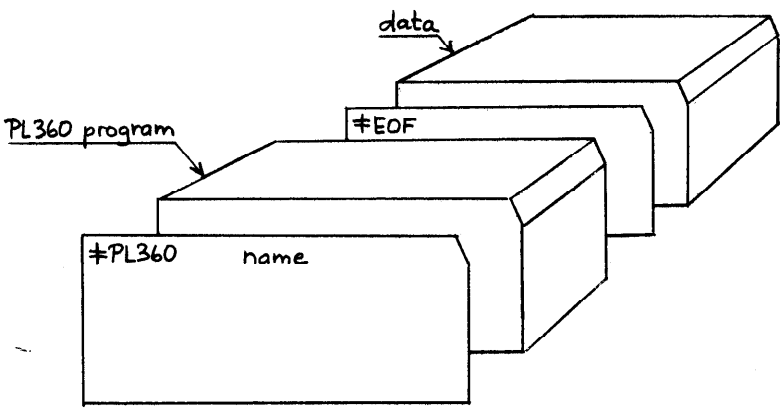
PL360	The PL360 compiler
DUPDECK	Program to duplicate card decks
LISTER	Program to list card decks

SYNPROC	Syntax processor program
TUP	Tape updating program
SYSTUP	System tape updating program
XREF	Program to generate table of cross-references of identifiers

3. Description of system programs

3.1. The PL360 compiler (PL360)

3.1.1. The form of input decks



3.1.2. The language

The PL360 programming language is described in the companion report CS 53 (revised):

"PL360, a programming language for the 360 computers".

The subsequent paragraphs specify the details pertinent to the present implementation, such as symbol representation and specific limitations.

3.1.2.1. Symbol representation

Only capital letters are available. Basic symbols which consist of underlined letter sequences in the Report, 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 SHRA
STEP THEN
ARRAY BEGIN SHORT UNTIL WHILE
COMMENT INTEGER LOGICAL SEGMENT
FUNCTION OVERFLOW REGISTER
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 Report CS53 or in paragraph 3.1.2.4.

MEM FPI
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
F0 F2 F4 F6
F01 F23 F45 F67
LA MVI MVC CLI CLC LM STM
SLDL SRDL IC STC CVD UNPK
ED EX SET RESET
READ WRITE PUNCH PAGE
READTAPE WRITETAPE REWIND MARKTAPE
FSPTM FSPREC BSPTM BSPREC
DUMP FPIRESET

3.1.2.3. Restrictions

The implementation imposes the following restriction upon the language:

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

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

3.1.2.4. Supervisor functions

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 (cf. X.8.).

READ — Read a card, assign the 80 character record to the memory area designated by the address in register R0. Set the condition code 1, if a control card is encountered (cf. also chapters 2 and 4).

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 (cf. 5.2.). 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 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 written record in bytes is specified by register R1; the record is designated by the address in register R0.

PAGE — Skip to the next page on the line printer.

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: Fowardspace one record.

BSPTM: Backspace past the previous tape mark.

FSPTM: Forwardspace past the next tape mark.

A program interruption (cf. [3]) due to arithmetic operations records the interruption code in the byte cell FPI . This cell, being part of the supervisor, is memory protected, and cannot be reset by the user's program directly.

FPIRESET: Reset the value of the cell FPI to 0 .

DUMP A specified area of memory is printed in hexadecimal form. Register R0 must specify the starting address of the area and R1 must specify its length (in bytes). The values of R2 and the condition code are altered by a call of the dump routine.

WRITETIME The time elapsed since the beginning of execution of the present program is printed. The values of registers R0-R2 and the condition code are altered.

3.1.2.5. Examples of programs

```
BEGIN COMMENT MAGIC SQUARE GENERATOR;
   ARRAY 132 BYTE LINE = " ";
   ARRAY 8 BYTE PATTERN = "        ";
   LONG REAL DEC;
   ARRAY 256 INTEGER X;

PROCEDURE MAGICSQUARE (R6);
   COMMENT THIS PROCEDURE ESTABLISHES A MAGIC SQUARE OF ORDER N,
      IF N IS ODD AND 1 < N < 16. X IS THE MATRIX IN LINEAR FORM.
      REGISTERS R0 ... R6 ARE USED, AND R0 CONTAINS N AS PARAMETER.
      ALGORITHM 118 (COMM.ACM, AUG.1962);
   BEGIN SHORT INTEGER NSQR;
      INTEGER REGISTER N SYN R0, I SYN R1, J SYN R2, Y SYN R3;
      INTEGER REGISTER IJ SYN R4, K SYN R5;
      NSQR := N; R1 := N * NSQR; NSQR := R1;
      I := N+1 SHRL 1; J := N;
      FOR K := 1 STEP 1 UNTIL NSQR DO
      BEGIN Y := I SHLL 6; IJ := J SHLL 2 + Y; Y := X(IJ);
         IF Y ¬= 0 THEN
         BEGIN I := I-1; J := J-2;
            IF I < 1 THEN I := I+N;
            IF J < 1 THEN J := J+N;
            Y := I SHLL 6; IJ := J SHLL 2 + Y;
         END ;
         X(IJ) := K;
         I := I+1; IF I > N THEN I:= I-N;
         J := J+1; IF J > N THEN J := J-N;
      END ;
   END ;

PROCEDURE GENERATE (R8);
   BEGIN INTEGER REGISTER I SYN R1, J SYN R2, IJ SYN R4, N SYN R6;
      J := 0; FOR I := 0 STEP 4 UNTIL 1020 DO X(I) := J;
      MAGICSQUARE;
      N := R0;
      FOR I := 1 STEP 1 UNTIL N DO
      BEGIN IJ := I SHLL 6 +4; R5 := @LINE(4);
         FOR J := 1 STEP 1 UNTIL N DO
         BEGIN MVC(5, B5, PATTERN); R3 := X(IJ); CVD(R3, DEC);
            ED(5, B5, DEC(5)); IJ := IJ+4; R5 := R5+7;
         END ;
         R0 := @LINE; WRITE;
      END ;
      ED(131, LINE, LINE); WRITE;
   END ;

   ED(131, LINE, LINE); COMMENT  FILL LINE WITH BLANKS;
   R0 := 3; GENERATE;
   R0 := 5; GENERATE;
   R0 := 9; GENERATE;
END .
```

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

```
BEGIN COMMENT TEST OF PROCEDURES INREAL AND OUTREAL;
   ARRAY 132 BYTE LINE = " ";
PROCEDURE INREAL (R5);
   COMMENT READS A CARD AND SCANS FOR A CHARACTER SEQUENCE REPRESEN-
      TING A REAL NUMBER. RESULT IN F01. USED ARE R0...R5 AND F0...F7;
   BEGIN INTEGER REGISTER CHAR SYN R0, ACCUM SYN R1, SCALE SYN R2;
      INTEGER REGISTER EXT SYN R3;
      LONG REAL REGISTER ANSWER SYN F01;
      BYTE SIGN, EXPOSIGN;
      LONG REAL CONVERTED = #4E0000000000000L;
      INTEGER CONVERT SYN CONVERTED(4);
      FUNCTION LTR (1,#1200);
      INTEGER INDEX;
      ARRAY 80 BYTE CARD;

   PROCEDURE NEXTCHAR (R4);
      BEGIN R3 := INDEX + 1;
         IF R3 > 71 THEN
         BEGIN R0 := @CARD; READ; IF ¬= THEN GOTO ALLDONE; R3 := 0;
         END;
         R0 := R0-R0; LC(R0,CARD(R3)); INDEX := R3;
      END ;

      R0 := 71; INDEX := R0; NEXTCHAR;
      WHILE R0 < "0" DO
      BEGIN IF R0 = "-" THEN SET (SIGN) ELSE RESET(SIGN); NEXTCHAR;
      END ;
      COMMENT  ACCUMULATE THE INTEGRAL PART IN ACCUM;
      ACCUM := CHAR AND #F; NEXTCHAR;
      WHILE CHAR >= "0" DO
      BEGIN CHAR := CHAR AND #F; ACCUM := ACCUM * 10S + CHAR; NEXTCHAR;
      END ;
      SCALE := 0;
      CONVERT := ACCUM; ANSWER := CONVERTED + 0L;
      IF CHAR = "." THEN
      BEGIN COMMENT PROCESS FRACTION. ACCUMULATE NUMBER IN ANSWER;
         NEXTCHAR;
         WHILE CHAR >= "0" DO
         BEGIN CHAR := CHAR AND #F; CONVERT := CHAR;
            ANSWER := ANSWER * 10L + CONVERTED;
            SCALE := SCALE - 1; NEXTCHAR;
         END ;
      END ;
      IF CHAR = "'" THEN
      BEGIN COMMENT READ SCALE FACTOR AND ADD IT TO SCALE;
         NEXTCHAR; IF CHAR = "-" THEN
            BEGIN SET(EXPOSIGN); NEXTCHAR;
            END ELSE
            IF CHAR = "+" THEN
            BEGIN RESET(EXPOSIGN); NEXTCHAR;
            END ELSE RESET(EXPOSIGN);
         ACCUM := CHAR AND #F; NEXTCHAR;
         WHILE CHAR >= "0" DO
         BEGIN CHAR := CHAR AND #F; ACCUM := ACCUM * 10S; NEXTCHAR;
         END ;
         IF EXPOSIGN THEN SCALE := SCALE - ACCUM
                     ELSE SCALE := SCALE + ACCUM;
      END ;
      IF SCALE ¬= 0 THEN
      BEGIN COMMENT F45 := 10 ** SCALE;
```

```
            IF SCALE < 0 THEN
            BEGIN SCALE := ABS SCALE; SET(EXPOSIGN);
            END ELSE RESET(EXPOSIGN);
            F23 := 10L; F45 := 1L; F67 := F45;
            WHILE SCALE ¬= 0 DO
            BEGIN SRDL (SCALE,1); F23 := F23*F67; F67 := F23;
               LTR (EXT,EXT); IF < THEN F45 := F45*F23;
            END ;
            IF EXPOSIGN THEN ANSWER := ANSWER / F45
                        ELSE ANSWER := ANSWER * F45;
         END ;
         IF SIGN THEN ANSWER := NEG ANSWER;
      END ;

   PROCEDURE OUTREAL (R4);
      BEGIN COMMENT CONVERT NUMBER IN F01 INTO CHARACTER STRING AT @R1;
         FUNCTION LTR (1,#1200);
         INTEGER REGISTER EXP SYN R0, SCALE SYN R2, EXT SYN R3;
         LONG REAL REGISTER X SYN F01;
         LONG REAL CONVERT;
         INTEGER CONVERTED SYN CONVERT(4), EXPO SYN CONVERT(0);
         BYTE SIGN;
         IF X = 0L THEN MVC (13,B1,"  0           ") ELSE
         BEGIN IF X < 0L THEN SET(SIGN) ELSE RESET(SIGN);
            X := ABS X; CONVERT := X;
            COMMENT OBTAIN AN ESTIMATED DECIMAL SCALE FACTOR FROM
               EXPONENT PART;
            EXP := EXPO SHRL 24 - 64 * 307S; IF < THEN EXP := EXP + 255S;
            EXP := EXP SHRA 8 - 1; SCALE := ABS EXP;
            COMMENT COMPUTE F45 := 10 ** SCALE;
            F23 := 10L; F45 := 1L; F67 := F45;
            WHILE SCALE ¬= 0 DO
            BEGIN SRDL (SCALE,1); F23 := F23*F67; F67 := F23;
               LTR (EXT,EXT); IF < THEN F45 := F45*F23;
            END ;
            COMMENT NORMALIZE TO 1 <= X < 10 ;
            IF EXP < 0 THEN
            BEGIN X := X * F45;
               WHILE X < 1L DO
               BEGIN X := X * 10L; EXP := EXP-1;
               END ;
            END ELSE
            BEGIN X := X / F45;
               WHILE X >= 10L DO
               BEGIN X := X * 0.1L; EXP := EXP+1;
               END ;
            END ;
            X := X * 1'7L ++ #4E000000000000005L;
            CONVERT := X; EXT := CONVERTED;
            COMMENT EXT IS HERE USED TO HOLD RESULTING INTEGER;
            IF EXT >= 100000000 THEN
            BEGIN EXT := EXT / 10; EXP := EXP +1;
            END ;
            MVC (13,B1,"    .       '   ");
            CVD (EXT,CONVERT); ED (9,B1,CONVERT(3));
            IF SIGN THEN MVI ("-",B1(1));
            CVD (EXP,CONVERT); ED (3,B1(10),CONVERT(6));
            IF EXP < 0 THEN MVI ("-",B1(11)) ELSE MVI ("+",B1(11));
         END ;
      END ;
      MVC (130, LINE(1), LINE(0));  COMMENT FILL LINE WITH BLANKS;
   L: INREAL; R1 := @LINE(6); OUTREAL; R0 := @LINE; WRITE; GOTO L;
   ALLDONE:
   END .
```

3.1.3. Instructions to the compiler

The compiler accepts instructions inserted anywhere in the sequence of input 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 subsequent execution.
$LIST	List subsequent source records on the printer.
$NOLIST	Do not list subsequent source records.
$PUNCH	Punch compiled program and data segments on cards.
$PAGE	Skip a page in the listing.
$0	Print source text only.
$1	Indicate the addresses of all variables and procedures upon their declaration.
$2	List addresses as after $1. Also list the produced machine code in hexadecimal notation.
$TAPEn	Read the subsequent source records from the tape unit with logical number n. If n is omitted, tape unit 7 is assumed.

3.1.4. Error messages of the compiler

Errors are indicated by the compiler with a message and a bar below the character which was read last.

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	A real register instead of an integer register is specified in a for clause.
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 is specified in a shift operation.
06	COMPARE TYPES	The types of comparands are incompatible.
07	REG TYPE OR #	Incorrect register specification.
08	UNDEFINED ID	An undeclared identifier is encountered.
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.

Error No.	Message	Meaning
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.
16	PROGRAM OFLOW	The current program segment is too large.
17	INITIAL OFLOW	The area of initialized data in the compiler is full. This can be circumvented by suitable segmentation.
18	ADDRESS OFLOW	The number used as index is such that the resulting address cannot be accommodated.
19	NUMBER OFLOW	The number is too large in magnitude.
20	MISSING .	An end-of-file has been read before a program terminating "." was encountered.
21	STRING LENGTH	The length of a string is either 0 or > 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.
24	ILLEGAL PARAM	A parameter incompatible with the specifications of the function is used.

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

3.1.5. 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. It starts with 1 for each segment.

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

3.2. Program to duplicate card decks (DUPDECK)

J. Wells

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 starting with 00001 and in increments of 1 .
$NOSEQUENCE	No sequencing 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 four option cards which are not listed and can appear anywhere in the deck.

$SEQUENCE	The following cards are listed with a card count starting with one and in increments of one. Sequencing is the initial option.
$NOSEQUENCE	No sequencing numbers are listed.

3.4. Tape Updating Utility Program (TUP)

The TUP starts out in the command mode. It reads and interprets a sequence of commands, each of which is punched on a card beginning in column one. If on any card "LISTER" is punched in columns 12-17, the information processed during the interpretation of the command is listed on the printer. The commands are:

$INPUTn	Unit A is assigned the logical device number n (in decimal). This can be used only before a command that uses the input tape A .
$OUTPUTn	Unit B is assigned the logical device number n (in decimal). This can be used only before a command that uses the output tape B .

$NEWTAPE — The subsequent card deck is read and put onto tape on unit B . Every record is provided with a sequence number. The increment is 10 . All 80 columns of the cards can be used. Note: No other command may follow $NEWTAPE .

$LISTER — The information on tape unit A is read and listed, including the sequence numbers.

$PUNCH — The information on tape unit A is read and punched without sequence numbers.

$RESEQUENCE — The records on unit A are read, provided with new sequence numbers (increment 10), and written onto tape on unit B .

$UPDATE — The 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 detected. m and n are five digit numbers punched in columns 12-16 and 20-24 respectively. (Leading zeroes must be punched!) If n is missing, only one card is deleted.

$INSERT m — The subsequent card records are inserted after the record with sequence number m . All cards are treated as data to be inserted, up to the command card

$END

The inserted records are provided with sequence numbers with increments of one. All records on the input tape A having a sequence number identical to one given to an inserted record are deleted. All 80 columns on data cards may be used.

$END	The TUP returns to the update mode.
$LISTER m	Listing starts (or resumes) with record m . m is punched in columns 12-15 (cf. $DELETE).
$NOLISTER m	Listing stops at the record with sequence number m . Note: all inserted and replaced records are listed in any case.
Other cards:	All other cards are treated as data cards, and must be provided with a sequence number in columns 75-80 . If its sequence number coincides with the sequence number of a record on the input tape, then this record is replaced by the one read from the card, otherwise the card record is inserted at the appropriate place.
Note:	Cards in the update deck must be properly sequenced, i.e. the numbers on "other cards," and the parameter m on command cards must be an increasing sequence. If there are no cards in the update deck, then tape A is simply copied onto tape B .

The standard tape assignments are:

Input Unit A: logical unit 6 (182)
Output Unit B: logical unit 7 (183)

The PL360 compiler uses logical unit 7 as its standard source tape input unit when $TAPE is specified, however $TAPEn (in decimal) can be used in the compiler to correspond to unit B of TUP .

3.5. System Tape Updating Program (SYSTUP)

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

3.5.1. Program Identification Card

The copy, punch, and update modes of the SYSTUP are controlled by a program identification card. Columns 2-9 of this card are the program name by which this program is recognized by job control as well as by SYSTUP, columns 10-72 are simply a comment field to be used for version identification, columns 73-80 are the version date field. If the date field is blank then the current date is put into the date field. The program identification card becomes the first record for the program on the output tape. Column one is completely ignored on the card.

3.5.2. Mode Control Cards

$INPUTn	The input tape is assigned the logical device number n (in decimal). Device 4 is the standard input unit if $INPUT is not used.
$OUTPUTn	The output tape is assigned the logical device number n (in decimal). Device 9 is the standard output unit if $OUTPUT is not used.
$PUNCH	This indicates a punch run for the SYSTUP. All the programs specified by program identification cards following this card are punched. Each program punched has the tape program identification card first followed by the object deck and ending with an EOF control card so that the deck is in the form that SYSTUP uses to load a program. At each program identification card, the input tape is rewound and searched for the specified program. Therefore, there is no specific order required for the cards. $COPY or $LIST should not be used in a punch run.

$COPY — This indicates a copy run for the SYSTUP. All the programs specified by program identification cards following this card are copied onto the new system tape. These programs are found on the input tape by rewinding the input tape each time and searching the tape for the program. A monitor is put on the new tape from one of the three sources described below. A copy run can be used to reorder a system tape. $PUNCH or $LIST should not be used in a copy run.

$LIST — This indicates a listing of the programs on the input tape is desired. The input tape is rewound and the program identification headers are listed on the printer. For a list run the entire deck should be the $LIST and possibly a $INPUT card followed by an end-of-file card.

If a $PUNCH, $COPY, or $LIST card is not encountered in the system control cards, then an update run is performed. Any update run is a sequential merge between the changes specified by the card deck and the input tape. The update function behaves in the following three ways:

1. If the current card is $INSERT then one of the following occurs:
   a. if columns 10-17 are blank, then the deck set following the $INSERT card is immediately loaded onto the output tape without refering to the input tape;
   b. if columns 10-17 are not blank, then the input tape is copied to the output tape up to and including the program whose name is in columns 10-17 . Then the following deck set is loaded as in case "a."
2. If the current card 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 to the output tape up to the program name on the program identification card. If an object deck indication follows the identification card then the new version of the program is loaded on to the output tape; otherwise the program is deleted from the output tape.

3. If the end of the card deck has been reached, then the rest of the input tape is copied on to the output tape. Note that this means that an update run with no update deck simply copies all the input tape programs to the output tape.

Therefore, the update mode has the following general features:

1. All programs on the input tape that are not specified in the update deck are simply copied to the output tape in the same order.

2. Programs can be inserted, changed, or deleted.

3. The EOF card separates each and every 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 just as if the end of the input tape had been read before starting that step.

3.5.3. Deck Indications

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

1. The card object deck immediately follows in the card reader. It is loaded until the next end-of-file card (usually an EOF card).

2. $TAPE with columns 10-17 blank indicates that the program is unlabelled and can be found on the scratch tape. The scratch tape is rewound and loaded.

3. $TAPE with columns 10-17 not blank indicates that the program is to be found labelled on another system tape mounted on the scratch unit. The scratch tape is rewound and searched for the program named in columns 10-17 . That program is then loaded. Note that the program name on the $TAPE card need not be the same as the one on the program identification card. Therefore, this feature can be used to rename a program.

If the desired program is on a tape other than the scratch tape (logical device 5) then $TAPEn can be used to specify the logical device (in decimal). However, n can not be the same as the input tape.

3.5.4. Monitor for New Tape

In order to make a self loading tape for either an update or copy run, it is necessary to first put a copy of the monitor on the new tape. Normally the monitor can just be copied from the input tape. Therefore, this was made the default option. However, the following two sources for the new monitor are also allowed.

1. $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 end-of-file card (usually a EOF card) signifies the end of the monitor deck. After the monitor has been loaded, the normal update of system programs is done unless the input tape is logical unit 0 . In that case the decks following are loaded without using an input tape. This allows SYSTUP to be used to make the first system tape also.

2. $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). The address at which execution will start must be in the half word starting

at memory location 20 (decimal). The use of this feature is intended mainly to facilitate the making of system tapes with different device assignments.

The monitor is always loaded starting at the timer position whenever the system tape is initial-program-loaded. Any information wanted in core below the timer must be moved there by the monitor after it has been loaded.

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 and insert an object deck of PNAME immediately after LISTER.

```
≠PL360
$NOGO
{LISTER source deck}
```

```
≠EOF
≠SYSTUP
$INPUT6
$OUTPUT8
 LISTER
$TAPE
≠EOF
$INSERT
 PNAME
 { object deck }
≠EOF
```

Note: The SYSTUP deck is ended only by reading the next job card or a pause card.

3.6. 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-g functions as described in EULER by Wirth and Weber [2]. The main input to the processor consists of the productions of the language. Each production is punched on one card. Columns 1-72 of the card are used for the production divided into six 12-character symbol fields. The left 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 1-5 symbols punched in columns 13-24, 25-36, 37-48, 49-60, 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 should 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 non-terminal symbol groups can be reordered in any desired fashion. In this way the user can specify his own symbol numbers.

$SYMPUNCH If the symbol table was calculated by the processor, then this causes the symbol table to be punched in the form used by SYNPROC (including the "**$SYMBOLS**" and two "**$$**" cards).

$CHECK

After the symbol table has been listed, a check is made whether any productions have 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 more than 199 symbols.

$LEFT

In general it is tedious to look up relations using the precedence matrix. If the precedence matrix exists, then $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 the output.

$RIGHT

This is entirely analagous 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.

$TAPE
$TAPEn

The results of the syntax processor are put on the system scratch tape (or tape n if n is specified). If the matrix does not exist then no tape output is made. If the functions are asked for then they are output on tape if they exist or no tape output is made if they don't 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 in order of their 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 have been doubled in the table in order to facilitate half word indexing in function calculations. The symbol number 0 fills out all right parts that were 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.

4. Matrix

The matrix is output in a completely packed form with 2 bits used for each relation (00 no relation, 01 < relation, 10 > relation, 11 = relation). The processor can handle a maximum of 256 symbols so there are 64 bytes for each row. The first row signifies 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.

3.7. Program to generate cross-reference tables of identifiers (XREF)

J. Keckler

The cross-reference routine will list alphabetically all identifiers in a source program with the numbers of the lines on which they occurred. An identifier is defined as a string of one or more letters and digits, the first character being a letter. According to various input options, one may request that certain identifiers not be references (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.

Control Cards

1. ≠XREF
2. $PL360 — to ignore all PL360 basic tokens, standard identifiers and identifiers on card(s) 3, cross-referencing all others

   or

   $IGNORE — to ignore the identifiers on card(s) 3, monitoring all others

   or

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

   ($IGNORE is assumed if card 2 is omitted)
3. A list of identifiers in free field, taking as many cards as necessary.

   (If omitted, the list is assumed to be empty)
4. $NOLIST — to supress the listing of the input program

   or

   $LIST — to list the input program

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

5. $CARD — indicates the input program is on cards

   or

   $TAPEn — indicates input program is on logical tape unit n, where n is a one or two digit integer. If n is omitted, logical unit 7 is assumed.

   (Card 5 must not be omitted)

6. input source deck, if on cards, followed by ≠EOF

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
500 total characters of special identifiers

Sample Deck Setups

```
≠XREF
$CARD
input deck
≠EOF
```

```
≠XREF
$PL360
$CARD
input deck
≠EOF
```

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

4. Program execution (run-time) errors

The following error conditions can occur at run time and are diagnosed by the supervisor. They result in program termination, (unless otherwise specified), and a dump of the data area of the interrupted program.

a. A "program-check" interruption occurred. This is indicated by the message

PRG PSW XXXXXXXXXXXXXXXX

where XX ... X denotes the program status word upon interruption in hexadecimal notation. If interruption occurred due to an arithmetic operation, the interruption code is stored in the byte cell FPI (floating point interruption), and control is returned to the interrupted program.* Such interrupts are counted, and the counts are listed (if $\neq 0$) after the end of program execution.

b. An attempt is made to read beyond the present job card file. The message

EOF PSW XXXXXXXXXXXXXXXX

is printed.

c. An illegal logical unit number has been used for an input-output operation. The message

DEV PSW XXXXXXXXXXXXXXXX

is printed.

d. The operator intervenes by causing an external interrupt. The message

EXT PSW XXXXXXXXXXXXXXXX

appears on the line printer and the operator console, and the system expects to receive instructions from the operator.

*Also, the condition code is set to 3 .

5. The PL360 system

5.1. Initial loading and operating the system

The process of initial loading consists of the following steps:

a. Reset system
b. Mount system tape on any 9-track unit
c. Stack jobs on the card reader
d. Make card reader, line printer, and tape 5 (used by the compiler) ready
e. Select the unit carrying the system tape on the Load Unit Switches
f. Press the Load Key
g. Enter the date (8 characters) from the typewriter

Execution of the job sequence stacked on the card reader is immediately started. Control is returned to the operator when either

a. a PAUSE control card is encountered, or
b. the operator presses the External Interrupt key.

The computer then accepts instructions from the operator via typewriter. 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 at the initial address XXXXXX are listed 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 (cf. 5.2).

b. device XX EOB

Devices are designated by logical numbers. The correspondence between logical numbers and actual device addresses is established by the device table. The above command causes the address AAA of

the device with logical unit number XX to be 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

The standard device assignment used on the SLAC computer is:

0	Typewriter	009
1	Printer	00E
2	Cardreader	00C
3	Card Punch	00D
4	System tape	282
5	Tape	181 (7-track)
6	Tape	182
7	Tape	183
8	Tape	184
9	Tape	283

c. EOB

Processing resumes with the next job in sequence.

The operator is informed about abnormal conditions encountered by the error analysis routines of the elementary input - output programs contained in the supervisor. 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 afflicted device, YYY its physical address, CCCC the encountered channel status, and DDDD the device status.

Message a. is given when a 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)

Note that if a storage dump is desired before processing the next job, then the interrupt key must be pressed first. 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.

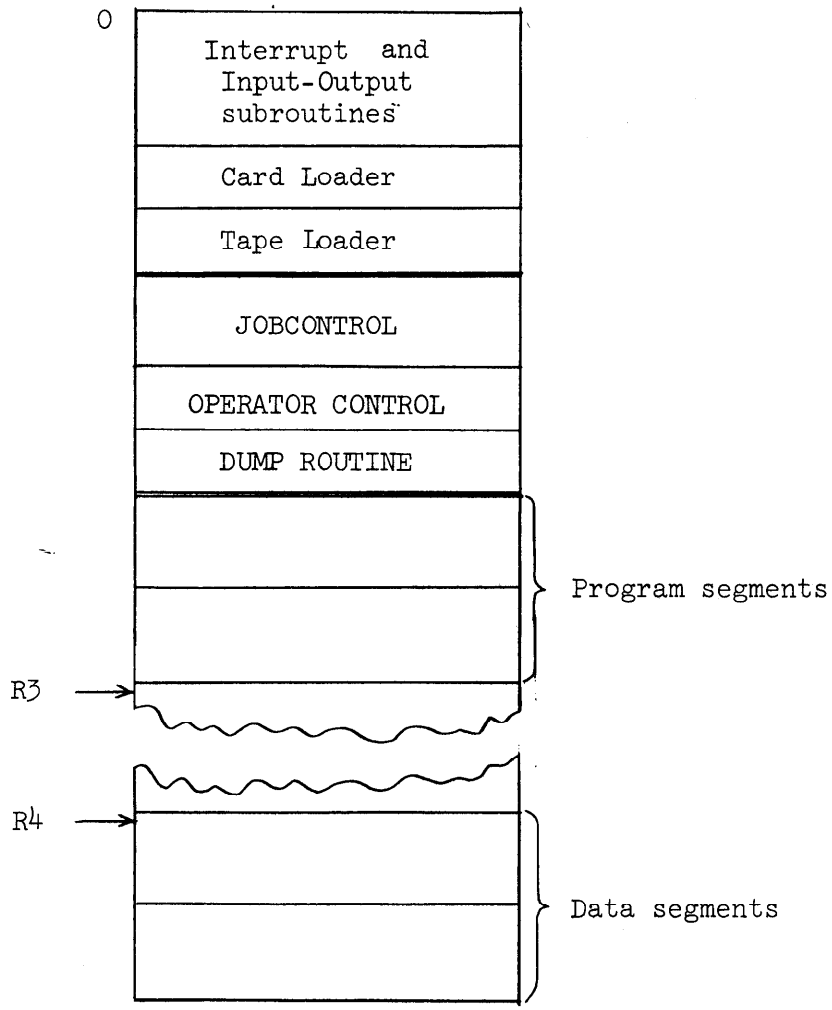
5.2. System organization

The PL360 system is a resident monitor which is logically divided into two parts:

1. The interrupt service routines, including the SVC routines performing input-output.

2. The jobcontrol routine which acts as the main program.

The storage layout during execution of a program is as follows:



Upon initial program loading, the core size and the availability of memory protection are determined. Before a program is loaded, memory is cleared. The program is then treated as a procedure and called with the following information in the registers:

R2 = return address

R3 = address of first byte following the "last" program segment

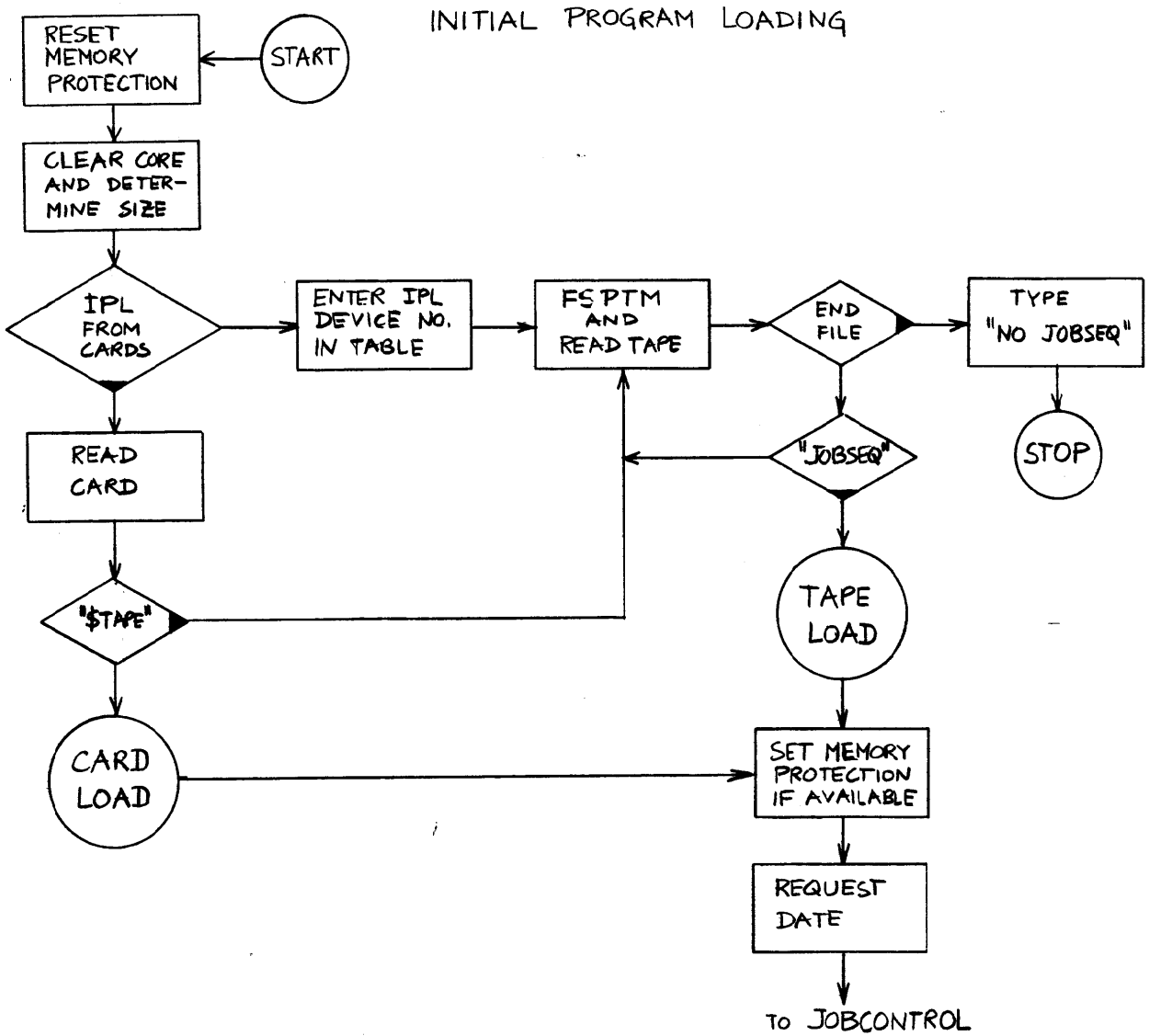
R4 = address of first byte of the "first" data segment

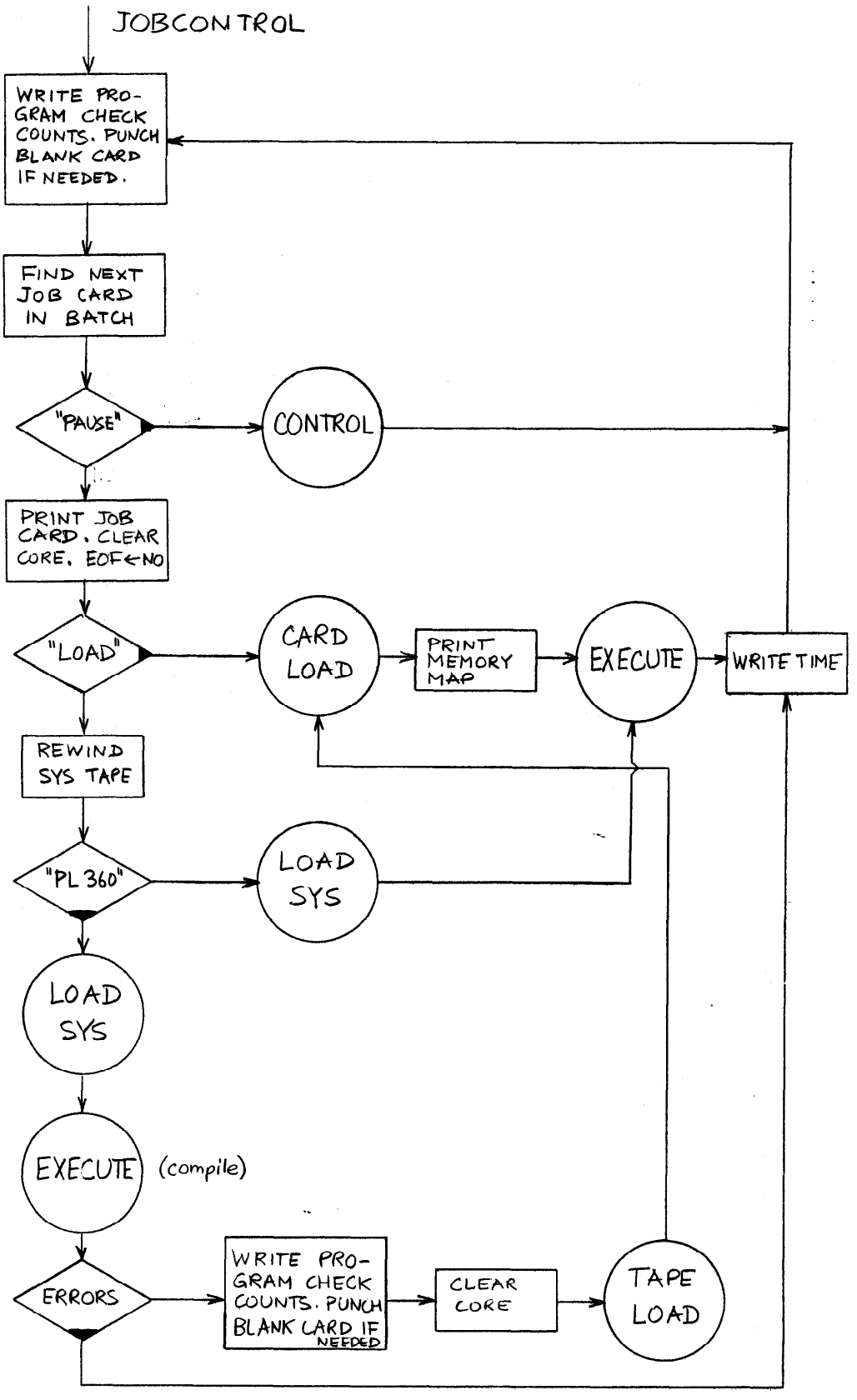
PL360 programs are compiled as if they were declared as segment procedures. Additionally, the return address in R2 is saved upon entry and restored upon exit.

During execution of a program, the monitor is memory protected (if possible). The protected area includes the segment reference table, which consists of 64 entries.

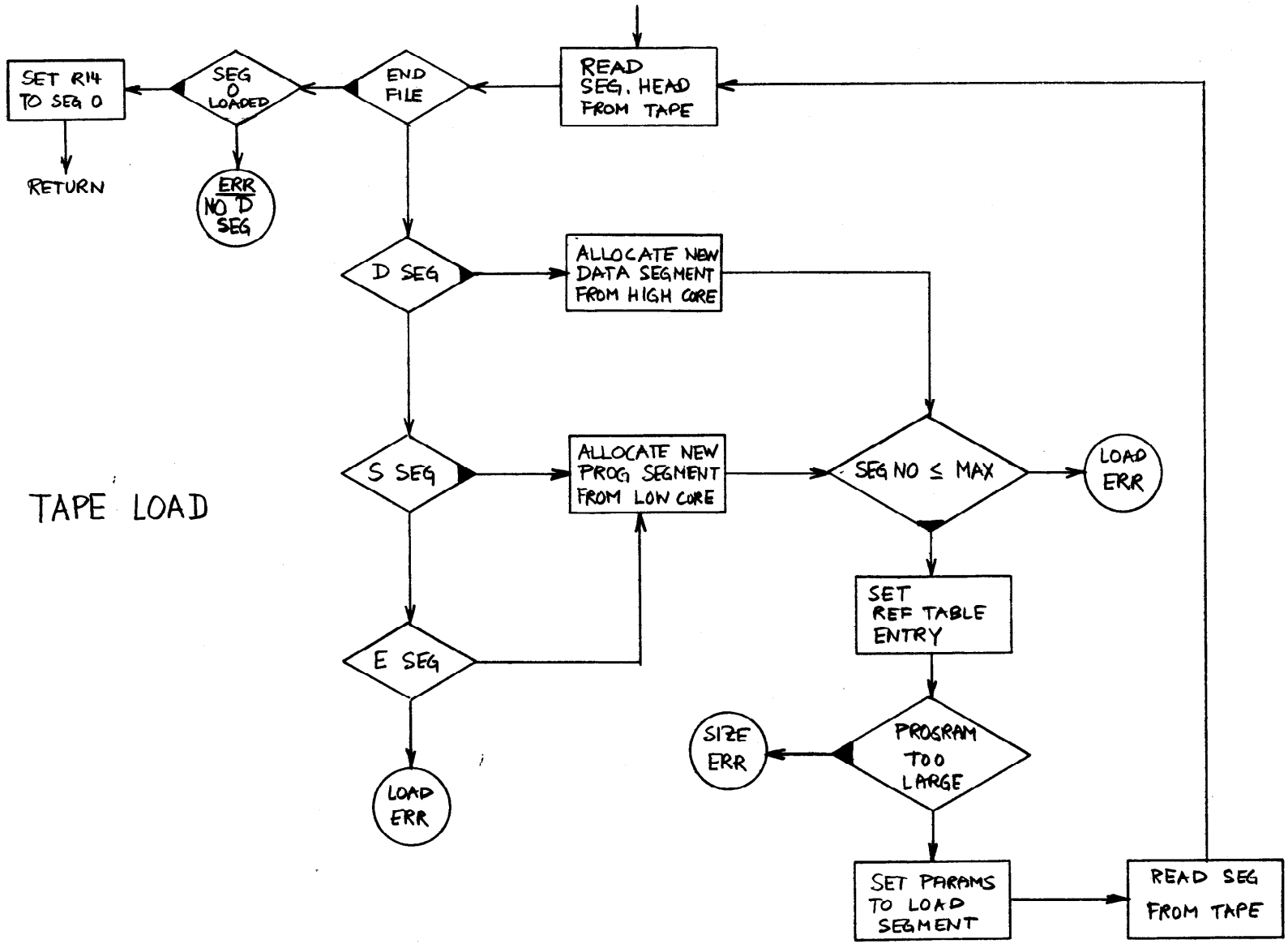
On the following pages, flowcharts are given of the main system routines:

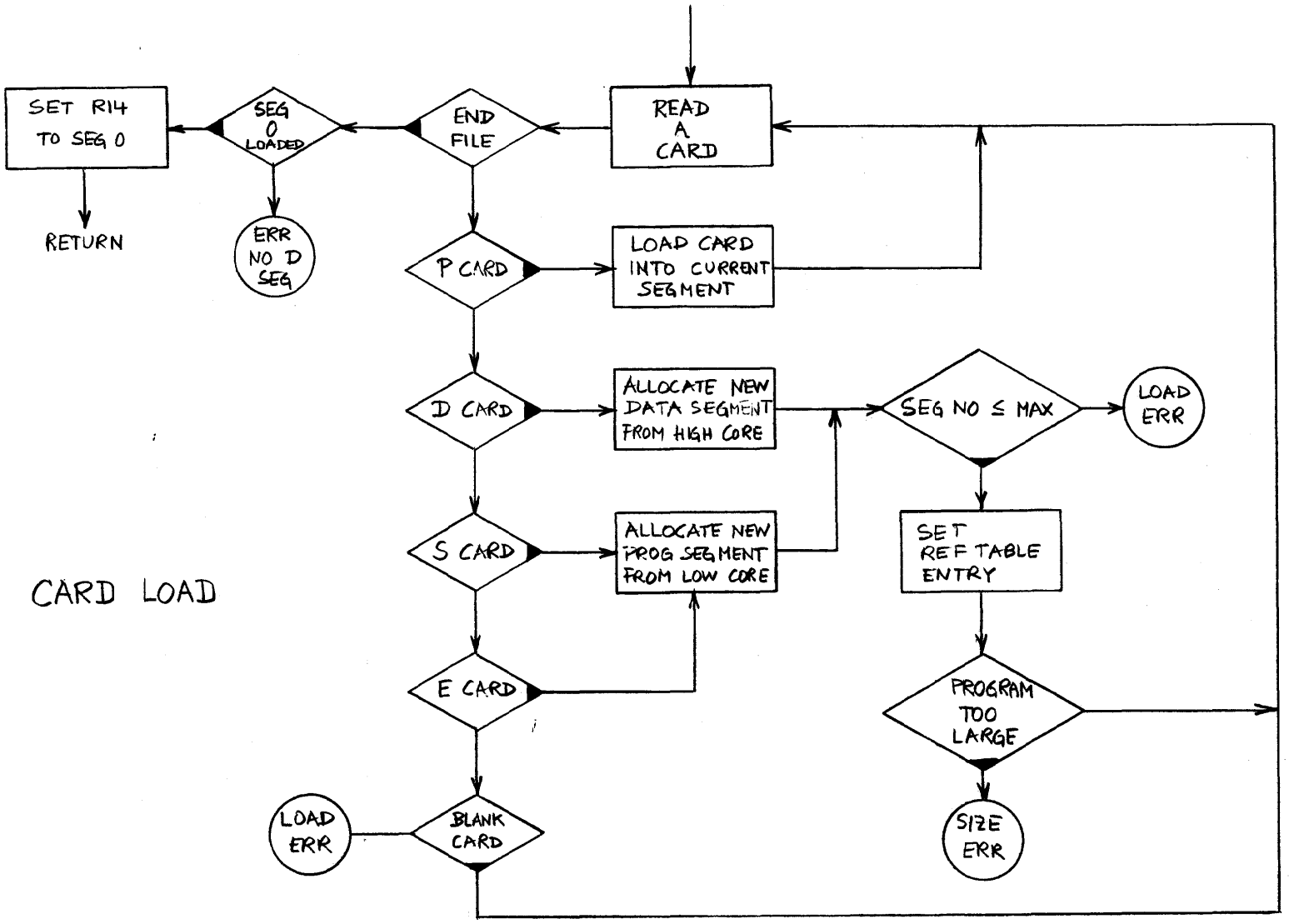
Table of Flowcharts


INITIAL PROGRAM LOADING
START
RESET MEMORY PROTECTION
CLEAR CORE AND DETER-MINE SIZE
IPL FROM CARDS
ENTER IPL DEVICE NO. IN TABLE
FSPTM AND READ TAPE
END FILE
TYPE "NO JOBSEQ"
STOP
"JOBSEQ"
READ CARD
"$TAPE"
TAPE LOAD
CARD LOAD
SET MEMORY PROTECTION IF AVAILABLE
REQUEST DATE
To JOBCONTROL

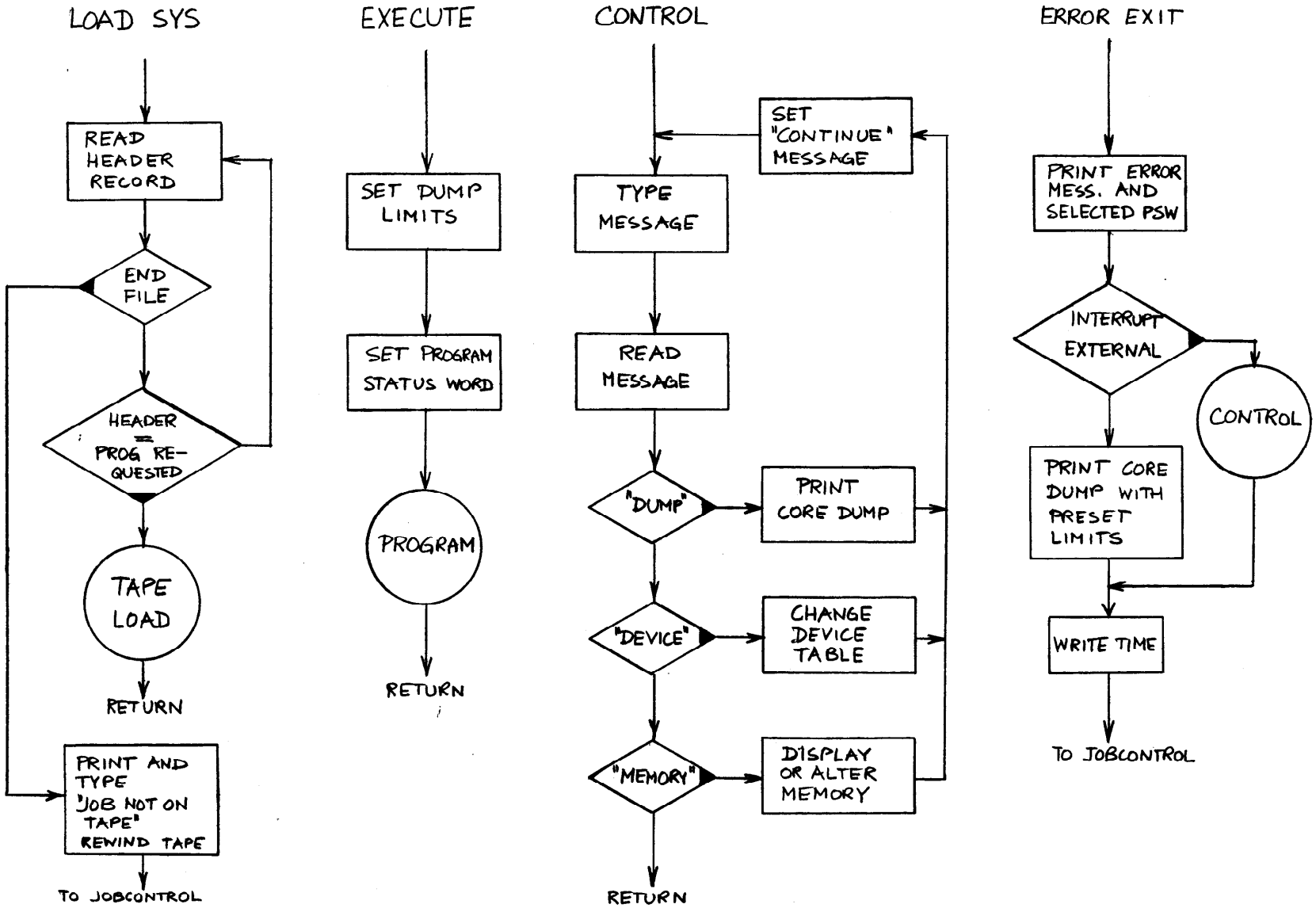

JOBCONTROL
WRITE PRO-GRAM CHECK COUNTS. PUNCH BLANK CARD IF NEEDED.
FIND NEXT JOB CARD IN BATCH
"PAUSE"
CONTROL
PRINT JOB CARD. CLEAR CORE. EOF←NO
"LOAD"
CARD LOAD
PRINT MEMORY MAP
EXECUTE
WRITE TIME
REWIND SYS TAPE
"PL 360"
LOAD SYS
LOAD SYS
EXECUTE
(compile)
ERRORS
WRITE PRO-GRAM CHECK COUNTS. PUNCH BLANK CARD IF NEEDED
CLEAR CORE
TAPE LOAD

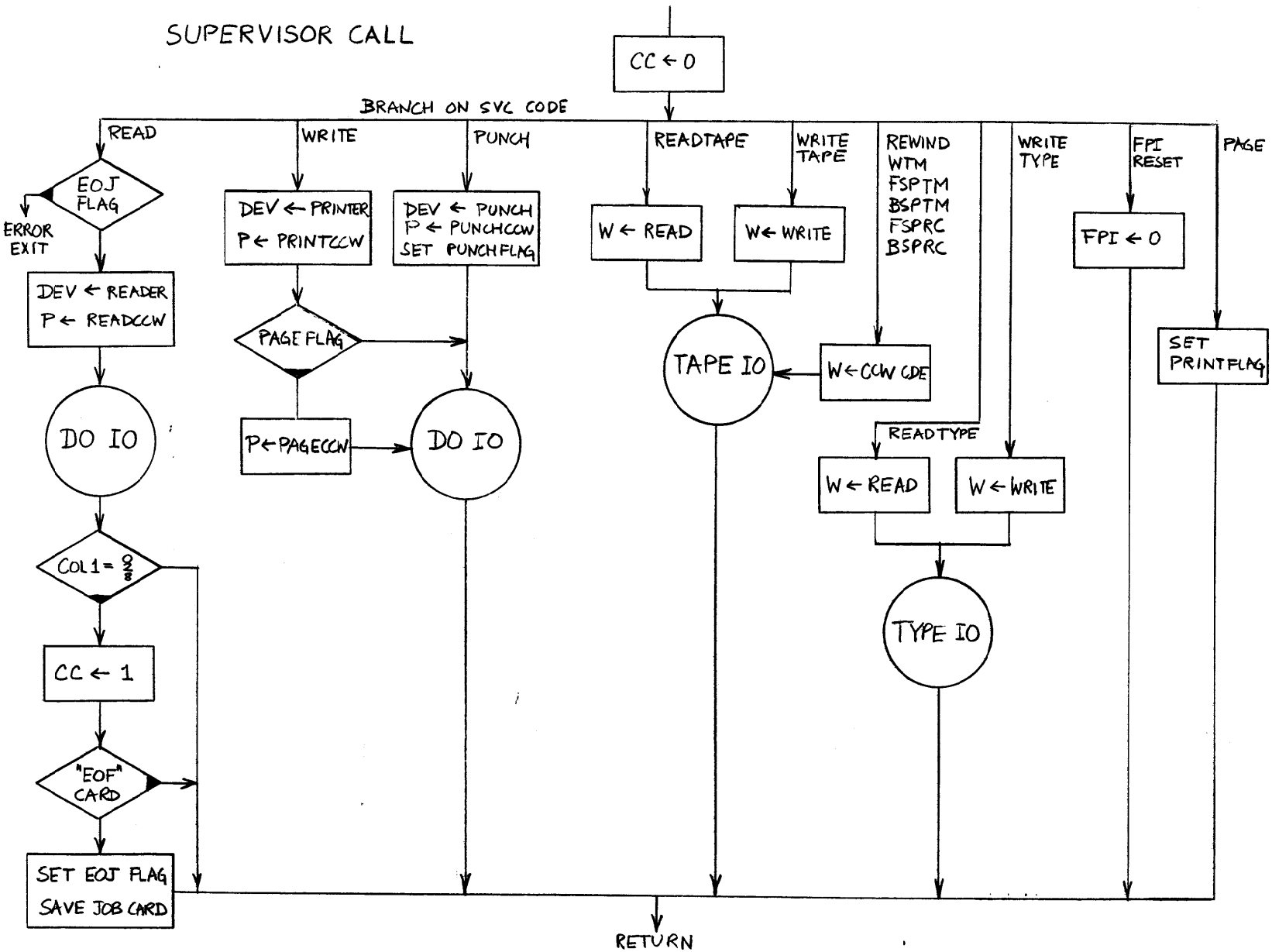
TAPE LOAD

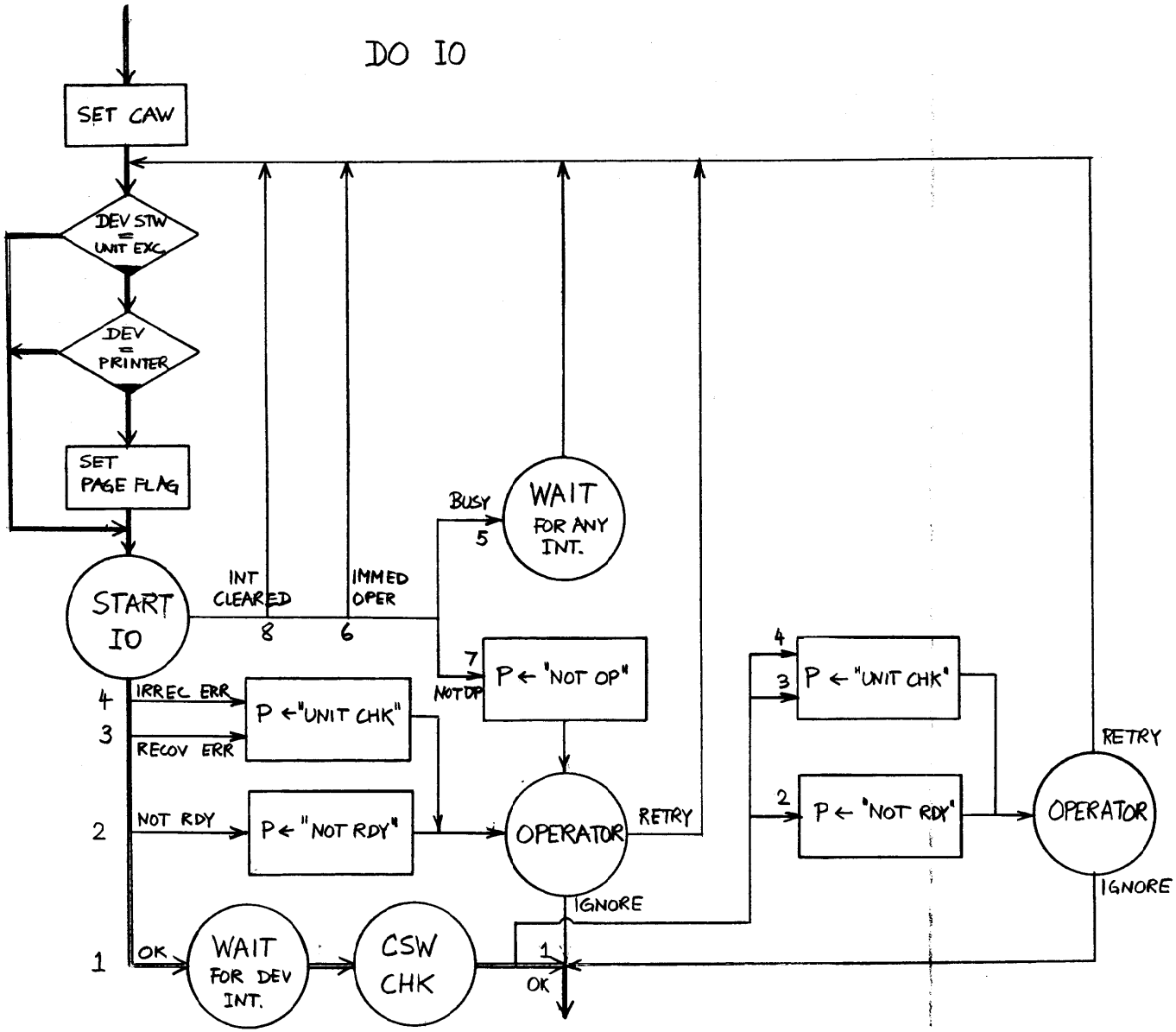

READ SEG. HEAD FROM TAPE
END FILE
SEG 0 LOADED
SET R14 TO SEG 0
RETURN
ERR NO D SEG
D SEG
ALLOCATE NEW DATA SEGMENT FROM HIGH CORE
S SEG
ALLOCATE NEW PROG SEGMENT FROM LOW CORE
SEG NO ≤ MAX
LOAD ERR
E SEG
LOAD ERR
SET REF TABLE ENTRY
PROGRAM TOO LARGE
SIZE ERR
SET PARAMS TO LOAD SEGMENT
READ SEG FROM TAPE

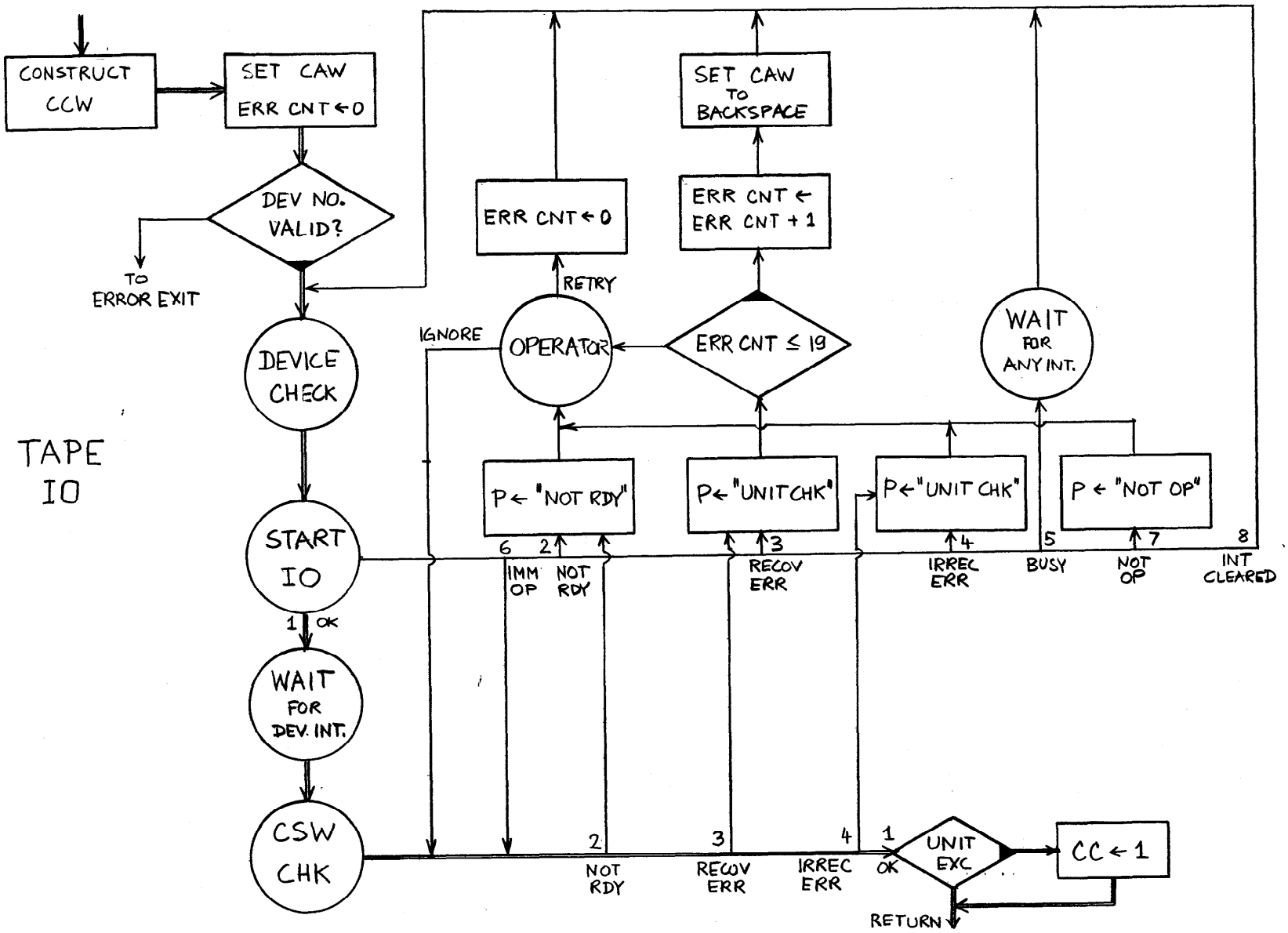

READ A CARD
END FILE
SEG 0 LOADED
SET R14 TO SEG 0
RETURN
ERR NO D SEG
P CARD
LOAD CARD INTO CURRENT SEGMENT
D CARD
ALLOCATE NEW DATA SEGMENT FROM HIGH CORE
SEG NO ≤ MAX
LOAD ERR
S CARD
ALLOCATE NEW PROG SEGMENT FROM LOW CORE
SET REF TABLE ENTRY
E CARD
PROGRAM TOO LARGE
SIZE ERR
BLANK CARD
LOAD ERR

CARD LOAD

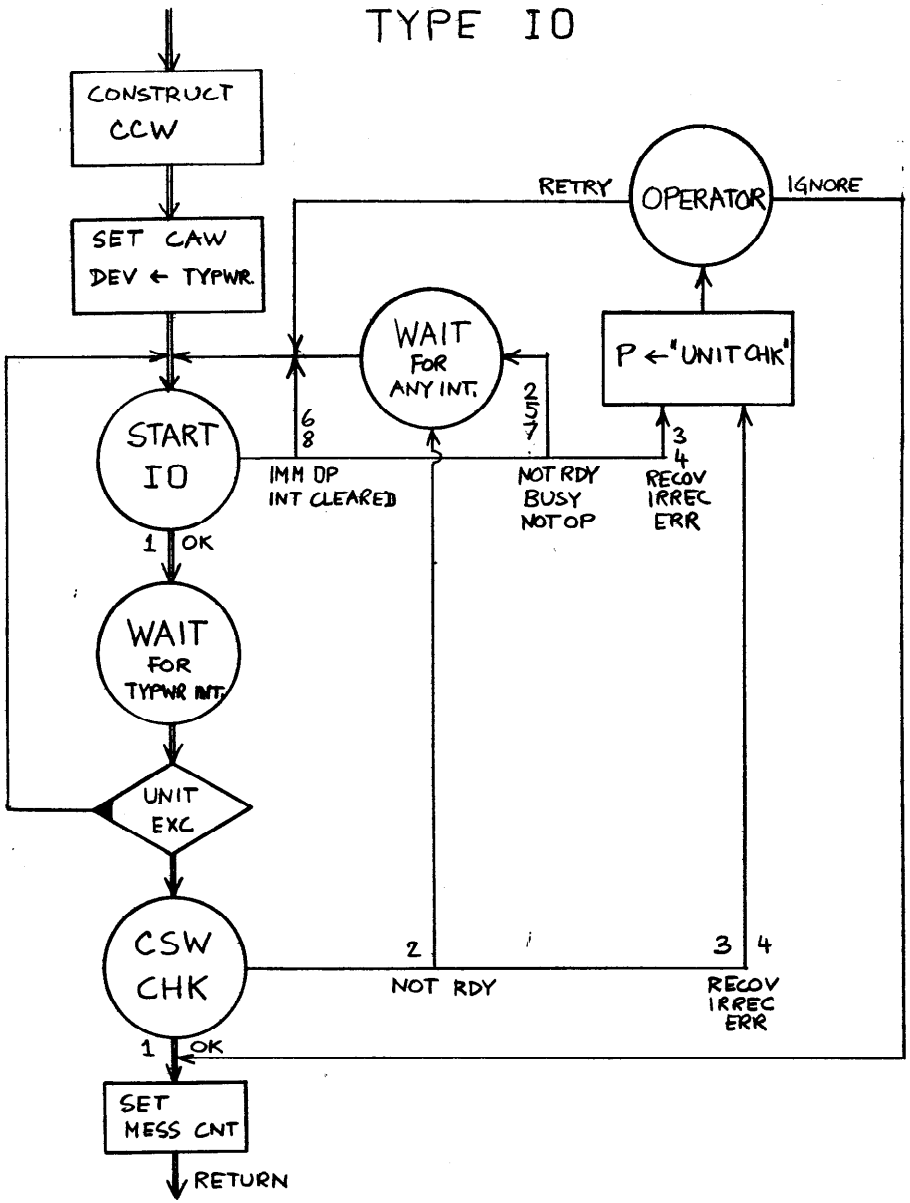

LOAD SYS
READ HEADER RECORD
END FILE
HEADER = PROG REQUESTED
TAPE LOAD
RETURN
PRINT AND TYPE "JOB NOT ON TAPE" REWIND TAPE
TO JOBCONTROL
EXECUTE
SET DUMP LIMITS
SET PROGRAM STATUS WORD
PROGRAM
RETURN
CONTROL
SET "CONTINUE" MESSAGE
TYPE MESSAGE
READ MESSAGE
"DUMP"
PRINT CORE DUMP
"DEVICE"
CHANGE DEVICE TABLE
"MEMORY"
DISPLAY OR ALTER MEMORY
RETURN
ERROR EXIT
PRINT ERROR MESS. AND SELECTED PSW
INTERRUPT EXTERNAL
CONTROL
PRINT CORE DUMP WITH PRESET LIMITS
WRITE TIME
TO JOBCONTROL

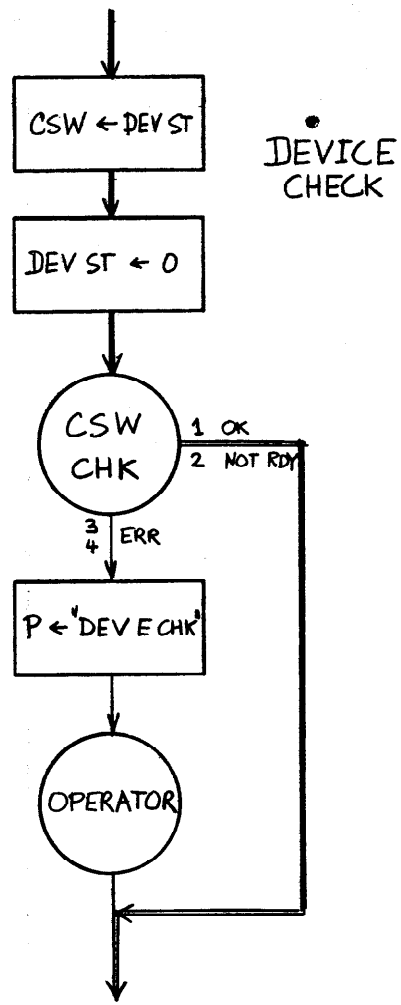

SUPERVISOR CALL
CC ← 0
BRANCH ON SVC CODE
READ
WRITE
PUNCH
READTAPE
WRITE TAPE
REWIND
WTM
FSPTM
BSPTM
FSPRC
BSPRC
WRITE TYPE
FPI RESET
PAGE
EOJ FLAG
ERROR EXIT
DEV ← READER
P ← READCCW
DO IO
COL 1 = 9
CC ← 1
"EOF" CARD
SET EOJ FLAG
SAVE JOB CARD
DEV ← PRINTER
P ← PRINTCCW
PAGE FLAG
P ← PAGECCW
DEV ← PUNCH
P ← PUNCHCCW
SET PUNCH FLAG
W ← READ
W ← WRITE
TAPE IO
W ← CCW CODE
READTYPE
W ← READ
W ← WRITE
TYPE IO
FPI ← 0
SET PRINT FLAG
RETURN


DO IO
SET CAW
DEV STW = UNIT EXC.
DEV = PRINTER
SET PAGE FLAG
START IO
INT CLEARED
IMMED OPER
8
6
BUSY
5
WAIT FOR ANY INT.
7
NOT OP
P ← "NOT OP"
4
IRREC ERR
3
RECOV ERR
P ← "UNIT CHK"
2
NOT RDY
P ← "NOT RDY"
OPERATOR
RETRY
IGNORE
1
OK
WAIT FOR DEV INT.
CSW CHK
4
3
P ← "UNIT CHK"
2
P ← "NOT RDY"
OPERATOR
RETRY
IGNORE
1
OK

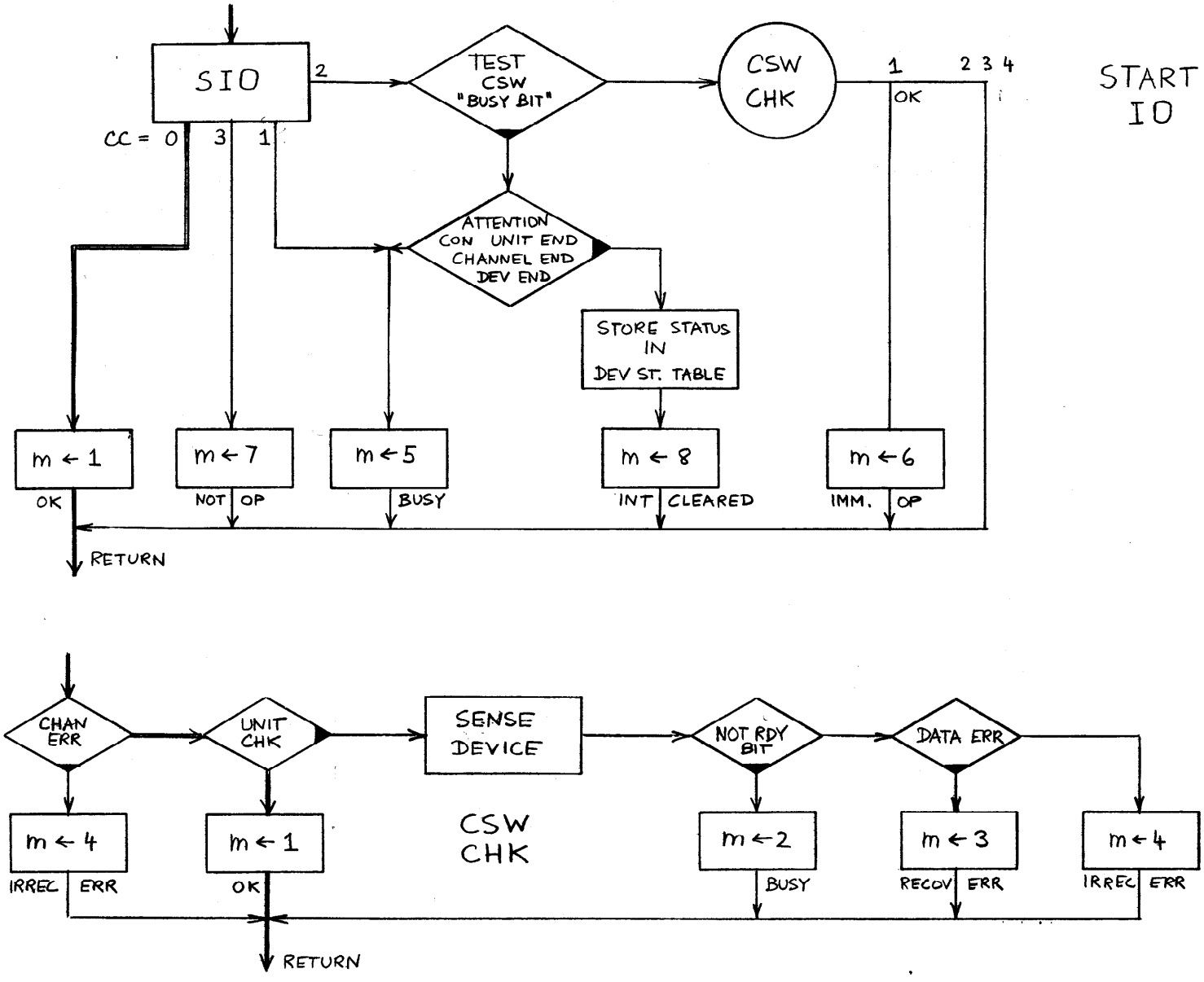

TAPE IO
CONSTRUCT CCW
SET CAW ERR CNT ← 0
DEV NO. VALID?
TO ERROR EXIT
DEVICE CHECK
START IO
1 OK
WAIT FOR DEV. INT.
CSW CHK
SET CAW TO BACKSPACE
ERR CNT ← ERR CNT + 1
ERR CNT ← 0
RETRY
IGNORE
OPERATOR
ERR CNT ≤ 19
WAIT FOR ANY INT.
P ← "NOT RDY"
P ← "UNIT CHK"
P ← "UNIT CHK"
P ← "NOT OP"
6 IMM OP
2 NOT RDY
3 RECOV ERR
4 IRREC ERR
5 BUSY
7 NOT OP
8 INT CLEARED
2 NOT RDY
3 RECOV ERR
4 IRREC ERR
1 OK
UNIT EXC
CC ← 1
RETURN

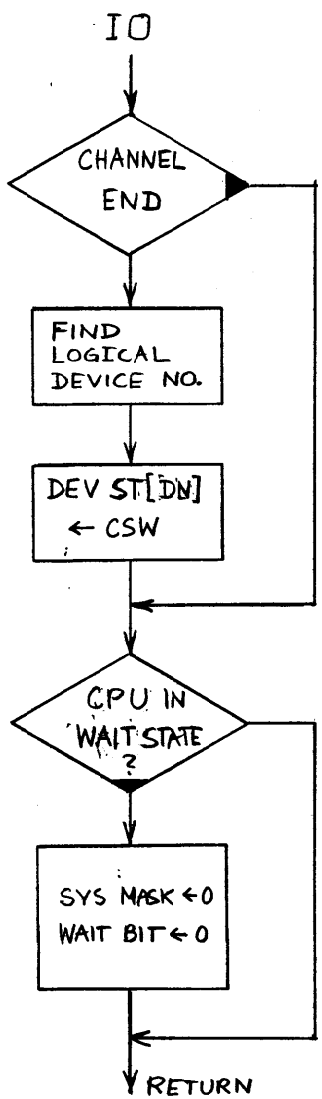
TYPE IO

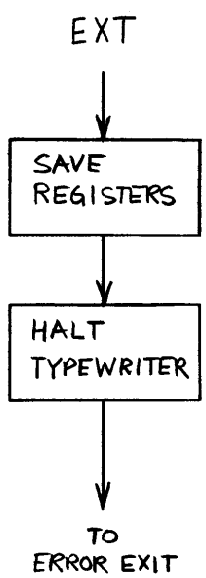


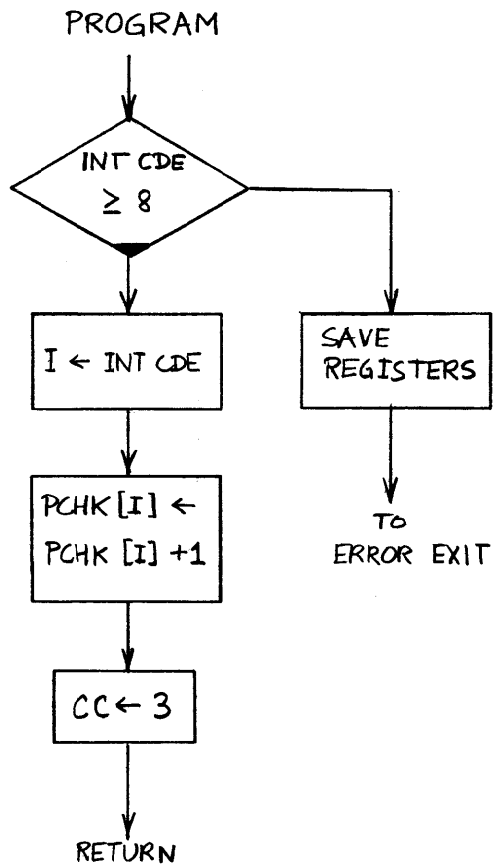


DEVICE CHECK


START IO
SIO
2
TEST CSW "BUSY BIT"
CSW CHK
1
2 3 4
OK
CC = 0
3
1
ATTENTION CON UNIT END CHANNEL END DEV END
STORE STATUS IN DEV ST. TABLE
m ← 1
m ← 7
m ← 5
m ← 8
m ← 6
OK
NOT OP
BUSY
INT CLEARED
IMM. OP
RETURN
CHAN ERR
UNIT CHK
SENSE DEVICE
NOT RDY BIT
DATA ERR
CSW CHK
m ← 4
m ← 1
m ← 2
m ← 3
m ← 4
IRREC ERR
OK
BUSY
RECOV ERR
IRREC ERR
RETURN


IO
CHANNEL END
FIND LOGICAL DEVICE NO.
DEV ST[DN] ← CSW
CPU IN WAIT STATE ?
SYS MASK ← 0
WAIT BIT ← 0
RETURN


EXT
SAVE REGISTERS
HALT TYPEWRITER
TO ERROR EXIT


PROGRAM
INT CDE ≥ 8
I ← INT CDE
PCHK [I] ← PCHK [I] +1
CC ← 3
RETURN
SAVE REGISTERS
TO ERROR EXIT

INTERRUPTS

6. The PL360/OS system

E. Satterthwaite

A PL360 system is available which is compatible with IBM System/360 Operating System (OS). The PL360/OS system is processed as a single OS job step; it consists of a core-resident linkage routine originally coded in OS Assembler Language and a set of PL360 system programs on a (logical) system tape. Among the latter set of programs is a core-resident job sequencing routine, which controls the processing of a batch of PL360 jobs as directed by the PL360 control cards described in chapter 2. In addition, OS Job Control Language (JCL) statements must precede the PL360 batch to control OS job sequencing and the association of PL360 logical input/output devices with OS data sets. Finally, the linkage routine must be supplied as an object module if it is not available in a system library.

Section 6.1 contains information about the use of the previously described PL360 system programs under OS. Section 6.2 is an introduction to the use of the Job Control Language as it pertains to the PL360/OS system. The internal organization of the system is described in section 6.3. Understanding of parts of these sections requires some familiarity with OS concepts and terminology. Furthermore, only the more elementary uses of OS job and data management facilities can be described in detail. The publication IBM Operating System/360: Concepts and Facilities (IBM Form C28-6535) contains an introduction to OS and further references.

6.1. PL360 Programming Under OS

6.1.1. Input/Output Considerations

OS data management services allow the problem programmer to code I/O requests in a manner which is relatively independent of physical device characteristics. Some of the I/O requests in the linkage routine have been so coded; details are provided in section 6.3. Supervisor I/O function statements are written in PL360 as described in section 3.1.2.4; however, the following restrictions are made:

a. READ

Reading any card with a "/*" in columns 1 and 2 will cause termination of the PL360 system. The message

PL360/OS TERMINATED BY OS DELIMITER

will be printed at the top of the next printer page.

b. Tape Functions

(1) Since physical tape marks have special significance to OS, logical tape marks are written as special records. Such records are 18 bytes long; the first 14 bytes are EO_{16} (corresponding to 0-2-8 punches). Since such records are recognized as tape marks and since the last four bytes are reserved for system pointers, such records should not be written using the WRITETAPE function.

(2) Whenever possible, use of the BSPREC and BSPTM functions should be avoided. Certain design goals require the use of the OS BSP instruction in performing these functions; BSP is very inefficiently implemented in current versions of OS (through Release 10).

(3) A variety of physical devices may be used as logical tape units. OS and/or these devices impose limits on the maximum record length which can be processed. These limits are:

Physical Device	Maximum Record Length (bytes)
2400-2402 tape	32760
2311 disk	3625
2314 disk	7188
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 PL360 job with an I/O error indication. A record length parameter is passed to each program in register 6. In addition to any user defined significance, this parameter is used by the PL360 compiler and system tape update programs as described in section 6.1.2. The value of the record length parameter may optionally be specified as a PL360/OS system parameter (see section 6.2.5.1); the default value is 3624.

c. WRITETIME

The printed time is real time. With some OS options, only part of that time was used by the PL360 system.

6.1.2. System Program Limitations

In addition to the input/output considerations above, certain restrictions are made on the use of the system programs.

6.1.2.1. The PL360 Compiler

The language processed is PL360 with the restrictions and extensions of section 3.1.2. 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 tape as multiple records, if necessary, to limit maximum record length to that specified by the record length parameter.

6.1.2.2. The System Tape Updating Program

Since the PL360/OS system linkage routine (monitor) is not included on the system tape, section 3.5.4 is not relevant. **$**LOAD and **$**MONITOR cards should not be used. Maximum record length on any new system tape generated will not exceed that specified by the record length parameter.

6.1.3. Program Execution Errors

Errors are processed as described in chapter 4 with the following exceptions:

a. The external interrupt facility is unavailable.

b. An end-of-job (EOJ) error is also caused by an attempt to read beyond the end of a data set on any logical tape.

c. The PSW displayed for input/output and EOJ errors has the form

OOOOOONNXXXXXXXX

where NN is the logical device number and XXXXXXXX specifies the location of the instruction following the one linking to the I/O routine.

Following the detection of errors, the system will attempt to proceed to the next PL360 job. Certain serious I/O errors cannot be accepted by OS and will terminate the PL360 system. In addition, the linkage routine, the job sequencing routine, and the problem program have identical memory protection keys, and the program reference table both shares this key and is addressed using register 14. Thus it is possible for a problem program error to cause PL360 system failure. Under OS options using memory protection, OS supervisory programs and user programs initiated by other OS jobs in the system will be protected from such modification.

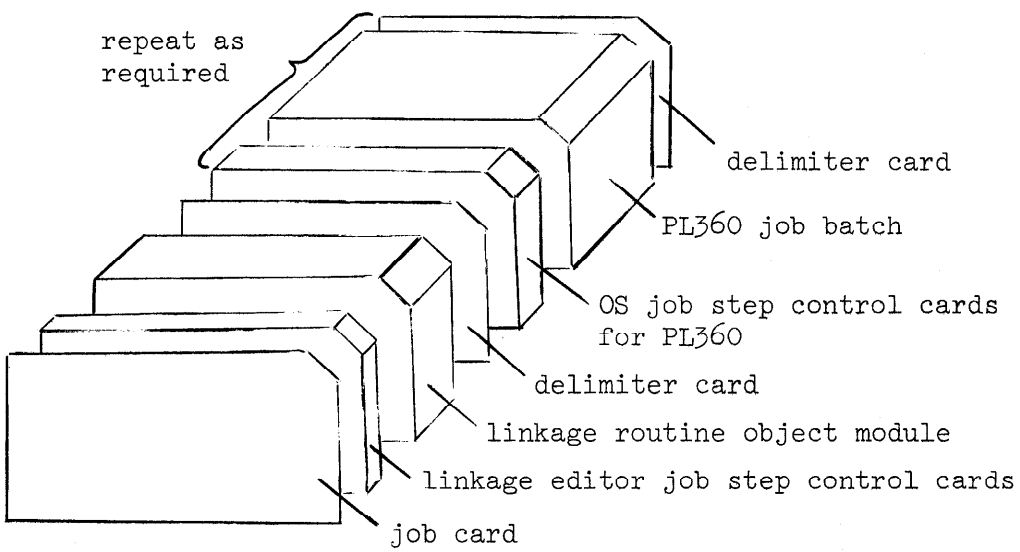
6.2. OS Control Statements for PL360

6.2.1. Introduction

In this section, the reader is assumed to be familiar with OS concepts and facilities and to have access to the publication IBM System/360 Operating System: Job Control Language, (IBM Form C28-6539). Use of the system with a card object module of the linkage routine and a full set of JCL statements will be described. At some installations, a catalogued procedure or load module may be available to reduce the number of non-PL360 cards; documentation of the use of such facilities is considered an installation responsibility.

6.2.2. PL360/OS Job Setup

A PL360/OS job consists of two or more job steps. In the first step, the OS linkage editor is used to produce a load module of the PL360 linkage routine. In each subsequent step, a batch of PL360 jobs is processed. For each such job step, DD cards are used to associate PL360 logical devices and OS data sets. Such associations are fixed during each job step but may be altered between job steps. The card deck organization required is shown schematically below:



The OS JCL statements required are described below. Information concerning the syntax and format of the cards containing these statements may be found in the IBM JCL manual (IBM Form C28-6539); the notation introduced in that manual is used in the following description. The delimiter card contains a "/*" in columns 1 and 2 . The PL360 job batch must contain the PL360 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.

6.2.3. The Job Card

This card must be prepared according to individual installation standards.

6.2.4. The Linkage Editor Job Step Control Cards

For most efficient resource utilization, the JCL statements for this step should be copied from those of the linkage editor step of the installation's standard catalogued procedure ASMFCLG, with the data set SYSLIN equated to SYSIN. A typical set of statements follows:

```
//LKED     EXEC  PGM=IEWL,PARM='NCAL'
//SYSLIN   DD    DDNAME=SYSIN
//SYSLMOD  DD    DSNAME=&GOSET(MAIN),UNIT=SYSDA,                    X
//               SPACE=(TRK,(20,10,1)),VOLUME=REF=SYS1.SCRTCH3,     X
//               DISP=(,PASS),DCB=(RECFM=U,BLKSIZE=3625),
//SYSUT1   DD    SPACE=(TRK,(20,10,1)),VOLUME=REF=SYS1.SCRTCH1
//SYSPRINT DD    DUMMY
//SYSIN    DD    *
```

6.2.5. The OS Job Step Control Cards for PL360

6.2.5.1. The EXEC Statement

Execution of the load module produced by the link edit step (named LKED) is specified by a statement of the form

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

An integer should be specified for blocksize; this parameter is optional (see section 6.1.1). The name of each job step within a job should be unique.

6.2.5.2. The DD Statements

DD statement names for the PL360/OS system should be of the form DEVICEn, where n is the number of the PL360 device to be associated with the described data set. The devices and associated logical characteristics supported by the standard PL360/OS system are as follows:

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

Data sets associated with devices 1 through 3 should consist of OS format F records with the following attributes:

Device Number	Logical Record Length (bytes)	Blocksize (bytes)
1	133 (1)	(2)
2	80	80
3	80	(2)

(1) includes an ASCII carriage control character supplied by the linkage routine

(2) specified by DD statement.

Data sets associated with devices 4 through 9 should consist of OS format U records; any logical tape marks to be processed must have been written by the PL360/OS system. DD statements for devices 1 through 4 are required; those for devices 5 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.

Section 2 of the IBM JCL manual (Form C28-6539) contains model DD statements for most common applications. The following notes should be considered a supplement to that section.

a. For DEVICE1 and DEVICE3, the DCB parameter is required. Appropriate values for the subparameters BLKSIZE and BUFNO must be specified. For all the other data sets, no DCB information is required.

b. If deferred mounting of magnetic tape volumes is requested, tapes need not be mounted until (and unless) referenced. With this option, the unit parameter has the form

UNIT=(address,,DEFER) .

Appropriate serial numbers should be supplied for unlabeled tapes, since they are used in mounting instructions directed to the operator.

6.2.5.3. Examples

The following examples illustrate appropriate JCL statements for the PL360 job step in the situations described.

a. PL360 programs are to be compiled and executed. The system is on unlabeled tape (reel identification GSG140); the scratch area is to be on disk; standard unit record options are desired.

```
//GO        EXEC   PGM=*.LKED.SYSLMOD
//DEVICE1   DD     SYSOUT=A,DCB=(BLKSIZE=133,BUFNO=2)
//DEVICE2   DD     DDNAME=SYSIN
//DEVICE3   DD     UNIT=SYSCP,DCB=(BLKSIZE=80,BUFNO=4)
//DEVICE4   DD     UNIT=(183,,DEFER),LABEL=(,NL),                    X
//                 VOLUME=SER=GSG140,DISP=(OLD,KEEP)
//DEVICE5   DD     SPACE=(TRK,(20,10)),VOLUME=REF=SYS1.SCRTCH1
//SYSIN     DD     *
```

b. In the first job step, named G01, a system program on unlabeled tape (reel identification SC1278) is to be compiled; the compiled program is to be used in a system update to a new tape reel. In the next job step, named G02, the new system tape is to be used, and logical card punch output is to be blocked and written on magnetic tape.

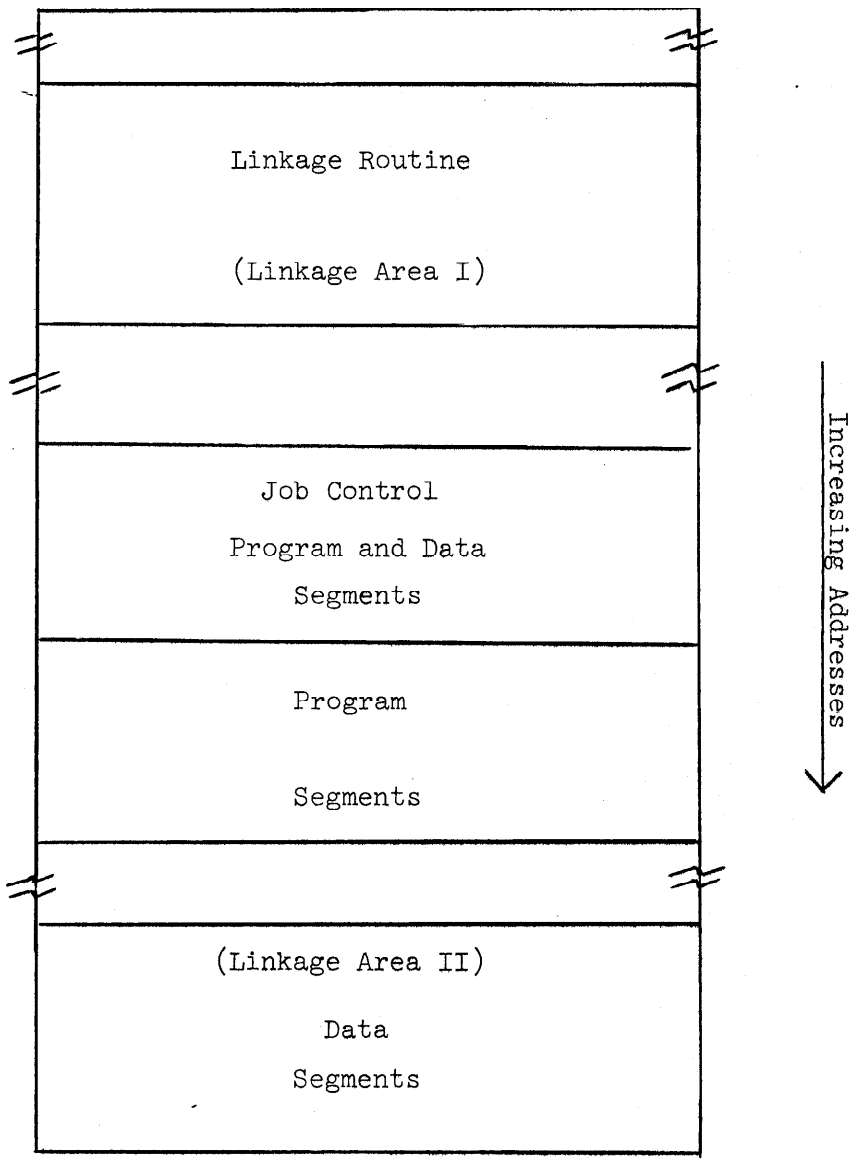
```
//G01       EXEC   PGM=*.LKED.SYSLMOD,PARM=32760
//DEVICE1   DD     SYSOUT=A,DCB=(BLKSIZE=133,BUFNO=2)
//DEVICE2   DD     DDNAME=SYSIN
//DEVICE3   DD     UNIT=SYSCP,DCB=(BLKSIZE=80,BUFNO=2)
//DEVICE4   DD     UNIT=(183,,DEFER),LABLE=(,NL),                    X
//                 VOLUME=SER=GSG140,DISP=(OLD,KEEP)
//DEVICE5   DD     UNIT=(2400,,DEFER),LABEL=(,NL)
//DEVICE7   DD     UNIT=(184,,DEFER),LABEL=(,NL),                    X
//                 VOLUME=SER=SC1278,DISP=(OLD,KEEP)
//DEVICE9   DD     UNIT=(283,,DEFER),LABEL=(,NL),                    X
//                 VOLUME=SER=GSG141,DISP=(NEW,PASS)
//SYSIN     DD     *

//G02       EXEC   PGM=*.LKED.SYSLMOD
//DEVICE1   DD     SYSOUT=A,DCB=(BLKSIZE=133,BUFNO=2)
//DEVICE2   DD     DDNAME=SYSIN
//DEVICE3   DD     UNIT=(282,,DEFER),LABEL=(,NL),                    X
```

```
//                VOLUME=SER=GSG201,DCB=(BLKSIZE=240,BUFNO=1),      X
//                DISP=(NEW,KEEP)
//DEVICE4  DD     DSNAME=*.G01.DEVICE9,DISP=(OLD,KEEP)
//DEVICE5  DD     SPACE=(TRK,(40,10)),VOLUME=REF=SYS1.SCRTCH1
//SYSIN    DD     *
```

6.3. System Organization

6.3.1. Storage Organization

The PL360/OS system consists of a linkage routine, coded in OS Assembler Language, and a set of system programs, coded in PL360. Among the latter is a job sequencing routine. That routine and the linkage routine are permanently core-resident; other PL360 system and user programs are loaded as directed by control cards. During execution of a PL360 job, storage is organized as indicated schematically in the following diagram:



The linkage routine occupies about 5000_{10} bytes of storage. The job sequencing, program and data segments, and the free storage area occupy a block of storage obtained by a GETMAIN instruction. The length (in bytes) of the block lies between the values of COREMIN and COREMAX defined in the linkage routine source code. In the standard system, these have values of 65536 and 131072 respectively.

The two linkage areas contain save areas and identical copies of the program reference table, which contains the segment base addresses. Area I is used by the job sequencing routines and is filled by the loader programs. Area II is used by problem programs; the reference table of Area I is copied into Area II at the completion of loading. The contents of register 14 always address the base of the linkage area in use; the contents of register 15 normally address the base of the program segment being executed.

The linkage areas consist of 92 full words each, used as follows:

Displacement (Bytes)	USE
+ 0 – + 252	user program segment base addresses (segments 0 through 63)
+ 256 – + 316	job sequencing segment base addresses (segments 64 through 79)
+ 320	linkage routine entry vector address
+ 324 – + 352	linkage routine register save area
+ 356	reserved
+ 360	reserved
+ 364	return address (to job sequencing)

6.3.2. The Linkage Routine

6.3.2.1. Linkage Conventions

Supervisor function statements in PL360 generate machine code of the following form (refer to section 6.3.1):

```
L     15,320(14)      load entry vector address
BAL   15,12*n(15)     link to n'th entry point
L     15,4*m(14)      reestablish addressing
```

where it is assumed that the statement occurs in segment m . Elements of the entry vector have the following form:

```
STM   12,3,324(14)     save registers
L     12,320(14)       establish linkage routine addressability
B     routine          branch to service routine.
```

The return sequence has the form:

```
CLI   *+1,0            set condition code to 0
LM    12,3,324(14)     restore registers
BR    15               return
```

For certain services specified in section 3.1.2.4, instructions to set the condition code appropriately replace the first instruction. In addition, most routines require a separate save and restore of register 14 in a location addressable through register 12 .

6.3.2.2. Unit Record Input/Output

I/O to the logical printer, card reader, and card punch is performed using the OS queued sequential access method with move-mode GET and PUT logic. Automatic buffering is supplied by OS. The card reader uses a block size of 80 bytes and two buffers; blocking and buffering information for the other logical devices is provided on the corresponding DD cards. The logical record length for the card punch is 80 bytes; for the printer, it is 133 bytes and an ASCII carriage control code is prefixed to each record by the linkage routine. Following a skip to channel 1, the next skip code is supplied after LINESMAX lines have been printed or a PAGE statement is executed. LINESMAX is defined in the linkage routine source code; the standard value is 60 .

6.3.2.3. Tape Input/Output

I/O to the logical tape units is performed using the OS basic sequential access method with READ, WRITE, and BSP logic. Records are considered to have the OS undefined format; those shorter than 18 bytes are automatically padded to that length. No buffering is provided, and by issuing a CHECK instruction, the linkage routine

assures that all I/O requests are completed before returning. Tape marks are written as special 18 byte records, the first fourteen bytes of which contain EO_{16} (corresponding to an 0-2-8 punch). For each logical device, a count (NBLOCKS) is maintained of the number of records written or read following either the load point or the inter-record gap which precedes the last tape mark record. When a tape mark record is written, the current count value is recorded in the last four bytes of that record, and the count is reset.

6.3.2.4. Other Linkage Routine Services

A portion of the linkage routine is used for system initialization. Storage is obtained by a GETMAIN instruction, linkage to capture program interruptions is established by a SPIE instruction, and logical devices 1 through 4 are opened by an OPEN instruction. Other devices are automatically opened the first time they are referenced. Upon system termination, storage is freed and logical devices 1 through 5 are closed. Other devices will automatically be closed by the system if necessary. The card and tape loaders are also included in the linkage routine; they are coded to be used as PL360 procedures.

6.3.3. The Job Sequencing Routine

The job sequencing routine is a minor adaptation of the corresponding routine for the stand-alone system. Status switching and access to the device table have been deleted. The program logic is described in section 5.2.

Appendix: Conversion of 026-punched cards

The system accepts the following two control cards anywhere in the input deck:

±026 and ±029

These control cards do not give rise to end-of-file indications, but cause the **READ** routine to perform a character translation on subsequently read cards, or to omit the translation respectively.

The translation, caused by ±026, permits the use of cards punched on Stanford's extended 026 keypunch equipment. The translation is specified by the following table. Note that letters and digits do not undergo translation.

holes	026	029	hex.
12-3-8	.	.	4B
12-6-8	<	<	4C
0-4-8	(	(	4D
12-5-8	[	(	4D
12	+	+	4E
0-6-8	<	|	4F
12-0		&	50
11-3-8	$	$	5B
11-4-8	*	*	5C
12-4-8	)	)	5D
11-5-8	]	)	5D
11-6-8	;	;	5E
6-8	x	¬	5F
11	-	-	60
0-1	/	/	61
0-3-8	,	,	6B
11-7-8		%	6C
0-5-8	←		6D
11-0		≥	6E
5-8	:	:	7A
12-7-8		#	7B
0-7-8		@	7C
7-8		'	7D
3-8	=	=	7E
4-8	'	"	7F

References

[1] N. Wirth, "A programming language for the 360 computers", Technical Report CS 53, Stanford University, June 1967.

[2] N. Wirth and H. Weber, "EVLER, A generalization of Algol, and its formal definition", Part 1, Comm ACM 9/1, pp. 13-23.

[3] IBM System/360 principles of operation, IBM Sys. Ref. Lib. A22-6821-2.