CP/M 2.2 File Control Block (FCB)

Complete Byte-by-Byte Reference

The File Control Block (FCB) is a 36-byte data structure used by CP/M to manage file operations. Each open file requires its own FCB, which contains the filename, extent information, and disk allocation details. Programs must initialize certain FCB fields before opening or creating files, while CP/M maintains other fields automatically during file operations. This document describes every byte of the FCB structure in detail.

FCB Structure Overview

Byte(s)	Field	Description
0	dr	Drive code (0=default, 1=A:, 2=B:, ... 16=P:)
1-8	f1-f8	Filename (8 bytes, space-padded, 7-bit ASCII)
9-11	t1-t3	File type (3 bytes, space-padded, with attributes)
12	ex	Extent number (low byte, 0-31)
13	s1	Reserved (set to 0 by programs)
14	s2	Extent number (high byte) / Module number
15	rc	Record count (0-128) in current extent
16-31	d0-d15	Disk allocation map (block numbers)
32	cr	Current record (0-127) within extent
33-35	r0-r2	Random access record number (24-bit)

Important: Sequential file access uses bytes 0-32 (33 bytes). Random access requires bytes 0-35 (36 bytes total). The default FCB at 0x005C has enough space for both, with random access fields overlapping the second FCB at 0x006C.

Byte-by-Byte Detailed Description

Byte(s)	Field	Size	Description and Usage
0	**`dr`**	1	Drive code. **User sets on open/make.** • 0 = Use current default drive • 1 = Drive A: • 2 = Drive B: • ... up to 16 = Drive P: • '?' (0x3F) = Wildcard for search operations Programs typically set this to 0 or a specific drive number.
1-8	**`f1-f8`**	8	Filename (8 characters). **User sets on open/make.** • 7-bit ASCII characters (A-Z, 0-9, some special chars) • Padded with spaces (0x20) if less than 8 characters • Automatically converted to uppercase by CCP • '?' (0x3F) in any position = wildcard for searches • Invalid characters: < > . , ; : = [ ] * ? • Bit 7 of each byte should be 0 (not used for attributes)
9-11	**`t1-t3`**	3	File type/extension (3 characters). **User sets on open/make.** • Same format as filename: 7-bit ASCII, space-padded • Common types: COM, ASM, BAS, TXT, HEX, etc. **Attribute bits (bit 7 of each type byte):** • **t1' (bit 7 of byte 9):** Read-Only flag (1=R/O, 0=R/W) • **t2' (bit 7 of byte 10):** System file flag (1=SYS/hidden) • **t3' (bit 7 of byte 11):** Archive flag (1=not modified)
12	**`ex`**	1	Extent number (low byte). **Set to 0 by user on open/make.** CP/M manages this automatically during file operations. • Range: 0-31 for standard systems • Files >16K require multiple extents • Actual extent = ((32*S2)+EX) / (EXM+1) where EXM = extent mask from Disk Parameter Block • Each extent typically represents 16K of file data • Sequential extents: 0, 1, 2, 3, ... (but may be non-contiguous on disk)
13	**`s1`**	1	Reserved byte. **User must set to 0.** • CP/M 2.2 does not use this field • Always initialize to 0 when opening/creating files • Some CP/M clones may use this for extensions • MP/M may use this for file locking information
14	**`s2`**	1	Extent number (high byte) / Module number. **Set to 0 by user.** CP/M manages this for very large files (>512K). • Normally 0 for files under 512K • Increments for files requiring more than 32 extents • Formula: File offset = ((32*S2)+EX)*16384 + CR*128 **Special usage:** • **Bit 7 (s2'):** File Write Flag in CP/M 2.2 - Set when file opened, cleared if FCB modified - Used internally to track if close needs to update directory

Byte(s)	Field	Size	Description and Usage
15	**`rc`**	1	Record count in current extent. **Set to 0 by user on open.** CP/M updates this automatically during file I/O. • Range: 0-128 (0x00-0x80) • Number of 128-byte records used in this extent • 128 (0x80) = extent is full (16,384 bytes) • Last extent of file typically has RC < 128 • Used to determine actual file size within extent

Byte(s)	Field	Size	Description and Usage
16-31	**`d0-d15`**	16	Disk allocation map. **CP/M manages automatically.** Contains block numbers where file data is stored. **Block size depends on disk format:** • 1K blocks: Each byte = one block number (0-255) - 16 bytes can reference 16 blocks = 16K max per extent • 2K blocks: Each byte = one block number (0-255) - 8 blocks referenced (using d0-d7 only) • 4K+ blocks: Two bytes per block number (little-endian) - d0-d1 = first block, d2-d3 = second block, etc. - Can reference 8 large blocks **Block numbering:** • Block 0 = first data block (after directory) • Blocks allocated as file grows • Unused entries = 0x00 • Non-contiguous blocks are normal (file fragmentation) **Determining block size:** Check DSM (disk size) in Disk Parameter Block: • If DSM < 256: 1-byte block numbers (1K or 2K blocks) • If DSM ≥ 256: 2-byte block numbers (4K+ blocks)
32	**`cr`**	1	Current record (sequential access). **User typically sets to 0.** CP/M updates during sequential read/write operations. • Range: 0-127 (points to one of 128 records in extent) • Increments automatically with each read/write • When CR reaches 128, next extent is opened automatically • Reset to 0 when new extent opened • File position = (EX*16384) + (CR*128) bytes from start • Setting CR=0 after open allows sequential access from start
33-35	**`r0-r2`**	3	Random access record number. **User sets for random access.** **These bytes are ONLY used for random access file operations.** Not used during sequential access. **r0-r1 (bytes 33-34):** 16-bit record number (little-endian) • Range: 0-65535 (0x0000-0xFFFF) • Points to specific 128-byte record in entire file • r0 = low byte, r1 = high byte **r2 (byte 35):** Overflow byte for very large files • Normally 0 for files under 8MB • CP/M 2.2: Allows files up to 8MB (65536 records) • CP/M 3: Full 24-bit support (16,777,216 records = 2GB) **Random access formula:** Record number = r0 + (r1 * 256) + (r2 * 65536) Byte position = Record number * 128 **BDOS Function 36 (Set Random Record):** Calculates r0-r2 from current sequential position (EX, CR). Allows switching from sequential to random access.

Understanding Extents

What is an Extent?
An extent is a portion of a file described by one directory entry. CP/M divides files into logical 16K segments called extents. Each extent can reference up to 16K of disk blocks through its allocation map (d0-d15).

Why Extents?
• Keeps counters as 8-bit values (simpler arithmetic on 8-bit CPUs)
• Limits directory entry size to 32 bytes on disk
• Allows files larger than 16K by chaining multiple extents

Extent Numbering:
The extent number is calculated as: `((32 * S2) + EX) / (EXM + 1)`
Where EXM (Extent Mask) comes from the Disk Parameter Block and depends on block size:

Block Size	EXM Value	Extents per Directory Entry
1K blocks	0	1 extent = 16K
2K blocks	1	2 extents = 32K
4K blocks	3	4 extents = 64K
8K blocks	7	8 extents = 128K

Example for 2K blocks (EXM=1):
• Directory entry 0: Covers extents 0-1 (EX values 0,1)
• Directory entry 1: Covers extents 2-3 (EX values 2,3)
• Directory entry 2: Covers extents 4-5 (EX values 4,5)

File Position Calculation:
```
File position = (Extent Number × 16384) + (CR × 128) bytes
```

FCB Initialization for Common Operations

Opening an Existing File (BDOS Function 15)

```
; Initialize FCB to open file TEST.TXT on drive B:
MVI A,2 ; Drive B:
STA FCB+0 ; Set dr field
LXI H,FNAME ; Point to 'TEST TXT'
LXI D,FCB+1 ; Point to f1 field
MVI B,11 ; 11 bytes (8+3)
CALL MOVE ; Copy filename and type
XRA A ; A = 0
STA FCB+12 ; Set EX = 0
STA FCB+13 ; Set S1 = 0
STA FCB+14 ; Set S2 = 0
STA FCB+15 ; Set RC = 0
STA FCB+32 ; Set CR = 0
LXI D,FCB ; DE points to FCB
MVI C,15 ; BDOS function 15 (Open)
CALL BDOS ; Open the file
CPI 0FFH ; Check for error
JZ ERROR ; Jump if file not found
; File opened successfully, ready for I/O
```

Creating a New File (BDOS Function 22)

```
; Initialize FCB to create file NEWFILE.DAT
XRA A ; A = 0
STA FCB+0 ; dr = 0 (default drive)
LXI H,NEWNAME ; Point to 'NEWFILE DAT'
LXI D,FCB+1 ; Point to f1 field
MVI B,11 ; 11 bytes
CALL MOVE ; Copy filename
XRA A ; A = 0
STA FCB+12 ; EX = 0
STA FCB+13 ; S1 = 0
STA FCB+14 ; S2 = 0
STA FCB+15 ; RC = 0
STA FCB+32 ; CR = 0
LXI D,FCB
MVI C,22 ; BDOS function 22 (Make)
CALL BDOS
CPI 0FFH
JZ DIRFULL ; Directory full
; File created, ready to write
```

Random Access (BDOS Functions 33-34)

```
; Read record number 1000 (decimal) from file
; 1000 decimal = 0x03E8 = low byte 0xE8, high byte 0x03

; First open the file (as shown above)
LXI D,FCB
MVI C,15 ; Open file
```

```
CALL BDOS

; Set random record number
LXI H,1000 ; HL = record number
SHLD FCB+33 ; Store in r0-r1
XRA A
STA FCB+35 ; r2 = 0

; Read the record
LXI D,FCB
MVI C,33 ; BDOS function 33 (Read Random)
CALL BDOS
ORA A
JNZ READERR
; Data now in DMA buffer (default 0x0080)
```

Converting Sequential Position to Random (Function 36)

```
; After sequential reads, convert current position to random
; Allows switching from sequential to random access

LXI D,FCB
MVI C,36 ; BDOS function 36 (Set Random Record)
CALL BDOS
; r0-r2 now contain equivalent random record number
; Formula: r0-r2 = (EX * 128) + CR
```

File Attributes (High Bits of Type Bytes)

File attributes are stored in bit 7 (high bit) of the file type bytes. The lower 7 bits contain the ASCII character for the file type.

Read-Only (t1' - bit 7 of byte 9):
• Set to 1: File cannot be written to or deleted
• Set to 0: File is read-write (normal)
• Set with BDOS function 30 (Set File Attributes)
• Example: If type is "COM" and file is R/O, byte 9 = 'C' | 0x80 = 0xC3

System File (t2' - bit 7 of byte 10):
• Set to 1: File is hidden from normal directory listings
• Set to 0: File is visible (normal)
• System files in User 0 can be accessed from other user areas
• Used for CP/M system files (CCP.COM, BDOS.COM, BIOS.COM)

Archive (t3' - bit 7 of byte 11):
• Set to 1: File has NOT been modified since last backup
• Set to 0: File has been modified
• Used by backup utilities to identify changed files
• Automatically cleared by BDOS when file is written to

```
Example FCB with attributes:
Byte 9: 'T' | 0x80 = 0xD4 (Type="TXT", Read-Only)
Byte 10: 'X' | 0x80 = 0xD8 (System file)
Byte 11: 'T' | 0x00 = 0x54 (Modified since backup)
```

Critical Programming Notes

1. Field Initialization:
Always initialize dr, f1-f8, t1-t3, EX, S1, S2, and RC before opening or creating files. Leave d0-d15 alone - CP/M manages the allocation map.

2. Sequential vs Random Access:
• Sequential: Uses bytes 0-32 (33 bytes)
• Random: Uses bytes 0-35 (36 bytes)
• Don't mix modes without calling function 36 to synchronize

3. Extent Overflow:
When reading/writing sequentially, CP/M automatically opens the next extent when CR reaches 128. Your program doesn't need to do anything - it's transparent.

4. File Size Calculation:
To find actual file size:
`Size = (Highest_Extent_Number × 16384) + (RC × 128) bytes` Use BDOS function 35 (Compute File Size) which sets r0-r2 to the file size in records.

5. Closing Files:
ALWAYS close files after writing (BDOS function 16). This flushes buffers and updates the directory. Failure to close can result in data loss.

6. Wildcard Searches:

Use '?' (0x3F) in filename/type fields for wildcard matching with functions 17-18 (Search First/Next).

7. Multiple FCBs:
Each open file needs its own FCB. The default FCB at 0x005C is convenient for single-file programs, but multi-file programs must allocate FCBs in their own data area.

8. User Numbers:
FCBs don't contain user numbers. The current user number is stored at 0x0004 (high nibble). Files are associated with the user number that was active when they were created.

9. Don't Modify CP/M Fields:
Never manually modify EX, S2 (except bit 7), RC, or d0-d15 during file operations. Let CP/M manage these fields. Only modify them when rewinding or repositioning files.

10. Rewinding a File:
To rewind to beginning:

```
XRA A ; A = 0
STA FCB+12 ; EX = 0
STA FCB+14 ; S2 = 0
STA FCB+15 ; RC = 0
STA FCB+32 ; CR = 0
```

Example FCB Hex Dumps

Example 1: Newly opened file "TEST.COM" on drive A:

```
Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11...
Value: 01 54 45 53 54 20 20 20 20 43 4F 4D 00 00 00 80 04 05...
Field: dr T E S T ■ ■ ■ ■ C O M ex s1 s2 rc d0 d1...

• dr=01 (Drive A:)
• Filename="TEST " (space-padded)
• Type="COM"
• ex=00 (first extent)
• s1=00, s2=00 (reserved/unused)
• rc=80 (128 records = full extent, 16K)
• d0-d15=04,05,06,07,08,09,0A,0B (8 blocks of 2K each)
```

Example 2: Read-only system file "HIDDEN.SYS":

```
Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
Value: 00 48 49 44 44 45 4E 20 20 D3 D9 D3 00 00 00 50
Field: dr H I D D E N ■ ■ S* Y* S* ex s1 s2 rc

• dr=00 (default drive)
• Filename="HIDDEN "
• Type="SYS" with attributes:
- Byte 09 (S): 0xD3 = 'S'(0x53) | 0x80 = Read-Only
- Byte 10 (Y): 0xD9 = 'Y'(0x59) | 0x80 = System
- Byte 11 (S): 0xD3 = 'S'(0x53) | 0x80 = Archived
• rc=50 (80 records = 10,240 bytes in this extent)
```