T H E   U N I V E R S I T Y   O F   M I C H I G A N

Memorandum 17

PDP-8 PROGRAM RELOCATION: CONCEPTS AND FACILITIES

D. L. Mills
and
V. M. Powers

CONCOMP: Research in Conversational Use of Computers
F. H. Westervelt, Project Director
ORA Project 07449

supported by:

ADVANCED RESEARCH PROJECTS AGENCY
DEPARTMENT OF DEFENSE
WASHINGTON, D.C.

CONTRACT NO. DA-49-083 OSA-3050
ARPA ORDER NO. 716

administered through:

OFFICE OF RESEARCH ADMINISTRATION ANN ARBOR

February 1968

TABLE OF CONTENTS

PDP-8 PROGRAM RELOCATION: CONCEPTS AND FACILITIES

This memorandum describes a method for segmenting PDP-8 programs for the purpose of facilitating program maintenance and residence in MTS (Michigan Terminal System) files. The method provides for program storage on a page-relocatable basis with relocation information contiguous to but not necessarily integral with text information. Linkages between separately assembled program segments are provided in a form very similar to those used in IBM System/360 systems.

Currently available utilities within MTS provide assembly and link-editing facilities, using programs stored either as punched card decks or in MTS files. Utilities are also included for the purpose of paper tape transcription either in PAL-compatible format or in a special format useful for dynamic loading via a data link to a remote machine. In addition to these MTS utilities, two relocating PDP-8 loaders are available which operate using the special dynamic-loading format. Each of these programs occupies one dedicated page of PDP-8 memory and operates in a multi-core-bank environment. One of these programs is designed to operate as a stand-alone utility, while the other is designed to operate within the RAMP system.*

* Mills, D. RAMP: A PDP-8 Multiprogramming System for Real-Time Device Control, Concomp Project Memorandum 5, University of Michigan, Ann Arbor, May 1967, 24 pp.

Mills, D. I/O Extensions to RAMP, Concomp Project Memorandum 11, University of Michigan, Ann Arbor, October 1967, 32 pp.

1. Basic Concepts

A control section (CSECT) is a contiguous block of assembled instructions and/or data. Its length may be from zero to 4096 words, and its starting address may be at any address in any core bank. However, its ending address must be in the same core bank. A CSECT containing instructions most likely (but not necessarily) will begin on a page boundary and extend not over a page in length. A CSECT containing only data or indirectly referenced storage most likely will not be constrained in this manner. A module is an assembly containing one or more CSECTs or the output of a link-edit procedure (described below) in which a number of assemblies are involved. If any of the constituent assemblies of a module contain external references to symbols defined in other assemblies, then the CSECTs defining these symbols are assumed to be present in the module. Although a module may or may not represent a complete operable program, it does represent the smallest program structure which can be loaded by name from system residence, either locally or via a data set.

Each CSECT in a module is assumed to be rather arbitrarily located relative to other CSECTs in the machine. Special external symbol dictionary (ESD) cards produced by the MTS assembler define the origin, length, and optionally the name of each CSECT. Linkages between these CSECTs are maintained in the usual fashion using address constants (adcons) and indirect-accesses. Relocation information, provided only for the adcons themselves, is produced by the MTS assembler in the form of relocation dictionary (RLD) cards.

If any CSECT either defines an entry point (value) of an externally referable symbol or references such a symbol presumably defined in another assembly, then the MTS assembler produces a special ESD card. Three types of ESD cards can be recognized: those produced by the assembler in response to control-section definition (CSECT pseudop), those in response to an entry point declaration (ENTRY pseudop), and finally those in response to an external symbol declaration (EXTRN pseudop).

Facilities are included for presetting the initial loading address for each core bank (FIELD pseudop) and for presetting the point at which the program will be started after loading is completed (END pseudop). Each of these pseudops is fully described below under "Assembler."

1.1 Object File Structure

Each assembly produces a deck of column-binary cards or of file images of such cards as appropriate. The order of production of the various cards is as follows: (a) ESD cards containing the CSECT definitions, entry point declarations and external symbol declarations, (b) text (TXT) cards containing the assembled instructions and data, (c) RLD cards containing the relocation information for the TXT cards, and finally (d) an END card terminating the object deck and optionally containing the starting address of the program. Appendix A describes the card and tape formats of the various types of cards.

A module is constructed of one or more such object decks placed one after the other. If any of these decks contain external symbol references then the entire module deck can be processed by the link editor (see below) to produce a deck which is free of such references and in the form of a single object deck. The edited deck can then be loaded directly by a simple PDP-8-resident relocating loader described below or used as a component of another module.

It is important to point out that if none of the relocation control pseudops (CSECT, ENTRY, EXTRN) are used in an assembly, the object deck produced is identical to that produced by either the 8SS (7090) or 8ASS (360) assemblers, with the exception that the card formats are slightly different (see Appendix A). An MTS utility is available which transcribes such decks into PAL-compatible paper tape which can be loaded using conventional DEC programs.

In addition, if the ESD and RLD cards are removed from a relocatable assembly object deck, then the remaining TXT and END cards can still be processed directly to produce PAL-compatible binary tape. It is, however, not in general possible to extend this feature to include object decks produced by a link edit process.

1.2 Loading Operations

The PDP-8 resident loader operates on modules as data and produces a loaded executable program. In the simplest case, only a single module is loaded in the machine and the machine is assumed to be dedicated during the loading process. It is possible during this simple loading process to scatter the various CSECTs of the module throughout the memory of the machine, with the restriction that CSECTs containing instructions are loaded at the proper displacements relative to page boundaries. Such a process would be useful in loading debugging and utility systems over host programs already loaded in memory. If linkages between modules loaded by this simple process are required, transfer vectors of some type must be provided at fixed locations in memory known to all loaded modules.

A more interesting and potentially useful case occurs in connection with multiprogramming systems resident in machines used in real-time control environments. Such systems are assumed to provide a uniform input/output device interface, a task management subsystem, a system-residence interface and a dynamic memory allocation subsystem. Systems such as these have been described in recent reports published by the Concomp Project.*

---

* See footnote on p. 1.

Object module storage in such a system may reside either on direct-access devices or in the form of paper tapes or cards. For this purpose, a data link to MTS will be considered a direct-access device. A module can be loaded from a direct-access device by name, in which case some sort of library management subsystem is postulated which provides the data representing the module in response to a command specifying its name. A module may also be loaded from a paper tape or card reader, in which case the machine operator must load the reader with the specified module, perhaps on command from the operator's console.

Once the module is acquired, the loading process continues as follows: first, if the loader itself is not resident in memory, a page of memory is seized and the loader is read in. The loader itself is then called as a task and its operations proceed in parallel with others in the system. Next, the loader seizes a page of memory for scratch uses. This page may contain portions of the loader together with a table of relocation factors for the various CSECTs loaded. Next, the loader reads all the ESD records of the module, seizes the necessary memory areas from the storage management subsystem, and makes entries in the relocation factor table. In addition, all memory areas are chained so that they can be reclaimed if the module is deleted from the system.

Following this setup procedure, the text of the module is read into the appropriate memory areas. Transmission from byte-oriented systems residence devices is assembled byte by byte and stored in memory. Transmission from word-oriented devices capable of data-break operations proceed directly to memory without byte assembly. (Note that the relocation factor table can be easily mapped into a channel-command list.)

Following the loading of all module text information, the RLD information is read and processed by the loader. This process develops all the linkages between the scatter-loaded CSECTs of the module. After this process the loader returns its own storage to the allocatable storage pool and returns to its invoking task. The loaded module is now ready for execution.

2. Functional System Components

The relocation procedure described above is supported by three principal system programs: the assembler, the link editor, and the loader. At present both the assembler and link editor are resident in library files in the MTS system and may be run from any terminal attached to that system. The loader is of course resident within the PDP-8 itself and is written in conventional machine language. Although many versions of the loader are immediately apparent, the particular stand-alone version supported at this time resides in any core bank at locations 7400-7577 inclusive. A scratch table is also maintained by the loader in each core bank at location 7276-7377 inclusive. The following sections describe the operation of these three programs.

2.1 Relocating Assembler

The assembler in MTS library file *8ASSR assembles PDP-8 code and produces either absolute or relocatable modules in the form of column binary card images, according to the format described in Appendix A. The basic input format and operation are essentially the same as for the PDP-8 assembler, *8ASS, except for the new procedure calls, the CSECT definitions, and the new structure of the object deck. For example, the relocatable assembler might be invoked by the following command:

```
"$RUN *8ASSR; 1=SOURCE 2=*8OPS 6=-T 8=*SINK* SPUNCH=CARDS"
```

An assembly is a program segment consisting of a number of CSECTs, each of which has a different integer, from 0 to 511, as CSECT identification (CSID). Each CSECT corresponds to a page-relocatable segment of PDP-8 code, or a labeled value (external symbol or entry point). The CSECTs may each be relocated differently by the loader. A CSID of 0 has a special connotation—an absolute (non-relocatable) CSECT. Thus, an assembly of one CSECT whose CSID is 0 is an absolute assembly.

CSECTs are defined by the programmer through use of the procedure calls CSECT, EXTRN, and ENTRY, as described below. In address and length calculations, an address value is considered to be a 12-bit integer modulo 4096.

1. CSECT n - Primarily used to define a new CSECT. A CSECT is defined with CSID=n. The address of the CSECT is either

a) the address of the next text word (machine instruction),

b) the value of the next ORG procedure call, or

c) the value of the ILC before the next DS procedure call,

whichever comes first. If the operand field n is empty, the CSID assigned is one more (modulo 512) than the last CSID assigned, or 0, if this is the first CSECT call. The length of the CSECT is computed by subtracting the address of the CSECT from the value of the ILC when the next CSECT procedure call appears. If a CSECT with the same CSID has been defined previously, the new section of code is treated as an extension of the previous CSECT, whose address is the lowest of the two address definitions and whose length is the largest of the two length definitions. The CSECT name is 8 blanks.

2. ENTRY m - A CSECT is defined whose CSID is sequentially next, whose length is 0, and whose address is the current value of the ILC. The CSECT name m is the first 8 characters of the operand field, padded with trailing blanks.

3. EXTRN m - A CSECT is defined whose CSID is sequentially next, whose length and address are zero, and whose name m is the first 8 characters of the operand field, padded with trailing blanks. The name is also defined into the assembler's symbol table as a relocatable operand (a variable) with

value 0, for use in adcon expressions.

The assembler passes to the link editor and the loader the CSECT structure, as defined by the CSECT, EXTRN, and ENTRY procedure calls, by means of ESD card images. There are three types of ESD cards: CSECT cards, external symbol cards, and entry point cards. Each card lists its unique CSID and the assembled address, length, and name of the CSECT. The ESD (CSECT) cards may be used to allocate storage, the ESD (EXTRN) cards to locate references to external symbols, and the ESD (ENTRY) cards to define external symbols. The assembler lists a table of the CSECTs, and punches the ESD cards, in the order defined, between Pass I and Pass II. Among the ESD cards, EXTRN cards defining symbols appearing in a certain control section are presumed to occur immediately following the CSECT card for that section. There are no restrictions other than these constraining the order of the cards within an object deck.

During Pass II, text (TXT) cards are produced. Each TXT card represents a contiguous block of machine words within a single CSECT. The TXT card lists the CSID of that portion of text included on the card, the assembled address of the first text word, and a number of machine words to be stored in consecutive locations. The loader loads these words into consecutive locations without change.

The information necessary to relocate adcons is provided by the assembler at the end of Pass II by means of a relocation dictionary (RLD cards). These cards are produced by scanning the reference information. During Pass II, each time a variable occurs in an opcode or operand expression, a reference item is added to a chained list which is appended to the assembler's symbol table entry for that variable. A reference item is a list entry which contains the value of the ILC corresponding to the occurrence of the variable. In the absolute PDP-8 assembler, *8ASS, a reference item contains only that value. The list of values from each variable's reference chain is printed out, in the order entered, as the information in the reference table which occurs at the end of the assembly listing.

The reference items in the relocatable assembler *8ASSR contain two more pieces of information: a set of flags and a CSID. One of the flags, the relocation flag, is set if the variable is a "relocatable operand" (a label or an external symbol) appearing as an argument to a "relocatable opcode" (the procedure call DC is the only one so defined at present); that is, if the variable is a label or external symbol appearing in an adcon expression. If the relocation flag is set then another flag indicates whether the value of the variable was added to or subtracted from the expression. In this case, the CSID in the reference item is set to be the "current CSID"; the CSID of the adcon.

Thus, in Pass II, the assembler builds a list for each label and external symbol which contains, for each adcon using that variable, the CSID and address of that occurrence, and an indication of whether the appropriate relocation factor should be added to or subtracted from that adcon value when it is loaded. This is the relocation information which is passed to the loader on RLD cards which follow all the TXT cards of an assembly.

Each RLD card contains a CSID, called the position ID, and a list of relocation items. The position ID is the CSID of a number of variables or external symbols which appear in adcons. Each two-word relocation item contains a flag to indicate positive or negative relocation and the CSID and address of one of those adcons.

The special symbol, *, is treated differently. Its value is the value of the program counter at each occurrence of the symbol, and its occurrences are listed in the reference items. The position ID on every RLD card which refers to occurrences of *, however, is the same as the CSID of the adcon. Thus, the value of * in an adcon can be relocated by the "current" relocation factor.

After the RLD cards, an END card is produced. It contains the CSID and address of the starting location, if a label appears in the operand field, or zeros, if none occurs.

The procedure call, FIELD n, causes a FIELD card to be produced among the TXT cards at each occurrence in Pass II.

2.2 <u>Link Editor</u>

The link editor is an assembly-language program written for MTS residence. It processes object decks produced by the assembler to produce a load module which can be input to the PDP-8 loader. The object program resides in an MTS file called *8LINK, obtains its source from SCARDS, produces its output on SPUNCH, lists dictionaries on SPRINT, and produces error diagnostics on SERCOM.

The input file to the link editor consists of one or more object decks in the order of loading in the PDP-8. Each of these decks consists of a sequence of ESD cards followed by TXT cards followed by RLD cards followed by an END card. A FIELD card giving the initial loading address for a core bank is expected to precede the first occurrence of an object deck to be loaded in that core bank. Only one of these cards is required for each core bank, but more can be used to force the loading of control sections at arbitrary memory locations.

Control section numbers (CSIDs) (other than CSID zero) within any one assembly need be unique only within that assembly and may occur in duplicate with CSIDs in other assemblies. The link editor will resequence the CSIDs so that they are unique within the output module. A CSID of zero indicates an absolute address and may be used for interconnection among modules. Control section numbers 000-077 are assumed to reside in core bank zero, 100-177 in bank one, and so forth to 700-777 which are assumed to reside in bank seven. This convention represents only an agreement between the link editor and the loader and can be readily changed.

The link editing process consists of three phases, performed in sequence. First, the collection of object modules is read, checksummed, and stored in a temporary sequential file. During this phase all cards in an object deck except the TXT and RLD cards are placed in an internal table called CSTAB. One of these tables is built for each object deck in the module and indexed by pointers in another internal table called MDTAB.

Second, the entire MDTAB-CSTAB structure is edited in such a way that:

a) nonzero CSIDs of CSECT and ENTRY cards are replaced by sequentially increasing integers (starting at one for each core bank) and the cards are punched with their new CSIDs.

b) FIELD cards are punched as-is.

c) Each time a CSECT or ENTRY card is punched, the MDTAB-CSTAB structure is searched for occurrences of EXTRN cards with the same name as that of the CSECT or ENTRY card. If one or more such cards are found, a field in the EXTRN card CSTAB entry is set to be the same CSID of the CSECT or ENTRY card.

d) A dictionary consisting of the processed cards and their presumed loading addresses is listed.

Finally, the temporary file containing the TXT and RLD cards is rewound and rescanned. During this scan each CSID found on a card is looked up for a match in the CSTAB corresponding to the particular object deck in which it resides. When such a match is found, the card is punched with new CSIDs found from the CSTAB entries (see (a) and (c) above). An END card is punched to terminate this process. The CSID and starting address punched on the END card is determined as that appearing in the last END card in the input module with a nonzero CSID; or, if none was present, a zero.

The output module from the link editor may be stored in three forms, depending upon end-use. If the module will be saved in the system or punched on cards, then it should be saved in column-binary card format. Such a format includes checksum and length information on the card and may include sequence identification. If the module will be transmitted to a remote machine over a data link or saved on paper tape for later use with the relocating loader described below, then it should be saved in relocatable-tape format. Such a format is in essentially a six-bit byte size with control codes designated by a one in the seventh bit. No use is made of the eighth bit whatsoever, so that vertical parity options can be used if desired. If the module is to be loaded in the PDP-8 using conventional DEC systems, then it should be punched in PAL format on paper tape. In this case relocation information is of course lost. At present it is impossible to punch a link editor output module in PAL format if the input which produced that module contained references to external symbols.

2.3 PDP-8 Loader

The relocating loader resides in any core bank in the PDP-8. It occupies exactly one page of memory and is so written as to be page relocatable with the exception of certain pointers to a region of memory in which the relocation tables are kept during the loading process. As distributed, the loader occupies locations 7400-7577, and the relocation tables occupy locations 7276-7377. One of these tables is kept in each core bank in the same storage locations, so that these locations should not contain loaded text. The table storage can of course be used as scratch area once the loaded program begins execution. Also, as distributed, the loader is assumed to read input tapes from the high-speed reader. Obvious patches or reassembly can change these parameters.

The loading process consists of three phases. The first phase involves storage allocation for the various CSECTs in the memory of the machine. This is accomplished while reading the ESD and FIELD records from the input tape. To facilitate this two words defining the origin and length of the current control section are kept as the first two words of the relocation table in each core bank. The initial value for the length is zero, and the origin is specified in the FIELD record for a core bank which should precede any CSECT or ENTRY records for that core bank. The FIELD record also causes the relocation factor for CSID zero to be set to zero. This can be changed using a CSECT 0 card (see Assembler). Following the FIELD records, space is allocated in the various core banks as the CSECT records are read. Each CSID is used as an index into the relocation table, which is 64 entries long for each core bank, and the relocation factors are computed for each CSECT and entry record and stored in the appropriate entries. A new FIELD card resets the origin of the next control section.

In the second phase, the TXT records are loaded into memory using the relocation factors defined by the preceding CSECT and ENTRY records. Except for the actual machine loading address, which is obtained from the origin address included in the record and the appropriate relocation factor, the text is not modified in any way during this phase.

The third and last phase involves the relocation of adcons in the loaded text. This is performed while reading the RLD records which follow the TXT records. Each RLD record references a CSID, the relocation factor of which is used to modify those adcons which occur at the locations given as the RLD items. Each of these items indicates a CSID and assembled address at which the adcon resides, together with a flag which indicates whether the relocation factor is to be added to or subtracted from the assembled value.

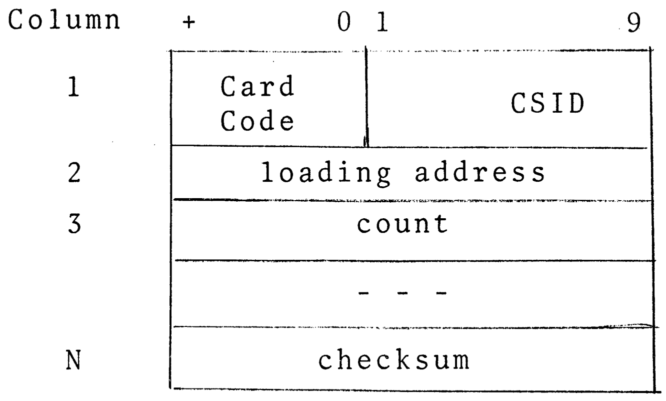
Following these three phases, a transfer to the loaded program is effected if so specified on the END record. Check-sum operations are separately controlled by special records which may occur in multiple anywhere in the tape before the END record.

APPENDIX A

CARD FORMATS

Eight types of cards are recognized by the various system component programs. The formats and uses of each are summarized below. Each card is assumed to be punched in column-binary format with each column representing a 12-bit word in the PDP-8 or two 6-bit bytes in MTS files. The order of transmission of the card is from the + row to the 9 row and from column 1 to column 80 in that order. In MTS files the high-order two bits of each byte are set to zeros.

All cards are built about the following format:



Column 1:

The card code is a three-bit field (+ through 0 rows) which identifies the card type according to the following code:

0 - (special—see below)	4 - ESD (CSECT)
1 - TXT	5 - ESD (ENTRY)
2 - END	6 - ESD (EXTRN)
3 - FIELD	7 - RLD

The three high-order bits of the CSID (1 through 3 rows) are used by convention to identify the core bank to which this card applies.

The six low-order bits of the CSID (4 through 9 rows) are used to identify the control section within a core bank to which this card applies.

Column 2:

The entire column contains a 12-bit loading address.

Column 3:

The entire column contains a 12-bit count of all remaining columns in the card except the checksum column (N-4).

Columns 4 through N-1:

Text coded for the particular card type is stored in these columns, which may contain either the name of an external symbol, a control section length, loaded text, or relocation information.

Column N:

The entire column contains the sum modulo 4096 of all columns on the card except the checksum itself.

The text information of the various cards is coded as follows, depending upon card code:

Code 0 (special):

Code reserved for future goodies. An all-zero (blank) card indicates that the checksum of the associated paper-tape image should be punched and reinitialized (see Appendix B: "Paper Tape Formats").

Code 1 (TXT):

Text information is punched exactly as assembled.

Code 2 (END):

Contains an optional four-column symbol (eight bytes) punched in "trimmed EBCDIC" consisting of the low-order six bits of each character.

Code 3 (FIELD):

The text portion is not used.

Code 4 (CSECT), Code 5 (ENTRY), Code 6 (EXTRN):

Contains two items, the first being the CSECT length (zero in ENTRY and EXTRN cards), and the second being a four-column symbol (eight bytes punched in "trimmed EBCDIC") consisting of the low-order six bits of each character.

Code 7 (RLD):

Contains a number of two-column (four-byte) RLD items, each of which is coded as follows:

```
                  +         0 1              9
                  ┌──────────┬────────────────┐
       Column N   │  flags   │      RLID      │
                  ├──────────┴────────────────┤
        N + 1     │     assembled address     │
                  └───────────────────────────┘
```

The RLID is coded exactly like the CSID entry in column 1 of the card (see above). The flags at present include only one bit (bit position 1) which,if set,indicates a negative rather than a positive relocation.

APPENDIX B

PAPER TAPE FORMATS

The relocating loader expects as input a tape transcribed in a simple fashion from an object deck structured as described above under "Card Formats." The transcription process is as follows:

a) Remove the count and checksum columns (two bytes each) and condense the text to eliminate these bytes.

b) Force a (01) as the high-order two bits of the first byte representing the card.

c) Punch the resultant record on paper tape with leader and trailer as desired.

d) Maintain a checksum, initially zero, of all bytes punched. Punch and reinitialize this checksum when a checksum card is found. (A checksum card is one with all bytes zero.)