X16 Edit user manual

December 14, 2025

Contents

1 Introduction

Thank you for trying out X16 Edit!

X16 Edit is a simple text editor written in 65C02 assembly especially for the Commander X16 retro computer.

The look and feel of the program is inspired by GNU Nano. There are, naturally, a lot of differences, but you will feel at home if you are used to Nano.

The Commander X16 was devised by David Murray a.k.a. the 8-Bit Guy. For more information on the platform, go to www.commanderx16.com.

2 Basic usage

2.1 Getting started

As of Kernal version R45, the editor is stored in ROM and can be started with the BASIC command EDIT.

There is also a RAM based version of the editor. It consists of just one file, X16EDIT-x.x.x.PRG, where x.x.x is the version number. The RAM based editor is loaded and run in the same way as a BASIC program.

To run the RAM based editor on hardware, just store the program file on an SD card. Insert the SD card into your X16 computer. Type `LOAD"X16EDIT-x.x.x.PRG"` to load the program. And then `RUN` to start it.

If you want to run the RAM based editor in the x16emu emulator, you first need to store the X16 Edit executable in the host computer's file system. Then type the following command in the terminal:

```
x16emu -prg X16EDIT-x.x.x.PRG -run
```

A valid path to the editor executable must be specified after the "-prg" option. Otherwise the emulator will not know where to find the program.

The "-run" option automatically starts the editor after the emulator is booted. You may remove this option and start the editor manually by typing `RUN` in the emulator.

2.2 Entering text

X16 Edit is a modeless text editor. As soon as it is started, everything you type is stored in the text buffer.

The cursor is moved with keys normally used for that purpose on a modern keyboard, *i.e.* the arrow keys, Tab, Home, End, PgUp and PgDn.

The go to line feature (Ctrl+L) lets you move the cursor to a specific line number.

As you type and reach the right margin, the editor does not by default insert a line break. Instead the current line is scrolled horizontally to make room for more characters. There is no limit to the length of a line other than the available memory.

2.3 Saving and loading text files

The current text buffer is saved to file on the SD card when you press Ctrl+O. Type in the file name you like to use, and press the ENTER key. You will be prompted to confirm before overwriting a file that already exists.

To load a file from the SD card, press Ctrl+R. Just type in the name of the file you want to load, and then press the ENTER key.

2.4 Keyboard shortcuts

X16 Edit is controlled by keyboard shortcuts. You find a complete list of shortcuts in Appendix A.

Shortcuts are primarily selected by pressing and holding down Ctrl at the same time as you press a shortcut key.

You may also press and release the ESC key. A message is displayed in the status bar indicating that the program is ready to receive a command. Just press a shortcut key without holding down any modifier.

2.5 Mouse control

The editor is primarily made to be controlled by the keyboard. There are, however, some basic mouse controls.

The cursor may be positioned in the file by clicking the left mouse button.

You may select part of the text by pressing the left mouse button where you want the selection to start and release it where you want the selection to end. Selections can be copied or cut. More on that later.

The mouse is automatically enabled when you start the editor. The mouse pointer is hidden while typing on the keyboard, and shown again when you move or click the mouse.

2.6 User interface

X16 Edit's user interface is similar to GNU Nano, and should be mostly self-explanatory. It consists of the following main parts:

- Title bar
- Editor area
- Status bar
- Shortcut list

The *title bar* is at the top row of the screen. The name of the current file is displayed at the center. If the text buffer has not been saved to file, it displays "New text buffer". At the right edge the letters "MOD" are shown if the text buffer has been changed since last saved to file.

The *editor area* is right below the title bar. It takes up most of the screen, and it is here you do all text editing.

The *status bar* is at the third row from the bottom of the screen. All messages from the program are displayed in the status bar. If the program needs to prompt you for input, the input prompt is displayed here as well.

At the input prompt you may either press ENTER to confirm your input or press ESC to abort the current operation.

The active *line break encoding* and the current *cursor position* are displayed at the left and right edge of the status bar respectively, if there is no active message or prompt.

The *shortcut list* takes up the last two rows of the screen. Here you find the most common keyboard shortcuts.

X16 Edit uses the screen mode (screen height and width) that is active when starting the program. Especially the shortcut list and the help screen change appearance to fit in low screen resolutions.

2.7 Built-in help

A lists of all keyboard shortcuts and a short description of each command is available in the built-in help screen (Ctrl+G).

3 More on text editing

3.1 Supported character sets

The editor supports the following built-in character sets:

- PETSCII upper case/graphics (standard and thin). This is the default mode of both the Commander X16 and the C64.
- PETSCII upper/lower case (standard and thin). This is the same mode as is available on the C64.
- ISO character set (standard and thin). This mode is new for the X16, and there is no corresponding mode supported by Commodore 8 bit computers. Text is encoded according to ISO-8859-15, making it easier to transfer files to and from modern computers.
- CP437.
- Cyrillic (standard and thin).
- Eastern latin (standard and thin).

On startup, the editor detects the current character set and continues using that.

You may change the character set by pressing Ctrl+E. Press left or right cursor keys to browse the supported character sets, and press Enter to select the new character set you want to use.

3.2 Quote mode

The PETSCII character set contains control codes that are commonly used in BASIC programs. The control codes are, for instance, used to clear the screen and to move the cursor. Normally, it is not possible to type PETSCII control codes in the editor. Control codes may be typed by activating the quote mode (Ctrl+Q). You exit quote mode by pressing ESC.

3.3 Tab stop

The default tab stop width is four spaces.

You may change the width by Ctrl+1–9.

3.4 Auto-indent

The auto-indent feature is used to keep the level of indentation when line breaks are inserted manually or automatically by the word wrap feature.

Auto-indent is turned off when the editor starts. To toggle the feature on or off, press Ctrl+A.

3.5 Word wrap

The editor does not automatically insert line breaks by default. Instead the current line is scrolled horizontally when you reach the right margin.

Automatic word wrap may be toggled on or off by pressing Ctrl+Z. When turned on you are prompted for the column where to wrap.

The word wrap feature works in a very simplified way. When you reach the right margin the editor breaks the line after the previous blank space. If there is no blank space on the line, the line break is inserted at the right margin. The line break position is not recalculated if you delete characters from the line or if you insert new characters at the beginning of the line.

3.6 Text justification

The justify command (Ctrl+J) divides the text buffer into paragraphs and recalculates all line breaks according to the line width specified in the word wrap feature (default 80). The word wrap function need not be enabled to run the justify function.

When auto-indent is turned off a new paragraph is considered to begin

- if two or more consecutive line breaks are found, or
- if a line starts with one or more blank spaces.

If auto-indent is turned on, a new paragraph is considered to begin

- if two or more consecutive line breaks are found,
- if a line contains only blank space characters, or
- if the level of indentation is different from the previous line.

3.7 Cut, copy and uncut

The editor supports cut, copy, and uncut (commonly called paste in other editors).

Parts of the text can be selected for copying or cutting with the keyboard or mouse. A selection is made with the keyboard by holding down Shift while you move the cursor with the arrow keys or the Home or End keys. A selection can similarily be made by holding the left mouse button and move the pointer to the end of the selection before the button is released.

If no text has been selected, the copy (Ctrl+C) and cut (Ctrl+K) commands copy a whole line into the clipboard.

You may do multiple copies or cuts into the clipboard. The clipboard holds a maximum of 3 kB of data.

Uncut (Ctrl+U) copies the content of the clipboard into the text buffer at the cursor position. It is possible to do repeated uncuts. The clipboard is cleared upon the first cut or copy after uncutting.

3.8 Search and replace

The search command (Ctrl+W) lets you search the text buffer for a particular string. The replace function (Ctrl+S) lets you replace it with another string.

Both search and replace are case sensitive. Search starts from the character after the cursor and is forward looking.

If the string you are searching for is found, the cursor is moved to that position.

When replacing a string, you are given the option to only replace the next occurrence or all subsequent occurrences.

3.9 Line break encoding

The currently used line break encoding is displayed at the left edge of the status bar.

The editor's default behavior is to use CR (ASCII 13) as line break when started in one of the PETSCII modes, and LF (ASCII 10) when started in ISO mode.

On opening a file, the editor detects the use of any of the following line break encodings:

- CR – Commodore or old Mac style
- LF – Unix style
- CRLF – Windows style

The line break encoding found in the file is thereafter used by the editor.

You can manually change the active line break encoding by pressing Ctrl+Enter.

4 More on file handling

4.1 Commodore DOS file paths

The editor accepts a full Commodore DOS file path when prompting you for a file name.

A Commodore DOS file path consists of a directory and a file name separated by a colon, for example:

- //MY-APP/:APPNAME.PRG
- /MY-APP/:APPNAME.PRG

The first example starting with double slashes is an absolute path from the root of the SD card. The second example starting with a single slash is a realtive path starting from the current directory.

4.2 The built-in file browser

At the prompt where a file name is entered you may alternatively press Ctrl+T to show the built-in file browser. The file browser is there to make it easier to move between directories and select files.

To select an item in the file browser, first highlight it with the up or down arrow keys, and then press Enter.

If the selected item is a directory, it will be made the new current directory, and its content will be displayed in the file browser.

If not all items fit on one page the listing is ended with "— MORE —". In case the items are spread over several pages, you may go to the next page with PgDn or Ctrl+V and back to a previous page with PgUp or Ctrl+Y.

If there are no more items to show the listing is ended with "— END —".

4.3 Change disk drive device number

By default the file handling functions use device #8. The device number may be changed by pressing Ctrl+D.

4.4 Disk drive commands

You may invoke disk drive commands by pressing Ctrl+I. The raw disk drive command is entered in the prompt that is displayed.

Any valid command may be invoked. Some of the most useful commands are:

- "C:dst=src", copy src file to dst file
- "R:dst=src", rename src file to dst file
- "S:filename", delete filename
- "CD:dirname", change current directory
- "MD:dirname", create directory
- "RD:dirname", remove directory

Just be careful! There is nothing stopping you from deleting files or even formatting the disk.

5 Miscellaneous functions

5.1 Color settings

Both the background and the text color may be changed while using the editor. The program runs in 16 color text mode, so there are 16 background and 16 text colors to choose from.

Ctrl+T cycles through text color options. And Ctrl+B cycles through background colors.

5.2 Text buffer size

The text buffer is stored in banked RAM, which is 512 kB expandable to 2 MB.

The text buffer may not exceed the available banked RAM.

Press Ctrl+M to get information on the remaining memory reserved for the text buffer. It will be reported as blocks free. A block may at most hold 251 characters.

5.3 Programming toolbox

Ctrl+F opens the Programming toolbox.

Currently only one tool, BASLOAD, is supported. By pressing Ctrl+B or only B when the toolbox is active, the open text text file is run through BASLOAD. BASLOAD loads and tokenizes a BASIC program stored in a plain text file so that it can run on the X16. The text file needs to saved to disk before the function is used.

5.4 Custom key scancode handlers

The X16 Kernal allows custom key scancode handlers by changing the vector at $032e-032f. A scancode handler gets the raw IBM key number read from the System Management Controller (SMC). The handler can intercept and change the scancode before it is processed by the Kernal's keyboard functions.

The editor itself uses a scancode handler to keep track of modifier key status. The handler also consumes all Ctrl key events. By default, the editor's scancode handler returns to the Kernal when done, disabling any custom scancode handler that was set up before starting the editor.

You can enable a custom scancode handler by pressing Ctrl+semicolon (;). The editor's scancode handler is called first, and the custom scancode handler is linked in thereafter. The chain of scancode handlers must eventually return to the Kernal for the keyboard to work. This is not within the editor's control after you enable custom scancode handlers.

5.5 User-configurable key bindings

The shortcut key bindings in X16 Edit are user-configurable.

On startup the editor reads custom key bindings from the file X16EDITRC in the root directory of the SD card. If that file is not found, the default key bindings are used.

X16EDITRC is simply a stream of bytes representing the custom shortcuts without any metadata. Each key is represented by the value returned from the Kernal function GETIN.

The values in X16EDITRC are bound to shortcuts in a fixed order, the same order as the shortcuts appear in Appendix A.

If X16EDITRC holds fewer values than there are shortcuts, the editor will use default bindings for the remaining ones. If there are more values in the file than there are shortcuts, the excess is ignored.

To make it a bit easier to setup X16EDITRC, you may use the provided tool (X16EDIT-KEYBINDINGS.PRG).

6 Advanced topics

6.1 Building X16 Edit from source

If you like to build X16 Edit yourself you may download the source code from www.github.com/stefan-b-jakobsson/x16-edit.

You will need the cc65 development tools to build the project. You will also need

the lzsa compression utility.

The project is built using Makefile.

- "make" or "make ram" will build the standard RAM version
- "make rom" builds the ROM version
- "make all" builds both targets

6.2 X16 Edit ROM version

The ROM version of the editor consists of two consequtive 16 kB ROM banks.

As of Kernal version R45, the editor is stored in ROM. If you want to store your own version of the editor in ROM as well you first need to prepare a custom ROM image. When you download the emulator you get the standard rom.bin image. Append X16 Edit to the end of it with the following command (Linux and MacOS):

```
cat rom.bin x16edit-rom-x.x.x.bin > customrom.bin
```

You then need to write the custom ROM image to the actual ROM circuit to use it on real hardware.

A custom ROM image can easily be attached to the emulator with the "-rom" option, for example as follows:

```
x16emu -sdcard sdcard.img -rom customrom.bin
```

The Kernal is in development, and the ROM bank layout may change over time. A user may also store other utilities in ROM. In order to programatically find in which ROM bank X16 Edit is stored, there is an application signature at $FFF0. The signature consists of the text "X16EDIT" followed by three bytes representing the program version (major, minor, patch).

If the editor's first ROM bank is 16, it can be started from BASIC like this:

```
BANK 16,16
SYS $C000
```

If you like to use one of the other entry points that are described in Appendix B, you need to write a startup program stored in RAM. Code samples doing this are also found in Appendix B.

6.3 The X16 Edit API

The X16 Edit API consists of the following four entry points:

- Default entry point, starts the editor with default options and an empty new text buffer
- Load file entry point, starts the editor with default options and then loads the specified text file
- Load file with options 1, starts the editor with custom options and then loads the specified text file
- Load file with options 2, same as previous, in addition supports moving the cursor to a specified line in the opened file

Information on how to call the different entry points and code samples is available in Appendix B.

A List of keyboard shortcuts

This is a complete list of keyboard shortcuts supported by X16 Edit. You may select commands in any of the following ways:

- Ctrl+shortcut key
- Press and release ESC+shortcut key
- A function key from the F-key column

Key	F-key	Description
G	F1	Display built-in help screen
X	F2	Exit from the program
O	F3	Write text buffer to file
R	F5	Open and read a file into the buffer
N	—	Create new text buffer
J	F4	Justify text buffer
Y	F7	Page up
V	F8	Page down
K	—	Cut line or selection to clipboard
C	—	Copy line or selection to clipboard
U	—	Paste (uncut) all content from clipboard
DEL	—	Deletes current line, no copy to clipboard
W	F6	Search and find string in buffer (case sensitive)
S	—	Replace string (case sensitive)
L	—	Goto line number
A	—	Toggle auto indent on/off
Z	—	Toggle word wrap on/off
E	—	Change charset
Q	—	Quote mode
I	—	Invoke DOS command
D	—	Set file storage device number, default is 8
T	—	Cycle through text colors
B	—	Cycle through background colors
M	—	Show memory usage (1 block=251 bytes)
space	—	Insert non-breaking space
1..9	—	Set tab stop width
ENTER	—	Set line break encoding
F	—	Programming toolbox
;	—	Custom scancode handler on/off

B X16 Edit API

B.1 Default entry point

Purpose: Start the editor with default options and an empty new text buffer

Call address: $080D (RAM version), $C000 (ROM version)

Parameters: None

B.2 Load file entry point

Purpose: Start the editor and load a specified text file

Call address: $0810 (RAM version), $C003 (ROM version)

Parameters:

Register	Address	Description
X		First RAM bank used by the program
Y		Last RAM bank used by the program
r0	$02–03	Pointer to file name
r1L	$04	File name length, or 0=no file

If the specified file does not exist, the editor will display an error message. If the file name length is 0, the program will not try to load a text file on startup.

The first and last RAM bank settings control what part of banked RAM is used by the program. This option may be used to reserve the rest of banked RAM for other purposes.

B.3 Load file with options entry point 1

Purpose: Start editor with custom options and then load the specified text file

Call address: $0813 (RAM version), $C006 (ROM version)

Parameters:

Register	Address	Bits	Description
X			First RAM bank used by the program
Y			Last RAM bank used by the program
r0	$02–03		Pointer to file name
r1L	$04		File name length, or 0=no file
r1H	$05	0	Auto indent on/off

r1H	$05	1	Word wrap on/off
r1H	$05	2–7	Unused
r2L	$06		Tab stop width (1..9)
r2H	$07		Word wrap position (10..250)
r3L	$08		Current device number (8..30)
r3H	$09	0–3	Text color
r3H	$09	4–7	Background color
r4L	$0A	0–3	Header text color
r4L	$0A	4–7	Header background color
r4H	$0B	0–3	Status bar text color
r4H	$0B	4–7	Status bar background color

Parameters out of range are silently ignored, and default values are used instead.

Color settings are ignored if both the text and background color is 0.

If the specified file does not exist, the editor will display an error message. If the file name length is 0, the program will not try to load a text file.

The first and last RAM bank settings control what part of banked RAM is used by the program. This option may be used to reserve the rest of banked RAM for other purposes.

B.4 Load file with options entry point 2

Purpose: Start editor with custom options and then load the specified text file. In addition to Load file with options entry point 1, this entry point adds an optional line number whereto the cursor is moved after opening the file.

Call address: $0816 (RAM version), $C009 (ROM version)

Parameters:

Register	Address	Bits	Description
X			First RAM bank used by the program
Y			Last RAM bank used by the program
r0	$02–03		Pointer to file name
r1L	$04		File name length, or 0=no file
r1H	$05	0	Auto indent on/off
r1H	$05	1	Word wrap on/off
r1H	$05	2–7	Unused
r2L	$06		Tab stop width (1..9)
r2H	$07		Word wrap position (10..250)
r3L	$08		Current device number (8..30)
r3H	$09	0–3	Text color
r3H	$09	4–7	Background color

r4L	$0A	0–3	Header text color
r4L	$0A	4–7	Header background color
r4H	$0B	0–3	Status bar text color
r4H	$0B	4–7	Status bar background color
r5–r6L	$0C		Goto line number (24 bits)

Parameters out of range are silently ignored, and default values are used instead.

Color settings are ignored if both the text and background color is 0.

If the specified file does not exist, the editor will display an error message. If the file name length is 0, the program will not try to load a text file.

The first and last RAM bank settings control what part of banked RAM is used by the program. This option may be used to reserve the rest of banked RAM for other purposes.

The cursor is moved to the goto line number (24 bits) on file open. The goto line number is ignored if 0. The cursor is placed on the last line, if the requested line does not exist.

B.5 Code samples for the RAM version

B.5.1 Default entry point

```
jsr $080d  ; No parameters, just call the entry point
rts
```

B.5.2 Load file entry point

```
ldx #$01               ; First RAM bank used by the editor
ldy #$ff               ; And last RAM bank
lda #<fname            ; Pointer to file name (LSB)
sta $02                ; Store in r0L
lda #>fname            ; Pointer to file name (MSB)
sta $03                ; Store in r0H
lda #fname_end-fname   ; File name length
sta $04                ; Store in r1L
jsr $0810              ; Call entry point
rts

fname:
    .byt "mytextfile.txt"
fname_end:
```

B.5.3 Load file with options entry point 1

```
    ldx #$01                ; First RAM bank used by the editor
    ldy #$ff                ; And last RAM bank
    lda #<fname             ; Pointer to file name (LSB)
    sta $02                 ; Store in r0L
    lda #>fname             ; Pointer to file name (MSB)
    sta $03                 ; Store in r0H
    lda #fname_end-fname    ; File name length
    sta $04                 ; Store in r1L
    lda #$01                ; Auto-indent on, word wrap off
    sta $05                 ; Store in r1H
    lda #$04                ; Tab width
    sta $06                 ; Store in r2L
    lda #$28                ; Word wrap position
    sta $07                 ; Store in r2H
    lda #$08                ; Disk drive device number
    sta $08                 ; Store in r3L
    lda #$b1                ; Text white, background light green
    sta $09                 ; Store in r3H (screen color)
    lda #$07                ; Text yellow, background black
    sta $0a                 ; Store in r4L (header color)
    lda #$00                ; Use default color
    sta $0b                 ; Store in r4H (status bar color)
    jsr $0813               ; Call entry point
    rts

fname:
    .byt "mytextfile.txt"
fname_end:
```

B.5.4 Load file with options entry point 2

```
    ldx #$01                ; First RAM bank used by the editor
    ldy #$ff                ; And last RAM bank
    lda #<fname             ; Pointer to file name (LSB)
    sta $02                 ; Store in r0L
    lda #>fname             ; Pointer to file name (MSB)
    sta $03                 ; Store in r0H
    lda #fname_end-fname    ; File name length
    sta $04                 ; Store in r1L
    lda #$01                ; Auto-indent on, word wrap off
    sta $05                 ; Store in r1H
    lda #$04                ; Tab width
    sta $06                 ; Store in r2L
```

```
    lda #$28                ; Word wrap position
    sta $07                 ; Store in r2H
    lda #$08                ; Disk drive device number
    sta $08                 ; Store in r3L
    lda #$b1                ; Text white, background light green
    sta $09                 ; Store in r3H (screen color)
    lda #$07                ; Text yellow, background black
    sta $0a                 ; Store in r4L (header color)
    lda #$00                ; Use default color
    sta $0b                 ; Store in r4H (status bar color)
    lda #$2c                ; Goto line number 300 on file open
    sta $0c
    lda #$01
    sta $0d
    lda #$00
    sta $0e
    jsr $0816               ; Call entry point
    rts

fname:
    .byt "mytextfile.txt"
fname_end:
```

B.6 Code samples for the ROM version

B.6.1 Search ROM banks for X16 Edit

The X16 Edit ROM bank may be identified by the application signature ("X16EDIT") stored at $FFF0. This code sample searches all ROM banks for the signature. If found, the ROM bank is returned in A with carry clear; otherwise carry is set on return.

```
find_me:
    lda $01                 ; Store current ROM bank on stack
    pha
    stz $01                 ; Prepare searching from ROM bank 0
    ldy #$00

scan:
    lda $fff0,y             ; Signature starts at $fff0
    cmp signature,y
    bne next                ; Signature didn't match, check next ROM bank
    iny                     ; Increase char pointer
    cpy #$07                ; Have we got 7 matching chars? If not, keep looking
    bne scan
```

```
    clc                     ; Set C = 0 as indicator X16 Edit was found
    lda $01                 ; Load ROM bank into A
    bra exit

next:
    ldy #$00                ; Reset char pointer
    inc $01                 ; Select next ROM bank
    lda $01
    cmp #$20                ; Have we checked all ROM banks?
    bne scan
    sec                     ; Set C = 1 as indicator X16 Edit was not found

exit:
    plx                     ; Restore original ROM bank
    stx $01
    rts

signature: .byt $58,$31,$36,$45,$44,$49,$54     ; = "X16EDIT"
```

B.6.2 Default entry point

```
    lda $01                 ; Store current ROM bank on stack
    pha
    jsr find_me             ; Search ROM banks
    bcs done                ; Exit if X16 Edit wasn't found
    sta $01                 ; Set ROM bank
    jsr $c000               ; Call entry point
done:
    pla                     ; Restore original ROM bank
    sta $01
    rts
```

B.6.3 Load file entry point

```
    lda $01                 ; Store current ROM bank on stack
    pha
    jsr find_me             ; Search ROM banks
    bcs done                ; Exit if X16 Edit wasn't found
    sta $01                 ; Set ROM bank
    ldx #$01                ; First RAM bank used by the editor
    ldy #$ff                ; And last RAM bank
    lda #<fname             ; Pointer to file name (LSB)
    sta $02                 ; Store in r0L
    lda #>fname             ; Pointer to file name (MSB)
```

```
    sta $03                 ; Store in r0H
    lda #fname_end-fname    ; File name length
    sta $04                 ; Store in r1L
    jsr $c003               ; Call entry point
done:
    pla                     ; Restore original ROM bank
    sta $01
    rts

fname:
    .byt "mytextfile.txt"
fname_end:
```

B.6.4 Load file with options entry point 1

```
    lda $01                 ; Store current ROM bank on stack
    pha
    jsr find_me             ; Search ROM banks
    bcs done                ; Exit if X16 Edit wasn't found
    sta $01                 ; Set ROM bank

    ldx #$01                ; First RAM bank used by the editor
    ldy #$ff                ; And last RAM bank
    lda #<fname             ; Pointer to file name (LSB)
    sta $02                 ; Store in r0L
    lda #>fname             ; Pointer to file name (MSB)
    sta $03                 ; Store in r0H
    lda #fname_end-fname    ; File name length
    sta $04                 ; Store in r1L
    lda #$01                ; Auto-indent on, word wrap off
    sta $05                 ; Store in r1H
    lda #$04                ; Tab width
    sta $06                 ; Store in r2L
    lda #$28                ; Word wrap position
    sta $07                 ; Store in r2H
    lda #$08                ; Disk drive device number
    sta $08                 ; Store in r3L
    lda #$b1                ; Text white, background light green
    sta $09                 ; Store in r3H (screen color)
    lda #$07                ; Text yellow, background black
    sta $0a                 ; Store in r4L (header color)
    lda #$00                ; Use default color
    sta $0b                 ; Store in r4H (status bar color)
    jsr $c006               ; Call entry point
done:
```

```
    pla                     ; Restore original ROM bank
    sta $01
    rts

fname:
    .byt "mytextfile.txt"
fname_end:
```

B.6.5 Load file with options entry point 2

```
    lda $01                 ; Store current ROM bank on stack
    pha
    jsr find_me             ; Search ROM banks
    bcs done                ; Exit if X16 Edit wasn't found
    sta $01                 ; Set ROM bank

    ldx #$01                ; First RAM bank used by the editor
    ldy #$ff                ; And last RAM bank
    lda #<fname             ; Pointer to file name (LSB)
    sta $02                 ; Store in r0L
    lda #>fname             ; Pointer to file name (MSB)
    sta $03                 ; Store in r0H
    lda #fname_end-fname    ; File name length
    sta $04                 ; Store in r1L
    lda #$01                ; Auto-indent on, word wrap off
    sta $05                 ; Store in r1H
    lda #$04                ; Tab width
    sta $06                 ; Store in r2L
    lda #$28                ; Word wrap position
    sta $07                 ; Store in r2H
    lda #$08                ; Disk drive device number
    sta $08                 ; Store in r3L
    lda #$b1                ; Text white, background light green
    sta $09                 ; Store in r3H (screen color)
    lda #$07                ; Text yellow, background black
    sta $0a                 ; Store in r4L (header color)
    lda #$00                ; Use default color
    sta $0b                 ; Store in r4H (status bar color)
    lda #$2c                ; Goto line 300 on file open
    sta $0c
    lda #$01
    sta $0d
    lda #$00
    sta $0e
    jsr $c009               ; Call entry point
```

```
done:
    pla                     ; Restore original ROM bank
    sta $01
    rts

fname:
    .byt "mytextfile.txt"
fname_end:
```

C License