Revision: Final Draft March, 2021

HP-71B MultiMod ROM Emulator

User Manual

Revision: Final Draft March, 2021

Every effort has been made to insure the accuracy of the information contained in this document. Descriptions of hardware behavior and operation my be updated from time to time as new software releases become available.

Copyright © 2021, Mark A. Fleming. All rights reserved.

Notice:

“HP-71B” and “HP” are registered trademarks of Hewlett-Packard Inc. or their derivative corporate trademark holder.

Acknowledgments
I wish to thank J-F Garnier, Tony Nixon, and Diego Díaz for their help and advice in making this ROM emulator possible.

Introduction

The HP-71B MultiMod ROM Emulator gives you the ability to plug in one or more ROM modules on your HP-71B. This is accomplished using a modern microcontroller to emulate the original ROM chip hardware. Using a microcontroller in place of an FPGA hardware implementation allows added programmable functionality that is used to provide a platform-independent means of loading and configuring ROM images. With just a serial connection and suitable terminal emulator like TeraTerm you can quickly add expensive and hard to find ROM modules to your 71B.

Features

- PIC18F27K40 Microcontroller, 64 MHz (16 MIPS)
- 128 KB Program Flash Memory, 120 KB usable
- Operating voltage 3.3 V to 5.5 V, absolute maximum rating 6.5 V
- 7 mA operating current, 25µA standby
- Power jumper to remove system bus load
- Seven 16 KB “chips” for ROM images
- Serial port interface for configuration and ROM image load
- Write-protected boot loader for software update
- ICSP connector for direct programming

Contents

WARNING
Use of Alkaline batteries is not recommended, and may reduce the operating life of the PIC processor! Use rechargeable cells such as Eneloop® NiMH or similar.

Installation

You should follow the usual safety practice when handling static sensitive electronic devices, particularly your HP-71B. Keep yourself grounded when installing or removing the MultiMod board. Handle the MultiMod board by its edges whenever possible, and avoid touching the contact pins in the 71B card reader well. An optional case[1] for the MultiMod board shields the board and makes it easier to insert and remove.

I strongly recommend removing your batteries first, after backing up any content. Though the HP-71B is designed for "hot" peripheral insert/remove, a bare board without an enclosure for pin alignment could result in a Memory Lost event. Remove the back cover over the card reader well, and the plastic header insert if present. Remove the J2 power jumper from the MultiMod board as shown in Figure 1 so that the PIC processor will not present a load on the 71B bus during installation. Insert the MultiMod board into the card reader well, aligning the pins in the well with the circular contacts on the board. The board should be pressed firmly down to fully engage the pins, but avoid pressing directly on the contacts themselves with your fingers. A plastic tool is a better option for pressing on or close to these contacts. Once the board is firmly fitted you can replace the J2 power jumper, batteries, and the back cover.

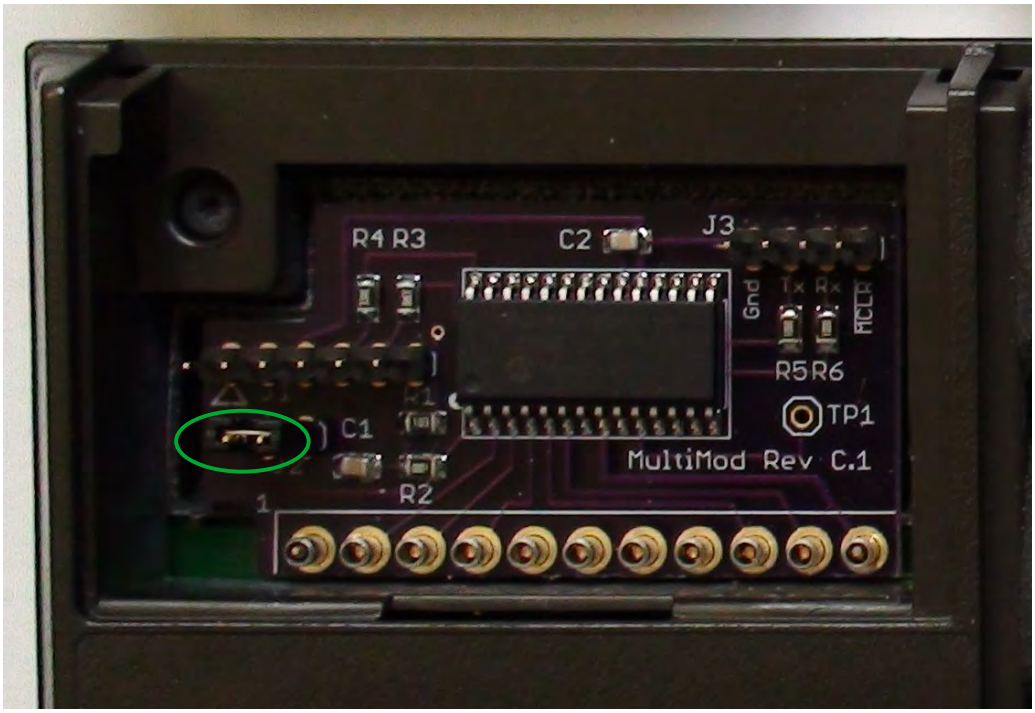


Figure 1: Power Jumper

The MultiMod emulator comes preloaded with a sample ROM configuration. Turn your HP-71B on and type the `VER$` command. You should see your current configuration. The MultiMod emulator does

1 Contact Dan Simpson, `dan` at `azgraphixnorth` '.' `com` for ordering details.

not plug in its ROMs by default. You need to tell it to plug in or remove its ROMs using the POKE command. Try the command `POKE "2C000","1"` and then turn the 71B off and back on. Now see what is displayed by the `VER$` command. Congratulations, you now have the Forth ROM, and J-F's marvelous Math 2B and JPC ROMs installed on your HP-71B!

Overview

The HP-71B MultiMod ROM Emulator can be thought of as seven 16KB flash memory chips, each of which can hold ROM image data of your choosing. A chip can hold a single 16KB ROM, or chips can be combined to hold 32KB, 48KB, or 64KB ROM images. Two chips can even hold a hard configured ROM image located at the 71B memory address E0000h. More than one ROM can be present – as many ROMs can be enumerated in a port as will fit in the seven available chips.

This manual section is focused on how to go about planning the layout of ROM images in the MultiMod emulator. A little background on how ROMs are discovered and configured during startup is provided as part of showing you how to install a set of ROMs of your own. Later sections walk you through the mechanics of loading ROM images and configuring the chips for ROM discovery.

> The PIC controller actually has eight 16KB Program Flash Memory (PFM) blocks numbered 0 to 7, but block 0 holds the write-protected boot block and the emulator software itself. **As a convention, we'll use the term "chip" for these flash memory blocks throughout this section**. More hardware-specific detail will follow in the programming section.

ROM Control

Each time the HP-71B turns on it probes each port to see what is installed. It does so in two stages, first sending a series of ID commands that ROM or RAM chips respond to with their hardware configuration strings. During this phase each chip is assigned a default address. In the second stage each chip is assigned a final address that it uses throughout the 71B operating session. Between configuration stages the 71B probes the bus to see if a hard configured ROM is present at address E0000h and marks that space as occupied when deciding where to put ROM or RAM chips during the second configuration stage.

You need to plug/unplug and cycle power each time you change your ROM configuration so that the 71B discovers these changes, just as you would when inserting or removing a physical ROM module. Plugging and unplugging the MultiMod emulator ROMs is done using the POKE command. Poking a "0" to address "2C000" will unplug the ROMs after cycling power. Poking a "1" will plug the ROMs back in after a power cycle. If you Poke the value "3" then a hidden 8K ROM, if present, will also be enumerated. Poking a "1" again will unplug the hidden ROM while leaving the other ROMs plugged in.

Now you'll next need to understand how to lay out your chosen ROM image data in the PIC flash memory and how to specify the way in which the ROM chips identify themselves to the 71B during the startup process.

Planning Your Layout

The first task of course is deciding what ROMs you want the MultiMod emulator to host. You can use any combination of ROM sizes as long as the sum of sizes does not exceed the total size of the seven 16KB chips. All ROM images must fit in an integral number of 16KB chips, that is, a 16KB ROM or smaller will fit into a whole 16KB chip; all ROMs start at the beginning of a 16KB chip, and ROMs larger than 16KB must occupy adjacent 16KB chips. This means whole ROM images can't be scattered about the chips, nor can they overlap within a chip. Finally, if you want to provide a hard configured ROM image, then that image **must** go into the last two chips, 6 and 7.

Simple enough. As a working example, let's use a configuration similar to that which comes with the MultiMod emulator. For this configuration we'll need the following files.

ROM Image File	ROM Size	# of 16KB Chips
HP-82478A_FORTH-ASSEMBLER_ROM.BIN	16KB	1
HP-82478A_FORTH-ASSEMBLER_HRD-FIX.BIN	32KB	2
MATH2B.BIN	32KB	2
JPC05.BIN	32KB	2

Note that the number of chips used equals the number of chips available. Not by accident! The next task is to map the ROM images we have to the seven available chips. The mapping process assigns ROM image content[2] to one or more of the seven chips using the rules first specified in this section. Let's assign the above images to chips in the order of image appearance, keeping in mind the rule for assigning a hard configured ROM image. The assignment appears below.

ROM Image	Chip Number
Forth Assembler (16K)	1
Math 2B (32K)	2
Math 2B (32K)	3
JPC05 (32K)	4
JPC05 (32K)	5
Forth Hard ROM (32K)	6
Forth Hard ROM (32K)	7

2 We'll see later the importance of distinguishing between ROM *content* in flash memory versus the *structure* of ROMs held in memory.

The 16KB ROM image goes into the first chip and the 32KB ROM images go into adjacent 16KB chips. The hard configured Forth ROM image goes in chip numbers 6 and 7. If you are not using a hard ROM then those two chips are available for any other ROM(s) you might want.

Another thing to keep in mind while laying out your ROM images is that ROMs are enumerated in the order they appear in the table. ROMs will appear in the above listed order when using the SHOW PORT command. So the Forth ROM will appear as `PORT(5.00)`, the Math ROM as `PORT(5.01)` and the JPC ROM as `PORT(5.02)`. Try using the command `CAT :PORT(5.02)` to see the content of the JPC ROM.

The above *structure* is suitable for other ROM arrangements. For example, if you wish the JPC ROM to appear first in your version command list, you can simply swap the ROM *content* between chip sets 2&3 and 4&5. An even more interesting arrangement if you have a version 1A of the HP-IB module, is to override its built-in ROM by placing the 16KB HP-IB 1B version just before the JPC ROM image. One such arrangement is shown here.

ROM Image	**Chip Number**
HP-IB 1B (16K)	1
JPC05 (32K)	2
JPC05 (32K)	3
Math 2B (32K)	4
Math 2B (32K)	5
Finance ROM (16K)	6
Text Editor ROM (16K)	7

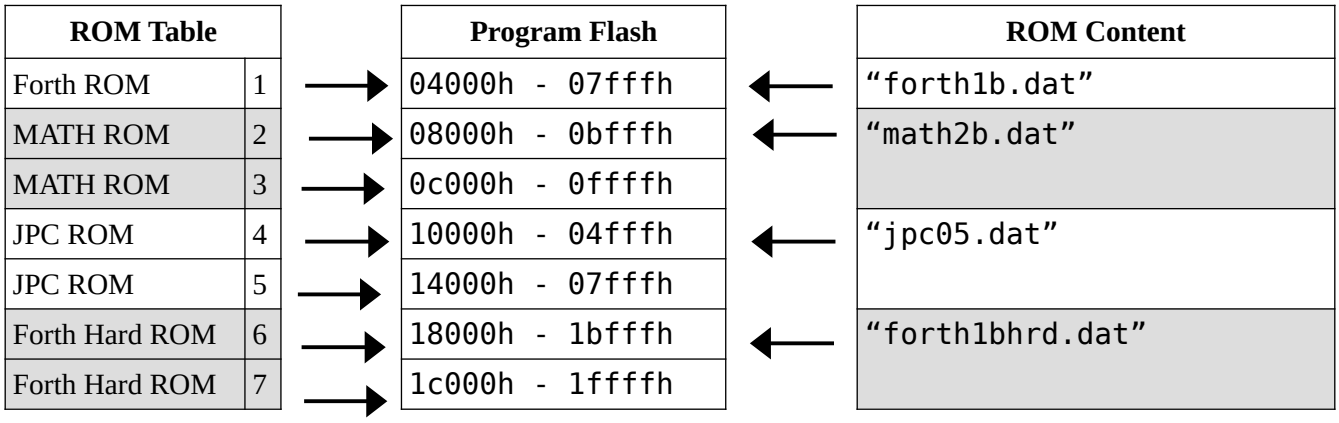
This is all the information needed before programming your own ROM emulator arrangement. Use the scratch table below to layout the ROMs you need.

ROM Image	**Chip Number**
	1
	2
	3
	4
	5
	6
	7

MultiMod Configuration Details

When configuring the MultiMod it's important to differentiate between the *structure* and the *content* of ROMs in the MultiMod. The structure outlined in the previous section consisted of a 16K ROM, two 32K ROMs, and a hard ROM, all residing in a set of seven 16KB chips. That structure is stored in an internal table that can be altered using serial monitor commands.

The ROM image *content* of a given chip is actually stored in the seven[3] 16KB Program Flash Memory blocks of the internal microcontroller as shown in the memory layout below. When you specify a chip number, you are pointing to an internal flash memory block.



The next section will illustrate how to use the serial monitor built into the MultiMod ROM Emulator software to configure the number and size of the ROMs in your MultiMod, and how to upload ROM images to the MultiMod flash memory. You will be working with an internal *ROM Configuration Table* that provides information on the ROMs you create in your MultiMod. The internal table contains the configuration string each ROM chip returns for an ID command (shown in bold) along with other control values. A illustrative representation of the table is shown next.

ROM Image Filename	**Size**	**Resv**	**Type**	**Class**	**Last**	Flag	Addr	Chip	Line
		0x00	0x01	0x00			0x00		1
		0x00	0x01	0x00			0x00		2
		0x00	0x01	0x00			0x00		3
		0x00	0x01	0x00			0x00		4
		0x00	0x01	0x00			0x00		5
		0x00	0x01	0x00			0x00		6
		0x00	0x01	0x00			0x00		7
MMIO Address	0x00	0x00	0x00	0x0C	0x02	0x00	0x00	0x00	8

Field	Data	Desc
Size	0x0A	16KB
	0x09	32KB
	0x08	64KB
Last	0x00	No
	0x08	Yes
Flag	0x00	Cont.
	0x01	Stop
	0x02	H-R

3 As we'll see later, half of program flash memory block 0 is available for a "hidden" 8K ROM.

Serial Monitor Update Procedure

You can use an ordinary USB to Serial cable, either 5V (preferred) or 3.3V logic level. Series resistors in the transmit and receive lines protect a 3.3V interface. Configure your terminal emulator for 19200 baud, 8-bit data, no parity, 1 stop bit, and XON/XOFF flow control. The monitor uses a Carriage Return for line termination so configure your terminal to convert CR to CR-LF (“Auto” in TeraTerm for the Receive Setting). Connect the serial lines to the four pin header J3 as shown in Figure 3.

Serial Connector Pin Assignment				
Pin	Ground	Rx	Tx	$\overline{\text{MCLR}}$
Serial Cable	Ground	Tx	Rx	RTS (optional)

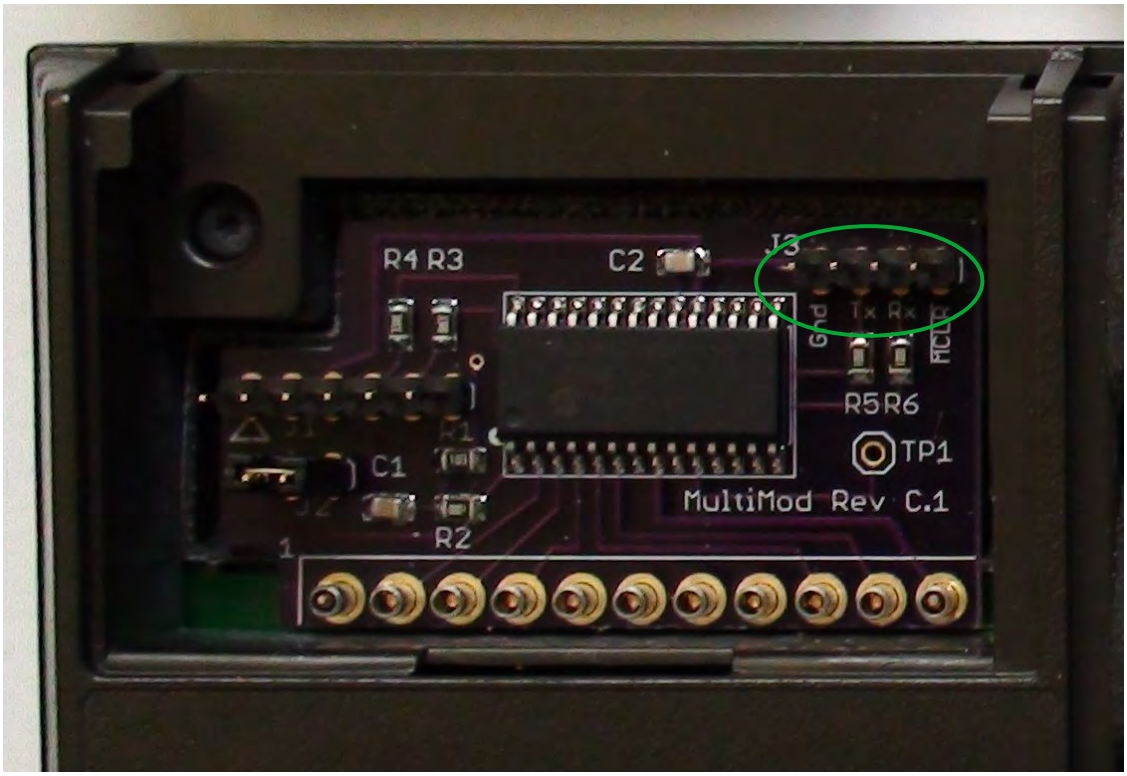


Figure 2: Serial Port Connector

Before starting a serial monitor session it is best to unplug your ROMs using the POKE command. Your HP-71B should be off when communicating with the MultiMod emulator since the 71B bus is ignored while the serial monitor is active. Press the Return key to transfer control to the serial monitor. It should respond with the message `Press ? For Help`. Let’s have a look at the available commands. Press ‘?’ for a quick list of commands.

Don’t forget to unplug ROMs whenever you make changes to ROM structure or content in the MultiMod and cycle power afterwards. Failure to do so could result in confusion on the part of your HP-71B and the likelihood it won’t recognize your changes. If so unplug with `POKE “2C000”,“0”` cycle power and plug your ROMs back in.

Command	**Argument(s)**	**Description**
ROM	Configuration or none	Configures a given table entry; No arguments list all entries
ERASE	Block # (0-7)	Erases the flash memory of the specified flash block
IMAGE	Block # (0-7)	Uploads a ROM image, starting in the specified flash block
LAST	Entry # (1-7)	Indicates the last table entry to be enumerated
HARD	Y or N	Indicates whether a hard ROM is present
COMMIT	Confirm Y or N	Commits a configuration to permanent flash
PLUG	Y or N	Sets the flag to plug or unplug ROMs when the 71B turns on
QUIT	none	Exit serial monitor, return to normal processing

Below is an arrangement of ROM table entries that contains some of the default content of your MultiMod ROM emulator. Each table entry points to a 16KB “chip” or program flash memory block. This table is used to represent the *structure* of ROMs stored in the MultiMod flash memory.

ROM Image Filename	**Size**	**Resv**	**Type**	**Class**	**Last**	Flag	Addr	Chip	Line
Forth1B	0x0A	0x00	0x01	0x00	0x08	0x00	0x00	1	1
Math2B	0x0A	0x00	0x01	0x00	0x00	0x00	0x00	2	2
Math2B	0x0A	0x00	0x01	0x00	0x08	0x00	0x00	3	3
JPC05	0x0A	0x00	0x01	0x00	0x00	0x00	0x00	4	4
JPC05	0x0A	0x00	0x01	0x00	0x08	0x01	0x00	5	5
Forth1B hard	0x0A	0x00	0x01	0x00	0x08	0x02	0x00	6	6
Forth1B hard	0x0A	0x00	0x01	0x00	0x08	0x02	0x00	7	7
MMIO Address	0x00	0x00	0x00	0x0C	0x02	0x00	0x00	0x00	8

Field	Data	Desc
Size	0x0A	16KB
	0x09	32KB
	0x08	64KB
Last	0x00	No
	0x08	Yes
Flag	0x00	Cont.
	0x01	Stop
	0x02	H-R

Let’s walk through this example to familiarize ourselves with the serial monitor commands before giving a detailed definition for each. Connect your USB serial cable as shown and tap return. After you see the opening prompt, type “r” followed by return. An important feature to note is that the serial monitor is not case sensitive and that it uses command auto-completion. When you typed “r” the monitor echoed ROM plus a space. Auto-completion saves time and avoids typing errors. You would see the output below.

Figure 3: ROM Table Listing

The first column in the listing is the table entry number, referred to as “Line” in the above table. The second column shows the size of the chip that the table entry represents (“Size”). The third column shows which program flash memory block the table entry points to for ROM image content (“Chip”). The fourth column indicates if the table entry (“Last”) holds a complete ROM image or is part of a sequence of chips making up a complete ROM entry.

Notice that chip 5 is marked LAST to indicate it is the last ROM that will be enumerated during startup. Since we have a Forth hard configured ROM present, the last two table entries are marked HARD. If you were now to type the command `LAST 3` and turn your 71B back on and plug the ROMs in, only the first two ROMs would be enumerated. The VER$ command would show that the Forth and Math 2B ROMs were present but the JPC ROM would no longer be present.

If we were creating this ROM structure, the commands would be as follows. Note the difference between what is displayed as you type and the actual keystrokes you enter. That’s auto-completion!

Keystrokes	**Command Display**	**Description**
`r·1·1·1`	`ROM 1 16K 1`	A single 16K ROM in one Chip
`r·2·c·2`	`ROM 2 CHIP 2`	The first of a two-chip sequence for a 32K ROM
`r·3·1·3`	`ROM 3 16K 3`	The last 16K chip in a 32K ROM sequence
`r·4·c·4`	`ROM 4 CHIP 4`	The first of a two-chip sequence for a 32K ROM
`r·5·1·5`	`ROM 5 16K 5`	The last 16K chip in a 32K ROM sequence
`r·6·1·6`	`ROM 6 16K 6`	Two chips for a hard-configured ROM, the ID
`r·7·1·7`	`ROM 7 16K 7`	String is not relevant for hard ROM
`l·5`	`LAST 5`	The last table entry is 5
`h·y`	`HARD YES`	There is a hard ROM in entries 6 & 7

The serial monitor is not case sensitive so it is perfectly acceptable to use lowercase letters as shown in the keystrokes column. If we were creating a ROM structure for the first time then the ROM content would need to be uploaded. To show that process, why not replace the Forth ROM with the HP-41 Translator ROM? This example will illustrate how you might use the same ROM *structure* to hold different ROM *content*. The process is quite simple; Erase blocks 1, 6, and 7 then upload your content. The commands are as follows.

Keystrokes	**Command Display**	**Description**
`e·1`	`ERASE 1`	Erase flash memory block 1 (“chip” 1)
`e·6`	`ERASE 6`	Erase flash memory block 6
`e·7`	`ERASE 7`	Erase flash memory block 7
`i·1`	`IMAGE 1`	Upload ROM image to block 1
`i·6`	`IMAGE 6`	Upload ROM image to block 6, continuing to 7

So how do you actually upload ROM content? When you type the Image command followed by the flash block (“chip”), the serial monitor responds with a carriage return and waits for you to send ASCII hex strings that encode the ROM binary content. Hex values are not case sensitive and the monitor will ignore non-hex characters such as space or comma. The Hex strings are broken into lines marked by carriage return or linefeed. The monitor will buffer the data until the end-of-line (CR or LF) character is received and then write the content to program flash. The process is terminated when the monitor sees multiple CR characters. The monitor will write data to the block you specify and continue with the next block if there is more data sent. This is what happens when you send the 32KB hard ROM for the HP-41 Translator to blocks 6 and 7.

Where does the data come from? You should already have a number of .DAT files containing the data for the publicly available ROMs in .BIN format. You can create a DAT file using the Python program as shown.

```
python3 bin2dat.py HP-82478A_FORTH-ASSEMBLER_ROM.BIN forth.dat
```

You send the DAT file content without any sort of transmission protocol other than the XON/XOFF handshake. In the TeraTerm terminal emulator you’ll find this as “Send File” in the File menu. Once the file is uploaded hit the Return key a couple of times.

Turn your HP-71B on, plug the ROMs back in and cycle power. After the VER$ command try typing `HP41` and see what happens. In the HP-41 emulator type `BASIC` to return to the regular BASIC prompt, then unplug ROMs before turning off the 71B. This illustrates how ROM *content* can be uploaded to a particular ROM *structure* in the MultiMod. Try for yourself loading other 32K ROM images in the two available 32K ROMs, or a 16K ROM in the single 16K ROM structure.

To further illustrate the difference between content and structure, let’s try to create the following structure.

ROM Image Filename	**Size**	**Resv**	**Type**	**Class**	**Last**	Flag	Addr	Chip	Line
Forth1B	0x0A	0x00	0x01	0x00	0x08	0x00	0x00	1	1
Math2B	0x09	0x00	0x01	0x00	0x08	0x00	0x00	2	2
JPC05	0x09	0x00	0x01	0x00	0x08	0x01	0x00	4	3
empty	0x0A	0x00	0x01	0x00	0x00	0x00	0x00	4	4
empty	0x0A	0x00	0x01	0x00	0x08	0x00	0x00	5	5
Forth1B hard	0x0A	0x00	0x01	0x00	0x00	0x02	0x00	6	6
Forth1B hard	0x0A	0x00	0x01	0x00	0x00	0x02	0x00	7	7
MMIO Address	0x00	0x00	0x00	0x0C	0x02	0x00	0x00	0x00	8

Field	Data	Desc
Size	0x0A	16KB
	0x09	32KB
Last	0x08	64KB
	0x00	No
	0x08	Yes
Flag	0x00	Cont.
	0x01	Stop
	0x02	H-R

Here we have the same arrangement of a 16K ROM and two 32K ROMs with a hard configured ROM. The difference lies in the fact that we're telling the HP-71B that the 32K ROMs are held in a single 32KB chip. Two of the internal 16KB program flash blocks ("chips") are still needed to hold ROM content as can be seen by the chip number assigned to the two table entries. The content location of the two 32K ROMs is unchanged. The Last flag changes to indicate ROM enumeration ends with the third ROM table entry. So here are the commands used to create this structure.

Keystrokes	**Command Display**	**Description**
`r·1·1·1`	`ROM 1 16K 1`	A single 16K ROM in one Chip
`r·2·3·2`	`ROM 2 32K 2`	A single 32K ROM in a 32K Chip (2 flash blocks)
`r·3·3·4`	`ROM 3 32K 4`	A single 32K ROM in a 32K Chip (2 flash blocks)
`r·6·1·6`	`ROM 6 16K 6`	A single 16K ROM in one Chip
`r·7·1·7`	`ROM 7 16K 7`	A single 16K ROM in one Chip
`l·3`	`LAST 3`	Last table entry is 3
`h·y`	`HARD YES`	There is a hard ROM in entries 6 & 7

If you turn the 71B back on and plug the ROMs in again you'll see the original configuration with Forth, Math and JPC ROMs. Nothing has changed as far as *content* and its location in program flash memory. Only the *structure* of ROMs contained in the MultiMod has changed.

To finish this extended example, let's access the so-called "hidden" ROM. As mentioned earlier, block 0 of program flash memory contains the write-protected boot loader and the ROM Emulator software. There is sufficient room left over to hold 8 KB of 71B ROM image data. For technical reasons this must be represented in the configuration table as a 16KB ROM. You can load the content with the IMAGE command but be warned, uploading more that 8 KB of data will overwrite what you have in

block 1 (or worse, if you fail to erase the block!) Try the following command sequence to create and upload data to block 0.

Keystrokes	Command Display	Description
r·4·1·0	ROM 4 16K 0	A single 16K ROM in block 0
l·4	LAST 4	Last table entry is 4
e·0	ERASE 0	Erase flash memory block 0
i·0	IMAGE 0	Upload ROM image to block 0

Use the `ulib.dat` file to upload the content of J-F's ULIB LEX library ROM. Turn the 71B back on, plug in ROMs and cycle power. You now have J-F's Ultimate ROM!

Serial Monitor Commands

Hopefully the previous extended example illustrates how you can configure MultiMod to support whatever ROMs you'd like to have at hand on your 71B. The serial monitor makes it easy to change ROM content by, for example, replacing the 32KB JPC ROM with the 32KB AmpiStats ROM. Or you can alter the structure by turning one of the 32 KB ROM entries into two 16 KB entries and uploading the Finance and Text Editor ROMs.

Let's have a detailed look at each of the available commands and the parameters each use.

Command Nomenclature

<entry>	ROM Enumeration Table entry number; Value 1 – 7
<size>	ROM size; 16K, 32K, 64K or CHIP (16K)
<chip>	Flash chip number; Value 0 - 7
[…]	Optional parameters

> You can use the ESCAPE key to cancel a command. If your terminal emulator interprets escape sequences, you may need to press the RETURN key after ESCAPE.

`ROM [<entry> <size> <chip>]`

The ROM command by itself with no parameters will list all of the entries in the ROM Configuration Table. This is useful as a check following any changes you have made. With three parameters, the ROM Enumeration Table is updated with the values given. Be sure any ROMs you define don't overlap in program flash memory, and be sure to mark the last table entry using the LAST command.

Any <size> value used sets the End-of-Module flag (EOM) telling the HP-71B that any chip that follows is another flash module. For a sequence of 16K chips, use one or more CHIP size and finish the sequence with a 16K size that sets the EOM flag. Terminating a sequence with any other size will likely yield odd results.

`LAST <entry>`

The LAST command marks the end of the defined ROMs that the MultiMod will enumerate on startup. Failure to mark the last entry could cause garbage to be presented to the 71B O/S, which usually will ignore anything that doesn't have a valid ROM signature. In the worse case however, the O/S has been known to get stuck in an infinite enumeration loop and never show the command prompt. Such a condition **may** require an INIT command **after** you remove the MultiMod power jumper, so take care to examine your new configuration before you try it!

`HARD Y|N`

The HARD command will mark the ROM Configuration Table's last two entries as having or not having a hard-configured ROM that will appear at address E0000h. Enter 'Y' or 'N' (case ignored) to set or clear this feature of the MultiMod. The 71B will check for the existence of a ROM in that location by reading the first nibble but will do nothing to validate the ROM content. Failure to provide the hard ROM to the Forth Application ROM will result in an interesting crash when the app is invoked.

`ERASE <chip>`

The ERASE command is needed to prepare a "chip" (program flash memory block) for new data. Failure to erase a non-empty flash block before using the IMAGE command will result in garbage. The only appropriate warning here is to be sure you are erasing the right block. Hit Escape to back out.

`IMAGE <chip>`

The IMAGE command is used to send a ROM binary content in an ASCII hexadecimal format. Characters 'A' through 'F' can be in either case, and any non-hex character (spaces or commas, for instance) is ignored. The hex strings sent must be broken up into lines terminated by either CR or LF (or CR-LF). The data that is sent is converted to binary and stored in a buffer, and that buffered data is written to flash on receipt of the end of line character.

Warning: Failure to limit the amount of data sent in a line will cause the buffer to overwrite system variables and crash the emulator. Failure to use the XON/XOFF protocol when sending data is likely to put the serial monitor in an unresponsive state. Both conditions will likely require resetting the MultiMod as outlined in the Recovery section in the Appendix.

`COMMIT Y|N`

The COMMIT command will write the ROM Configuration Table that resides in RAM back to flash. When the MultiMod first powers up or is reset, it copies a table held in program flash memory into RAM where it is subsequently used each time the MultiMod responds to initialization. The serial monitor alters this temporary copy in RAM. If that copy should prove to be seriously invalid, it can be restored to its initial state by resetting the MultiMod.

The COMMIT command rewrites the table in flash with the (altered) content held in RAM. The command should **never** be used without testing the configuration first. Remember that the ROM *content* is kept in program flash memory and survives a reset. If you want the *structure* to survive a reset, use the COMMIT command to make it permanent.

`QUIT`

The QUIT command exits the serial monitor and returns control to the normal ROM emulator functionality. **Always use Quit to exit the serial monitor** or the ROM Emulator will not respond when you use your 71B.

`PLUG Y|N`

The PLUG command allows you to plug in (Y) or unplug (N) the ROMs. The command can be used to unplug ROMs if you forgot to do so before entering the serial monitor. If you did unplug them, you can plug them back in, saving time when you turn the 71B back on.

A Note On Error Checking

A certain amount of error checking and data validation is done in the serial monitor and in the emulation code that uses the data altered by the monitor. The limitations of available program flash memory dictate an approach which assumes the user of the monitor understands what is being done and is adept at making appropriate changes. Some effort is made to simplify usage, through auto-completion for instance, but little is done to support a casual user. That support is rightfully delegated to an external program tool that uses the serial monitor commands to alter configuration, erase blocks, and upload image data. A GUI interface to the MultiMod could be written in Python or Java so that the user need only drag and drop BIN files to a panel, move them about to establish enumeration order, then program them into a MultiMod with a single button.

Appendices

Appendix A. Operation, Update, and Recovery

Operation

The Microchip PIC processor used in the MultiMod is rated for normal operation between 2.3V and 5.5V, requiring at least 3.0V to function at its designed operating frequency. The absolute maximum supply voltage, beyond which damage will occur, is 6.5V. What does this mean as far as using the MultiMod in an HP-71B? The following table of measured voltage at the card reader pins should help.

Condition	MultiMod Out	MultiMod In
AC Adapter, no batteries	5.55V	5.51V
Freshly charged NiMH batteries	5.38V	5.26V
Alkaline batteries (fresh Everyready Gold)	6.04V	Not Measured
AC Adapter, NiMH batteries	5.55V	5.49V
AC Adapter, Alkaline batteries	6.04V	Not Measured

The supply voltage when an HP 82069D AC Adapter is used tends to be the higher of the voltage from batteries or adapter. With any of the measured values with the MultiMod in, the supply voltage seen by the PIC processor is no more than a few percent higher than its recommended maximum operating voltage. Does this mean you can use regular alkaline cells in your 71B with a MultiMod installed?

Perhaps, but as they say, Your Mileage May Vary. These measures could change somewhat with the load from installed front port & HP-IL modules, and from different brands of alkaline cells, hence the warning to stick with rechargeable cells versus disposable ones. There is headroom between the 5.5V recommended operating maximum versus the 6.5V absolute maximum supply voltage, but use of alkaline batteries might possibly shorten the processor operating life. Only you can find out for sure!

The good news is, long term operation using the AC adapter with/without NiMH cells is perfectly OK.

The MultiMod consumes approximately 7mA when operating and 25µA after entering sleep mode several minutes after the HP-71B is turned off. According to the HP-71B Owner’s Manual, the 71B 30µA when off, so with the MultiMod inserted and in sleep mode, power consumption roughly doubles. The 71B consumes approximately 0.75mA when on but not running and 10mA when running a program. The presence of the MultiMod would raise Idle current to 7.75mA when the 71B is on, and 17mA when running a program. Reduced battery life can be expected, but when the MultiMod isn’t needed its power jumper can be removed to eliminate its impact on battery life.

Recovery

What do you do if something goes wrong with a configuration you've loaded? In most cases you just won't see the ROMs you expect in the output of the version command. If altering the configuration doesn't help, then it's possible the PIC processor is hung. You can try turning the 71B off, remove the J2 jumper for at least 5 seconds[4] before replacing the jumper, then turn the 71B back on. A more reliable way of resetting the processor is to briefly short the Ground and $\overline{\text{MCLR}}$ pins. This is done by shorting the leftmost and rightmost pins of the J3 serial connector.

If the PIC processor in the MultiMod should "run wild" then there is a very real chance of bus collisions as both the PIC and Saturn processors try to drive the bus. That will definitely cause the Saturn processor to crash. Hard. Pressing the ON button in this scenario will have no effect, nor will pressing the INIT key sequence (ON and '/' at the same time). I've only ever observed this behavior during development, but it's still worth mentioning. Recovery from this scenario, if a direct reset doesn't work, would involve removing the batteries from the 71B for a while.

The only way to muddle things up so that the operation of the MultiMod is compromised[5] would be to overwrite the emulator code itself. This can only be done using the boot loader or a PICKit3 programmer, and it would likely need to be more than a little intentional.

If the emulator software is no longer functional then you can't expect the application to transfer control to the boot loader client when it sees the Break condition. You'll need force the boot loader to respond. First, remove the J2 jumper. Now connect the serial cable, run the boot loader client, and use it to assert Break. Put the J2 jumper back in so that the PIC processor goes through the boot loader during the power-up reset sequence. The boot loader will respond to the Break by not transferring control to the emulator application. You can then press the button to connect to the boot loader and restore the emulator software from the source Hex file.

Update

You'll find a Hex file named `MULTIMODvxxx.HEX` in the `src` subdirectory. The 'xxx' indicates the version number of the MultiMod emulator software. Occasional updates will be available with newer version numbers. Once enhancement and other changes are greatly reduced in frequency, the source itself will be released.

Updating the system software involves loading the latest system Hex file into the boot loader client and writing the software to the processor flash memory. Follow the serial monitor directions for connecting a USB to serial cable to the MultiMod. Launch the "AN1310ui.exe" program and set its configuration as shown.

4 A 0.1 µF capacitor needs to discharge.

5 Someone somewhere will no doubt prove me wrong by finding some *other* way to munge things up. Let me know how.

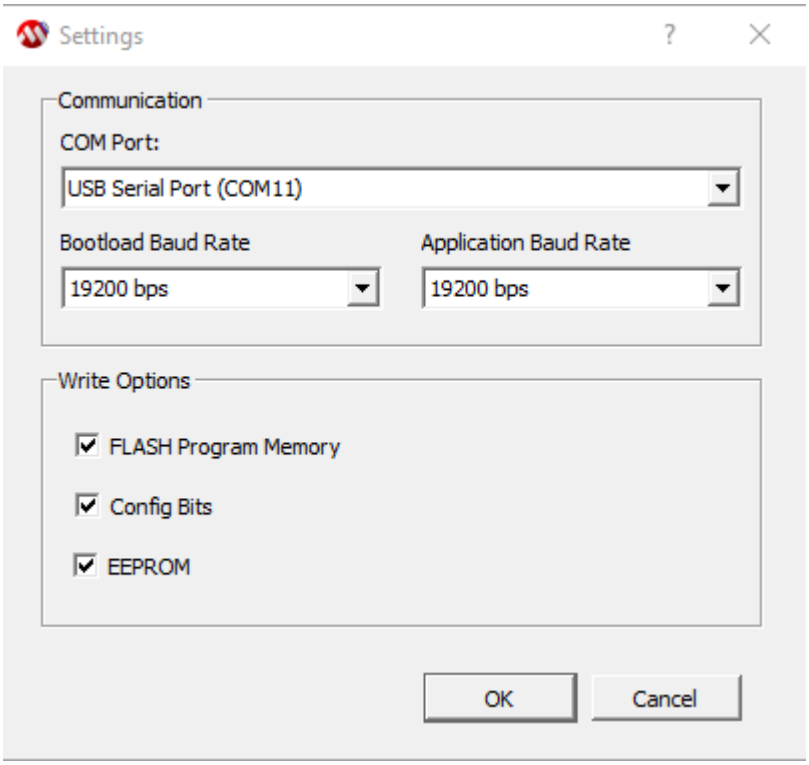


Figure 4: Boot Loader Client Settings

Once the client is configured, press the blue Pause button to send a Break to the MultiMod board so that control is transferred to the boot loader, then press the red Stop button to establish a connection. Use the File/Open or folder icon to read in the Hex file you wish to program into the processor's flash memory. All that's needed now is to press the red-arrow down Write button to download your content.

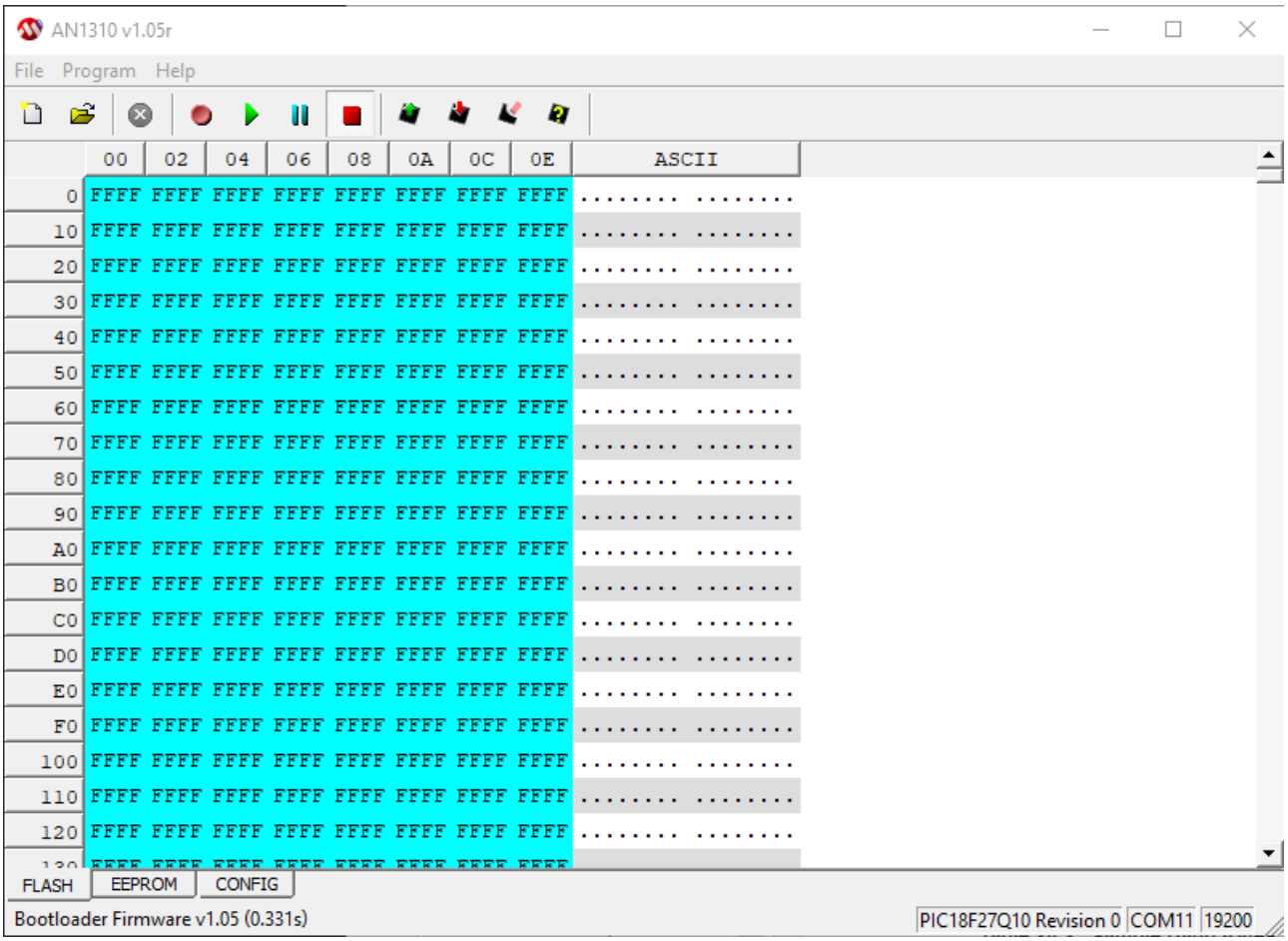


Figure 5: Client Main Page

Once the new version of the MultiMod ROM Emulator software is installed you can exit the boot loader client. You may need to reset the MultiMod PIC processor by removing and replacing the J2 power jumper or resetting the processor as detailed in the Recovery section.

Note: A platform independent Python3 version of the Microchip® boot loader client is under development. The Microchip client source code is available for those adventurous souls who care to venture into unsupported territory. Just ask.

Appendix B. Making Your Own ROM

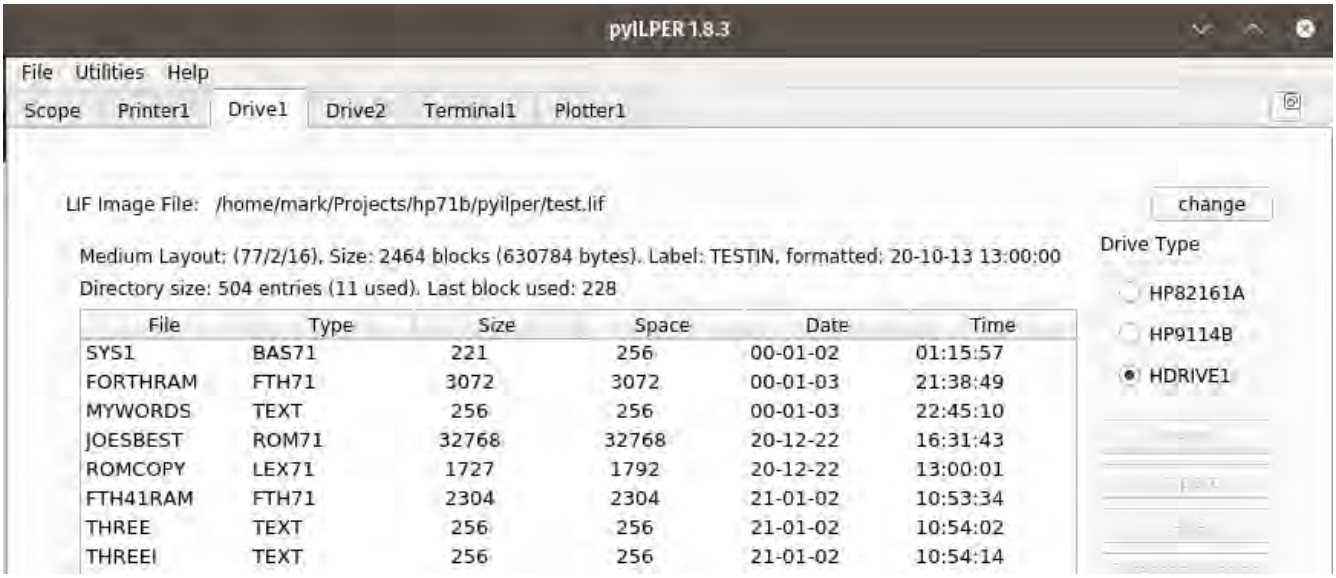
My thanks to Bob Prosperi for explaining how this is done. I've used Christoph Gießelink's Emu71[6] emulator for Windows to build my own ROM images, and you can too. You'll need to install the emulator as well as Anaconda/Miniconda for pyilper[7]. A LIF disk image containing files needed can be found in the LEX directory.

To begin, decide what size ROM you want to make and add an independent RAM of that size to your Emu71. Steps for doing so are

- Turn off the emulator
- Select Port Configuration from the Edit menu and pick an empty port (Port 2 in this example)
- Press Add. A default 32KB RAM is shown. Change the size appropriately, then press Apply
- Exit configuration and turn the emulator on
- Use the command FREE PORT(2) to create the independent RAM (IRAM)

At this point you're ready to copy the files you want on your ROM over to the IRAM you just created. Start the Anaconda shell and type pyilper. I'll assume you have a similar pyilper device setup as shown below. I'll use the following device assignment in the example

```
ASSIGN IO ":P1,:D1,:D2,:T,:PL"
```



I generally use Drive1 as my working drive and Drive2 to hold what I'm working with at the moment, so this example will assume you've loaded the working LIF image in Drive1 and the LIF image(s) you're using for your ROM in Drive2.

6 https://hp.giesselink.com/emu71.htm
7 https://github.com/bug400/pyilper/

The next step is to copy the files for your ROM to the IRAM you created. This is a three step process that can be a bit tedious if you have a lot of files, so consider writing a BASIC program to automate the task. The steps are to copy the desired file to main RAM, then copy it to IRAM, then purge the file from main RAM using the commands

```
COPY “MYROMFILE:D2”
COPY MYROMFILE TO “:PORT(2)”
PURGE MYROMFILE
```

Once copying is complete it’s a good idea to back up your IRAM in case you want to change the content later. Go back to Port 2 in the Edit/Port Configuration view. Click to select the IRAM you created then right-click and select the Save Memory Data… command. Save the IRAM content as a BIN file. Give it a name to remind you it is RAM not ROM, i.e. MYIRAM.

When you are done, load the ROMCOPY LEX file from drive 1 and use the following commands to create a ROM image

```
COPY “ROMCOPY:D1”
ROMCOPY “:PORT(2)” TO “MYROM:D1”
ROMCOPY “MYROM:D1” TO “:PORT(2)”
PURGE “MYROM:D1”
ROMCOPY “:PORT(2)” TO “MYROM:D1”
```

Now for the ROM BIN file. Go back to Port 2 in the Edit/Port Configuration view. Select the IRAM you created then right-click and select the Save Memory Data… command. Save the ROM you created as a BIN file with a name such as MYROM.BIN to use later. You can validate the BIN file by going to another port (port 3 for example) and use Add & Apply to create a ROM using your BIN image. Use the command `CAT :PORT(3)` to confirm your files are present.

With that ROM tested and ready to load, use the `bin2dat.py` program to convert the file to a format suitable to the serial monitor. Use the monitor to load the ROM into your MultiMod ROM emulator. Enjoy!