**CP/M‑M3 Operating System**

**Programmer's Guide**

| **When** | **Who** | **What** |
| --- | --- | --- |
| 2013-12-30 | Roger Ivie rivie@ridgenet.net | Began original version. |

Contents

[1. INTRODUCTION TO CP/M‑M3 1](#_Toc376641096)

[1.1. CP/M‑M3 Architecture 1](#_Toc376641097)

[1.2. Transient Programs 2](#_Toc376641098)

[1.3. File System Access 2](#_Toc376641099)

[1.4. Programming Tools and Commands 2](#_Toc376641100)

[1.5. File Specification 2](#_Toc376641101)

[1.6. Wildcards 3](#_Toc376641102)

[1.7. Terminology 4](#_Toc376641103)

[2. DATA STRUCTURES 5](#_Toc376641104)

[2.1. The Base Page: cpm\_basepage\_t 5](#_Toc376641105)

[2.1.1. Initialization of the Base Page 7](#_Toc376641106)

[3. THE CCP AND TRANSIENT PROGRAMS 9](#_Toc376641107)

[3.1. CCP Built-in and Transient Commands 9](#_Toc376641108)

[3.2. Loading a Program in Memory 9](#_Toc376641109)

[3.3. Base Page Initialization by the CCP 9](#_Toc376641110)

[Table 1 Program Modules in the CP/M Operating System 1](#_Toc376641111)

[Table 2 Delimiter Characters 3](#_Toc376641112)

[Table 3 CP/M‑M3 Terminology 4](#_Toc376641113)

[Table 4 CP/M‑M3 Programmer's Guide Conventions 4](#_Toc376641114)

[Figure 1 Declaration of the Base Page 5](#_Toc376641115)

[Figure 2 The tail Field of the Base Page 7](#_Toc376641116)

[Figure 3 Base Page Fields Initialized by the BDOS 7](#_Toc376641117)

[Figure 4 Base Page Fields Initialized by the CCP 7](#_Toc376641118)

# INTRODUCTION TO CP/M‑M3

CP/M‑M3 is a port of CP/M‑68K to the ARM Cortex‑M3 architecture. The CCP and BDOS portions are built from the DRI source code for CP/M‑68K version 1.2. These DRI sources are combined with new glue and BIOS sources to build a system capable of running on the Cortex‑M3.

Since most of the operating system is built from unmodified DRI sources, most of the CP/M‑68K documentation is applicable to CP/M‑M3. This manual is derived from the CP/M‑68K Programmer's Manual.

Nevertheless, there are some significant differences between CP/M‑68K and CP/M‑M3. Some things are simplified and some things are enhanced. Programmers familiar with CP/M‑68K are encouraged to keep a wary eye out for differences.

Like CP/M‑68K, CP/M‑M3 contains most of the facilities of other CP/M systems with additional facilities required to address the larger memories available to the Cortex‑M3 processor. The CP/M‑M3 file system is upward compatible with CP/M‑80 version 2.2 and CP/M‑86 version 1.1. The file structure supports up to sixteen drives with up to 512 megabytes on each drive and a maximum file size of 32 megabytes.

## CP/M‑M3 Architecture

Like CP/M‑80, the CP/M‑M3 operating system resides within reserved tracks of the system disk. In the reference system, CP/M‑M3 resides within the internal flash of the LM3S9D92 processor, treating that flash as a disk; from the perspective of CP/M‑M3, resetting the system causes the processor to magically execute the bootstrap. The reserved tracks cover the bootstrap and operating system, with the remainder of the flash containing files. The bootstrap of the reference system copies CP/M to the processor's RAM and the system runs from that RAM. However, other configurations are possible.

The CP/M‑M3 operating system consists of the following modules:

| **Module** | **Mnemonic** | **Description** |
| --- | --- | --- |
| Console Command Processor | CCP | User interface that parses the user command line. |
| Basic Disk Operating System | BDOS | Provides functions that access the file system. |
| Glue | n/a | Adapts the CP/M system to a specific processor architecture. |
| Basic I/O System | BIOS | Provides the functions that interface peripheral device drivers for I/O processing and support a specific hardware platform. |

Table Program Modules in the CP/M Operating System

The reference system contains a modular BIOS that provides a regular interface to device drivers for devices categorized as consoles and disks. It is possible to support a wide variety of hardware without modifying the core of the reference BIOS. The modular BIOS provides some unique functions that are discussed in section ???.

The sizes of the CCP and BDOS modules depend upon the compiler and architecture involved, but should be largely fixed for any given release. The size of the glue varies depending upon the needs of the processor architecture. The size of the BIOS varies depending upon the needs of the specific hardware platform.

The regions of memory from which the operating system runs and into which user programs are loaded are determined when the operating system is built; they may vary depending upon the needs of the platform involved. User programs are tagged with the processor architecture upon which they expect to run and with the address of the region in which they expect to run; this prevents the system from attempting to run a user program that is not appropriate for the system configuration.

All CP/M‑M3 modules remain resident in memory. The CCP cannot be used as a data area subsequent to transient program load.

## Transient Programs

The CP/M‑M3 operating system provides a contiguously addressed region of memory into which user programs are loaded. This region is called the Transient Program Area (TPA). CP/M-M3 loads executable files, called command files, from disk to the TPA. These commands are called transient commands or transient programs because they temporarily reside in memory, rather than being permanently resident in memory and configured into CP/M. The format of a command file is described in section ???.

## File System Access

Programs access the services provided by CP/M‑M3 by invoking BDOS and BIOS functions. Section ??? describes the BDOS functions in detail. The BDOS provides a mechanism to locate and call BIOS functions, but those functions are beyond the scope of this manual. BIOS functions are discussed in ???. The modular BIOS supplied with the reference system provides a handful of additional functions; they are discussed in ???.

In addition to these functions, CP/M‑M3 maintains a base page in the TPA for each transient program. The base page contains initial values for the File Control Block (FCB) and Direct Memory Access (DMA) buffer. For details on the base page and loading transient programs, refer to section ???.

## Programming Tools and Commands

At the moment, there are no extant tools for developing CP/M‑M3 programs on the target system. Programs are developed on a host system using a cross-compiler and arrangements are made to get them to the target system.

CP/M‑M3 and the supplied utilities were developed on a Windows host using the Code Sourcery cross compiler. Since Code Sourcery is based on GNU C, development with other GNU C tools should be similar. Some work may be necessary to develop with other environments.

Currently, programs are made available to the target system by including them in the disk image that is burned to the internal flash of the LM3S9D92 processor. A disk image manipulation tool, imaginatively called dim, is provided to build the image. The program is a Windows executable developed using Visual Studio Express. Usage of dim is discussed in ???.

## File Specification

The CP/M‑M3 file specification is compatible with other CP/M systems. The format contains three fields: a 1 character drive select code (d), a 1- through 8-character filename (f...f) and a 1- through 3-character filetype (ttt) as shown below:

Format: d:ffffffff.ttt

Example: B:MYRAH.DAT

The drive select code and filetype fields are optional. A colon (:) delimits the drive select field. A period (.) delimits the filetype field. These delimiters are required only when the fields they delimit are specified.

Values for the drive select code range from A through P when the BIOS implementation supports 16 drives, the maximum number allowed. The range for the drive code is dependent on the BIOS implementation; the reference system supports only drive A. Drives are labeled A through P to correspond to the 1 through 16 drives supported by CP/M‑M3. However, not all BIOS implementations support the full range.

The characters in the filename and filetype fields cannot contain delimiters (the colon and period). A command line and its file specifications, if any, that are entered at the CCP level are automatically put in upper case internally before the CCP uses them.

However, not all commands and file specifications are entered at the CCP level. CP/M‑M3 does not prevent you from including delimiters in file specifications that are created or referenced by functions that bypass the CCP. For example, the BDOS Make File function (22) allows you to create a file specification that includes delimiters, although the CCP cannot parse and access such a file.

In addition to the delimiter characters already mentioned, you should avoid using the delimiter characters listed in Table 2 in the file specification of a file that you create. Several CP/M build-in commands and utilities have special uses for these characters.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Character** | **Description** |  | **Character** | **Description** |
| [ ] | Square brackets |  | ( ) | Parentheses |
| < > | Angle brackets |  | = | Equals sign |
| \* | Asterisk |  | & | Ampersand |
| , | Comma |  | ! | Exclamation point |
| | | Vertical bar |  | ? | Question mark |
| / | Forward slash |  | $ | Dollar sign |
| . | Period |  | : | Colon |
| ; | Semicolon |  | + | Plus sign |
| - | Minus sign |  |  |  |

Table Delimiter Characters

## Wildcards

CP/M‑M3 supports two wildcards, the question mark and the asterisk. Several utilities and BDOS functions allow you to specify wildcards in a file specification to perform the operation or function on one or more files. However, BDOS functions support only the ? wildcard.

The ? wildcard matches any character in the character position occupied by the wildcard. For example, the file specification M?RAH.DAT indicates that the second letter of the filename can be any alphanumeric character if the remainder of the file specification matches. Thus, the ? wildcard matches exactly one character position.

The \* wildcard matches one or more characters in the field or remainder of the field that this wildcard occupies. CP/M‑M3 internally pads the field or remaining portion of the field occupied by the \* wildcard with ? wildcards before searching for a match. For example, CP/M‑M3 converts the file specification B\*.DAT to B???????.DAT before searching for a matching file specification. Thus, any file with a filename that starts with the letter B and has a filetype of DAT matches this file specification.

## Terminology

The following table lists terminology used throughout this guide to describe CP/M‑M3 values and program components.

| **Term** | **Meaning** |
| --- | --- |
| Nybble | A 4-bit value. |
| Byte | An 8-bit value. |
| Word | A 16-bit value. |
| Longword | A 32-bit value. |
| Address | A 32-bit value that specifies a location in storage. |
| Offset | A fixed displacement defined by the user to reference a location in storage, other data source, or destination. |
| Text Segment | The section of a program that contains the program instructions. |
| Data Segment | The segment of a program that contains initialized data. |
| Block Storage Segment (bss) | The segment of a program that contains data that is either uninitialized or initialized to zero. |

Table CP/M‑M3 Terminology

The following table describes conventions used in this manual.

| **Convention** | **Meaning** |
| --- | --- |
| [ ] | Square brackets in a command line denote optional parameters. |
| 0xn | The characters 0x precede numeric values that are represented in hexadecimal notation. |
| numeric values | Unless otherwise stated, numeric values are presented in decimal notation. |
| (n) | BDOS function numbers are enclosed in parentheses when they appear in text. |
| .  .  . or ... | A vertical or horizontal ellipsis indicates missing elements in a series unless otherwise noted. |
| RETURN | The word RETURN refers to the RETURN key on the keyboard of your console. Unless otherwise noted, you must press RETURN after you enter a command line on your console to invoke a command. |
| CTRL‑X | The mnemonic CTRL‑X instructs you to press the key labeled CTRL while you press another key indicated by the variable X. For example, CTRL‑C instructs you to press the CTRL key while you simultaneously press the key lettered C. |

Table CP/M‑M3 Programmer's Guide Conventions

In figures that illustrate the layout of memory, addresses increase to the left.

# DATA STRUCTURES

Like most programs, CP/M‑M3 is held together by data structures, some of which are used to communicate with programs. This section describes those data structures.

The data structures described in this section are used to interact with the BDOS; they are declared in cpm.h. Additional data structures used to communicate with the BIOS are described in ???.

## The Base Page: cpm\_basepage\_t

The base page is a 256-byte data structure that describes the environment of a transient program, including where the program has been loaded, the command tail given to the program, and, most importantly, a way for the program to locate the services offered by the operating system. The operating system initializes the base page as it prepares to launch the program. Once the system is ready for the program to run, the CCP calls the entry point of the program, passing it a pointer to the base base.

Unlike CP/M‑80, the base page does not reside at a fixed location in memory. While it is true that the base page resides at the beginning of the TPA, the location of the TPA itself may vary from platform to platform. A transient program should use the pointer passed to its entry point to locate the base page instead of a built-in knowledge of the layout of the platform.

The CP/M‑M3 base page is declared as follows:

|  |
| --- |
| typedef struct cpm\_basepage\_s {  void \*tpabase; /\* :0x000 \*/  void \*tpatop; /\* :0x004 \*/  void \*textbase; /\* :0x008 \*/  unsigned int textlen; /\* :0x00c \*/  void \*database; /\* :0x010 \*/  unsigned int datalen; /\* :0x014 \*/  void \*bssbase; /\* :0x018 \*/  unsigned int bsslen; /\* :0x01c \*/  unsigned int freelen; /\* :0x020 \*/  unsigned char loadedfrom; /\* :0x024 \*/  unsigned char reserved0[ 3 ]; /\* :0x025 \*/  unsigned int (\*bdos)( /\* :0x028 \*/  unsigned short int func,  unsigned int parm );  void (\*entry)( /\* :0x02c \*/  struct cpm\_basepage\_s \*basepage );  void \*initialsp; /\* :0x030 \*/  unsigned int reserved1; /\* :0x034 \*/  cpm\_fcb\_t fcb2; /\* :0x038 \*/  cpm\_fcb\_t fcb1; /\* :0x05c \*/  unsigned char tail[ 128 ]; /\* :0x080 \*/  } cpm\_basepage\_t; /\* :0x100 \*/ |

Figure Declaration of the Base Page

The fields within the base page are:

* tpabase: Holds the base address of the Transient Program Area (TPA), the region of memory into which transient programs are loaded. Since the base page resides at the beginning of the TPA, this should also the address of the base page itself.
* tpatop: Holds the address of the byte immediately following the Transient Program Area (TPA); the byte addressed is not within the TPA.
* textbase: Holds the base address of the region of memory into which the text section of the program was loaded. Since the text section is loaded immediately after the base page, which occupies 256 bytes of memory, this will normally be tpabase + 0x100.
* textlen: Holds the length, in bytes, of the text section of the program. Since the CP/M‑M3 program loader is a simple-minded beast in the mold of CP/M‑80 that considers a program file to contain only a single combined text and data section, this is the size of the file that held the program.
* database: This field is intended to hold the base address of the data section of the program, but is not used by CP/M‑M3. The simple-minded program loader of CP/M‑M3 considers a program to consist of only a single, combined, text and data section. Programs should not rely on the value of this field.
* datalen: This field holds the length of the data section of the program. CP/M‑M3 initializes this field to zero, because it considers a program to consist of only a single combined text and data section.
* bssbase: This field is intended to hold the base address of the bss section of the program, but is not used by CP/M‑M3. The simple-minded program loader of CP/M‑M3 considers a program to consist of only a single, combined, text and data section. Programs should not rely on the value of this field.

**NOTE:** CP/M‑M3 does not clear the bss section of a program that it has loaded into memory. A program must clear its own bss section if required[[1]](#footnote-1).

* bsslen: This field holds the length of the bss section of the program. CP/M‑M3 initializes this field to zero, because it considers a program to consist of only a single combined text and data section.
* freelen: This field holds the length, in bytes, of the region of memory between that occupied by the program and the top of the TPA.
* loadedfrom: This field indicates from which drive the program was loaded. The number stored here is zero-based; zero refers to drive A and 15 refers to drive P.
* bdos: This field holds a pointer to the BDOS entry point; that is, a pointer to the function which the program should call to use the services of the operating system. Calling the operating system is discussed in section ???.
* entry: This field holds a pointer to the entry point of the program. The CP/M‑M3 program loader obtains this pointer from a data structure at the beginning of the program; see section ???. After the BDOS has loaded the program and the CCP has parsed the command line, the CCP calls the entry point of the program through this pointer.
* initialsp: This field holds the value used by the CCP to initialize the stack pointer before entering the program. Since the stack is placed at the top of the TPA, this field will normally hold the same value as tpatop.
* fcb2: This field holds a File Control Block (FCB) initialized by the CCP. If the command line contained at least two file specifications, this FCB is initialized from the second file specification. If the command line contained only one file specification, the filename and filetype fields of this FCB contain spaces.
* fcb1: This field holds a File Control Block (FCB) initialized by the CCP. If the command line contained at least one file specification, this FCB is initialized from the first file specification. If the command line held no file specifications, the filename and filetype fields of this FCB contain spaces.
* tail: This field holds the command tail, the portion of the command line that remains after the command itself is removed from the line. If the command line contains up to two file specifications, the CCP parses them into the fcb1 and fcb2 fields, but does not remove them from the command line; they remain in the command tail.

|  |  |
| --- | --- |
| |  | | --- | |  |   Figure The tail Field of the Base Page |

The tail field is a character buffer containing the following bytes:

* + tail[ 0 ] holds the number of bytes in the command tail.
  + tail[ 1 ] through tail[ tail[ 0 ] ]holds the Characters in the command tail.
  + tail[ tail[ 0 ] + 1 ] holds a NUL byte to terminate the string.

Because tail contains both a count byte and a terminal NUL, the command tail can hold no more than 126 characters.

### Initialization of the Base Page

The BDOS and the CCP conspire to initialize the base page when a transient program is loaded. The program is loaded by the BDOS Program Load function (59), which initializes the following fields:

|  |
| --- |
| tpabase, tpatop, textbase, textlen, datalen, bsslen, freelen, loadedfrom, bdos, initialsp, and entry. |

Figure Base Page Fields Initialized by the BDOS

After the program is loaded, the CCP parses the command line and uses information from the command to initialize the following fields:

|  |
| --- |
| fcb1, fcb2, and tail. |

Figure Base Page Fields Initialized by the CCP

In addition to initializing fields in the base page, the CCP ensures that the DMA address is set to refer to the tail field of the base page before it enters the loaded program.

A program that sidesteps the CCP to load another program using the BDOS Program Load function (59) is responsible for doing the portion of the base page initialization that would have been done by the CCP.

## The File Control Block: cpm\_fcb\_t

|  |
| --- |
| typedef struct cpm\_fcb\_s {  unsigned char dr;  unsigned char f[ 8 ];  unsigned char t[ 3 ];  unsigned char ex;  unsigned char s[ 2 ];  unsigned char rc;  unsigned char d[ 16 ];  unsigned char cr;  unsigned char r[ 3 ];  } cpm\_fcb\_t; |

The fields within the file control block are:

dr: Indicates on which drive the file resides. This is one-based; drive zero refers to the default drive, drive 1 to A:, drive 2 to B:, etc.

f: The file name; the portion of the filename before the period.

t: The file type; the portion of the filename after the period. The most-significant bit of each of these bytes is used for a file attribute. File attributes are:

t[0]: Set indicates that the file is read-only.

t[1]: Set indicates that the file is a system file.

t[2]: Set indicates that the file has been archived. The operating system clears this bit when the file is written to. Archiving programs can set this bit to detect that the file has been changed since it was last archived.

ex: The current extent number. A CP/M file is a collection of extents, each containing 16,384 bytes.

s: Reserved for system use. Byte s[ 1 ] needs to be cleared for Open, Make, and Search services.

rc: Record count. Holds the number of 128-byte records held in the current extent of the file.

d: Disk allocation map. Reserved for system use.

cr: Current record. Holds the number of the next record in the current extent to be accessed sequentially. Advanced by the sequential operations.

r: Random record number. Holds the number of the next record in the file to be affected by a random operation. Not used for sequential operations (can be omitted if a file is accessed strictly sequentially). The least-significant byte is r[ 0 ], regardless of native byte ordering.

## The Load Parameter Block: cpm\_lpb\_t

This function is used for the Program Load service, which is implemented by the architecture-specific glue.

|  |
| --- |
| typedef struct cpm\_lpb\_s {  unsigned char \*fcb; /\* :0x00 \*/  unsigned int tpabase; /\* :0x04 \*/  unsigned int tpatop; /\* :0x08 \*/  cpm\_basepage\_t \*basepage; /\* :0x0c \*/  unsigned int flags; /\* :0x10 \*/  } cpm\_lpb\_t; |

The fields are:

* fcb: Holds a pointer to the file control block through which the program is to be read.
* tpabase: Holds the address of the first byte of the transient program area into which the program is to be loaded.
* tpatop: Holds the address of the byte immediately following the transient program area into which the program is to be loaded.
* basepage: Receives a pointer to the base page for the program.
* flags: Holds loader control flags. CP/M-68K defines the following flags, but they are not supported by CP/M-M3:
  + bit 0: The program is to be loaded into the bottom of the transient program area when this bit is clear. It is loaded into the top when this is set.

## The BIOS Parameter Block: cpm\_bpb\_t

A pointer to a structure of this type is passed to the Direct BIOS Call service. It provides the information needed to call a BIOS function, to wit: the function number and parameters to be passed to the function.

|  |
| --- |
| typedef struct cpm\_bpb\_s {  unsigned short int func; /\* :0x00 \*/  unsigned long int p1; /\* :0x02 \*/  unsigned long int p2; /\* :0x06 \*/  } \_\_attribute\_\_((packed)) cpm\_bpb\_t; |

The fields are:

* func: Holds the number of the BIOS service to be called.
* p1: Holds the first parameter to be passed to that function.
* p2: Holds a second parameter to be passed to that function, should one be required.

## The Executable Header: cpm\_exehdr\_t

A CP/M-M3 executable program begins with a header of this type.

|  |
| --- |
| typedef struct cpm\_exehdr\_s {  unsigned int branch; /\* :0x00 \*/  unsigned int tag; /\* :0x04 \*/  unsigned int region; /\* :0x08 \*/  void (\*entry)( cpm\_basepage\_t \*); /\* :0x0c \*/  } cpm\_exehdr\_t; |

The fields are:

* branch: Space to hold a branch to the program entry point. Whether this is needed depends upon how the entry field is used. Other architectures (VAX and ARM) entered a program by jumping to this field. The Cortex-M3 architecture added the entry field as a means of ensuring that the entry point address is odd to indicate that Thumb instructions are to be executed.
* tag: Holds a magic number that identifies the processor architecture for which the program was compiled. For the Cortex-M3 port, this field is expected to contain 0x0000334d ("M3").
* region: Holds the address at which the program expects to be loaded.
* entry: Holds a pointer to the program's entry point. The entry point is passed a pointer to the base page, which provides a means for it to use the CP/M services.

## The Transient Program Area Descriptor: cpm\_tpa\_t

This type is used by function 63 to describe the transient program area. It contains pointers to the first byte of the TPA, the byte immediately following the TPA, and flags that indicate whether the function is to set or get the limits and, if setting, whether those limits are to be permanent.

|  |
| --- |
| typedef struct cpm\_tpa\_s {  unsigned short int flags;  char \*base;  char \*top;  } \_\_attribute\_\_((packed)) cpm\_tpa\_t; |

# THE CCP AND TRANSIENT PROGRAMS

This section discusses the Console Command Processor (CCP), built-in and transient commands, loading and exiting transient programs, and CP/M‑M3 memory models.

## CCP Built-in and Transient Commands

After an initial cold start, CP/M‑M3 displays a sign-on message at the console. The disk containing the system, usually drive A, is logged in automatically. The standard prompt (>), preceded by the letter for the drive containing the system, is displayed on the console screen. This prompt informs the user that CP/M‑M3 is ready to receive a command line from the console.

In response to the prompt, a user types the filename of a command file and a command tail, if required. CP/M‑M3 supports two types of command files: built-in commands and transient commands. Built-in commands are configured and reside in memory with CP/M‑M3. Transient commands are loaded into the TPA and do not reside in memory allocated to CP/M‑M3. The following list contains seven built-in commands that all CP/M‑M3 systems support[[2]](#footnote-2):

DIR DIRS ERA REN TYPE USER SUBMIT

A transient command is a machine-readable executable program file in memory. A transient command file is loaded from disk to memory. Section ??? describes the format of transient command files.

When the user enters a command line, the CCP parses it and tries to execute the file specified. The CCP assumes a file is a command file when it has any filetype other than SUB. When the user specifies only the filename but not the filetype, the CCP searches for and tries to execute a file with a matching filename and a filetype of COM. The CCP searches the current user and user number 0 for a matching file. If a command file is not found, but the CCP finds a matching file with a filetype of SUB, the CCP executes it as a submit file.

## Loading a Program in Memory

Either the CCP or a transient program can load a program in memory with the BDOS Program Load function (59) described in section ???. After the program is loaded, the TPA contains the program's segments (text, data, and bss), a user stack, and a base page.

### Program Segments

Fundamentally, a program contains three segments:

* A text segment, containing the program's executable code.
* A data segment, containing pre-initialized data.
* A bss segment, containing uninitialized data or data initialized to zero.

When the program is linked, the segments from the individual modules are gathered and written to a file containing an image of the program as it appears in memory. The bss segment is typically omitted from the image.

The CP/M‑M3 program loader is a simple-minded beast inspired by CP/M‑80. It expects an executable file to contain an absolute binary image of the program linked to reside at the location at which it will be loaded. From the perspective of CP/M‑M3, an executable program consists of a single combined text and data segment; CP/M‑M3 does not know where the text segment ends and the data segment begins.

#### Initialization of the BSS Section

The bss section is not initialized by CP/M‑M3 when it loads a program; the section is typically omitted from an executable image and CP/M‑M3 does not know how large it is. While this is not a problem for uninitialized variables, it is troublesome for variables initialized to zero that have been placed in the bss segment by the linker. The program should begin by executing a small chunk of code that clears the bss segment, startlingly similar to this:

|  |
| --- |
| void clearbss( void )  {  extern unsigned char \_\_bss\_start\_\_;  extern unsigned char \_\_bss\_end\_\_;  extern unsigned char \_\_end\_\_;  unsigned char \*ThisByte;  for( ThisByte = &\_\_bss\_start\_\_; ThisByte < &\_\_bss\_end\_\_; ThisByte++ )  \*ThisByte = 0;  return;  } |

Figure Clearing the BSS Section

This function makes use of symbols defined by the GNU linker to describe the location and size of the bss segment. Your mileage may vary.

### The User Stack

Before entering a program, CP/M‑M3 initializes the stack pointer so that the stack resides at the top of the region into which the program was loaded; the stack grows downward. Nothing is pushed onto the stack before the program is entered; in accordance with the ARM procedure calling standard, the return address and the pointer to the base page are passed in registers.

The space between the top of the bss segment and the stack pointer is free space available for use by the loaded program.

### The Base Page

The base page is a 256-byte data structure that describes the environment of a transient program, including where the program has been loaded, the command tail given to the program, and, most importantly, a way for the program to locate the services offered by the operating system. A base page exists for each program loaded in memory. Unlike other CP/M systems, the base page in CP/M‑M3 does not reside at a fixed absolute address prior to a program being loaded; the BDOS Program Load function (59) determines the absolute address of the base page when the program is loaded into memory. When a program is loaded by the CCP, the BDOS Program Load function (59) and the CCP conspire to initialize the base page as the operating system prepares to launch the program. Once the system is ready for the program to run, the CCP calls the entry point of the program, passing it a pointer to the base page.

The BDOS Program Load function (59) places the base page in the first 256 bytes of the region into which the program is loaded. For a program loaded by the CCP, this means that the base page resides in the first 256 bytes of the TPA. Since the TPA can be located at different addresses for different CP/M‑M3 platforms, a transient program should use the pointer passed to its entry point to locate the base page instead of a built-in knowledge of the layout of the platform.

The CP/M‑M3 base page is declared in cpm.h as follows:

|  |
| --- |
| typedef struct cpm\_basepage\_s {  void \*tpabase; /\* :0x000 \*/  void \*tpatop; /\* :0x004 \*/  void \*textbase; /\* :0x008 \*/  unsigned int textlen; /\* :0x00c \*/  void \*database; /\* :0x010 \*/  unsigned int datalen; /\* :0x014 \*/  void \*bssbase; /\* :0x018 \*/  unsigned int bsslen; /\* :0x01c \*/  unsigned int freelen; /\* :0x020 \*/  unsigned char loadedfrom; /\* :0x024 \*/  unsigned char reserved0[ 3 ]; /\* :0x025 \*/  unsigned int (\*bdos)( /\* :0x028 \*/  unsigned short int func,  unsigned int parm );  void (\*entry)( /\* :0x02c \*/  struct cpm\_basepage\_s \*basepage );  void \*initialsp; /\* :0x030 \*/  unsigned int reserved1; /\* :0x034 \*/  cpm\_fcb\_t fcb2; /\* :0x038 \*/  cpm\_fcb\_t fcb1; /\* :0x05c \*/  unsigned char tail[ 128 ]; /\* :0x080 \*/  } cpm\_basepage\_t; /\* :0x100 \*/ |

Figure Declaration of the Base Page

The fields within the base page are:

* tpabase: Holds the base address of the region of memory into which the program was loaded. For programs loaded by the CCP, this is the base address of the Transient Program Area (TPA). Since the base page resides at the beginning of the load region, this should also the address of the base page itself.
* tpatop: Holds the address of the byte immediately following the region into which the program was loaded; the byte at this address is not within the load region.
* textbase: Holds the base address of the region of memory into which the text section of the program was loaded. Since the text section is loaded immediately after the base page, which occupies 256 bytes of memory, this will normally be tpabase + 0x100.
* textlen: Holds the length, in bytes, of the text section of the program. Since the CP/M‑M3 program loader is a simple-minded beast in the mold of CP/M‑80 that considers a program file to contain only a single combined text and data section, this is the size of the file that held the program.
* database: This field is intended to hold the base address of the data section of the program, but is not used by CP/M‑M3. The simple-minded program loader of CP/M‑M3 considers a program to consist of only a single combined text and data section. Programs should not rely on the value of this field.
* datalen: This field holds the length of the data section of the program. CP/M‑M3 initializes this field to zero, because it considers a program to consist of only a single combined text and data section.
* bssbase: This field is intended to hold the base address of the bss section of the program, but is not used by CP/M‑M3. The simple-minded program loader of CP/M‑M3 considers a program to consist of only a single combined text and data section. Programs should not rely on the value of this field.

**NOTE:** CP/M‑M3 does not clear the bss section of a program that it has loaded into memory. A program must clear its own bss section if that is required[[3]](#footnote-3).

* bsslen: This field holds the length of the bss section of the program. CP/M‑M3 initializes this field to zero, because it considers a program to consist of only a single combined text and data section.
* freelen: This field holds the length, in bytes, of the region of memory between that occupied by the program and the top of the region into which the program was loaded.
* loadedfrom: This field indicates from which drive the program was loaded. The number stored here is zero-based; zero refers to drive A and 15 refers to drive P.
* bdos: This field holds a pointer to the BDOS entry point; that is, a pointer to the function which the program should call to use the services of the operating system. Calling the operating system is discussed in section ???.
* entry: This field holds a pointer to the entry point of the program. The CP/M‑M3 program loader obtains this pointer from a data structure at the beginning of the program; see section ???. After the BDOS has loaded the program and the CCP has parsed the command line, the CCP calls the entry point of the program through this pointer.
* initialsp: This field holds the value used by the CCP to initialize the stack pointer before entering the program. Since the stack is placed at the top of the load region, this field will normally hold the same value as tpatop.
* fcb2: This field holds a File Control Block (FCB) initialized by the CCP. If the command line contained at least two file specifications, this FCB is initialized from the second file specification. If the command line contained only one file specification, the filename and filetype fields of this FCB contain spaces. The FCB is described in section ???.
* fcb1: This field holds a File Control Block (FCB) initialized by the CCP. If the command line contained at least one file specification, this FCB is initialized from the first file specification. If the command line held no file specifications, the filename and filetype fields of this FCB contain spaces. The FCB is described in section ???.
* tail: This field holds the command tail, the portion of the command line that remains after the command itself is removed from the line. If the command line contains up to two file specifications, the CCP parses them into the fcb1 and fcb2 fields, but does not remove them from the command line; they remain in the command tail.

|  |  |
| --- | --- |
| |  | | --- | |  |   Figure The tail Field of the Base Page |

The tail field is an array of 128 characters containing the following bytes:

* + tail[ 0 ] holds the number of bytes in the command tail.
  + tail[ 1 ] through tail[ tail[ 0 ] ]holds the characters in the command tail.
  + tail[ tail[ 0 ] + 1 ] holds a NUL byte to terminate the string.

Because tail contains both a count byte and a terminal NUL, the command tail can hold no more than 126 characters.

#### Initialization of the Base Page

The BDOS and the CCP conspire to initialize the base page when a transient program is loaded.

The CCP extracts the command from the command line and loads the program using the BDOS Program Load function (59). That BDOS function allocates space for the base page and initializes the following fields:

* tpabase, which contains the base address of the TPA (and, consequently, the address of the base page itself).
* tpatop, which contains the address of the byte immediately following the TPA.
* textbase, which contains the address at which the program was loaded. Since the program is loaded immediately after the base page, textbase contains tpabase + 0x100.
* textlen, which contains the size, in bytes, of the loaded program.
* datalen and bsslen, which are intended to describe the length of the .data and .bss sections of the program, are cleared. As described in section ???, the executable format used by CP/M‑M3 is similar to that of CP/M‑80; it considers a program to consist of only a combined .text and .data section.
* freelen, which contains the size, in bytes, of the region of memory between the end of the program and the top of the TPA.
* loadfrom, which contains the zero-based number of the drive from which the program was loaded.
* bdos, which contains a pointer to the BDOS entry point; the program is expected to call this function to access the BDOS services.
* entry, which contains the entry point of the program. This is obtained from a data structure, described in section ???, at the beginning of a program.
* initialsp, which contains the initial value of the stack pointer. Since the stack resides at the top of the TPA, this is initialized to the same value as the tpatop field.

Once the program has been loaded, the CCP parses the command line and initializes the following fields:

* fcb1, which contains an FCB constructed from the first file specification found on the command line. If there was not such a file specification, the filename and type fields of fcb1 are initialized to spaces.
* fcb2, which contains an FCB constructed from the second file specification found on the command line. If there was not such a file specification, the filename and type fields of fcb2 are initialized to spaces.
* tail, which contains the command tail. The command tail has been converted to upper case. The format of the command tail is described ???.

The CCP then calls the entry point of the program, passing it a single parameter: a pointer to the base page. The program may exit by returning from its entry point or using the BDOS System Reset function (0).

1. Most programs written in C require the bss section to be cleared because most C compilers place variables that have been initialized to zero in the bss section. [↑](#footnote-ref-1)
2. Additional built-in commands may exist to support the needs of a specific hardware platform. Consult the documentation for the platform for more information about such commands. [↑](#footnote-ref-2)
3. Most programs written in C require the bss section to be cleared because most C compilers place variables that have been initialized to zero in the bss section. [↑](#footnote-ref-3)