Chapter 1
Installation and Initial Setup

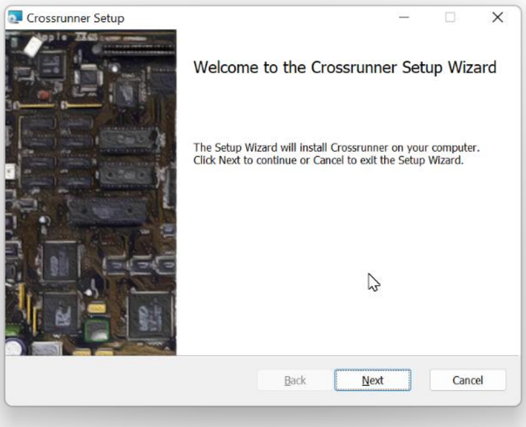
Installation and Setup on Windows

Running the Installer

Crossrunner requires Windows 7 64 bit or later. A compatible game controller, such as an Xbox controller, is recommended for software that requires a joystick. Npcap (or equivalent) is required for low-level Uthernet II emulation.

❖ Crossrunner performs much lower-level emulation than existing emulators including KEGS and Sweet 16, and as such requires a more powerful machine to run. However, it should run on any machine capable of running Windows 7. If your aim is maximum emulated machine speed, rather than emulation accuracy, then it is recommended that you choose another emulator.

Double-click on the “Crossrunner.msi” installer file to launch the installer. Click the “Next” button.

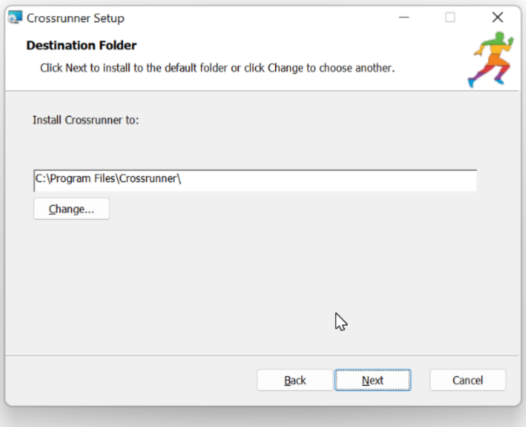


Crossrunner Setup

Welcome to the Crossrunner Setup Wizard

The Setup Wizard will install Crossrunner on your computer.
Click Next to continue or Cancel to exit the Setup Wizard.

Select the directory where you want to install Crossrunner, or keep the default selection, and then click “Next”.



Crossrunner Setup

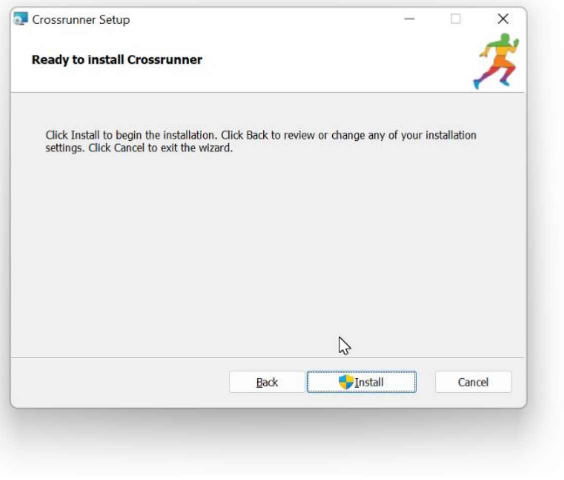
Destination Folder

Click Next to install to the default folder or click Change to choose another.

Install Crossrunner to:

C:\Program Files\Crossrunner\

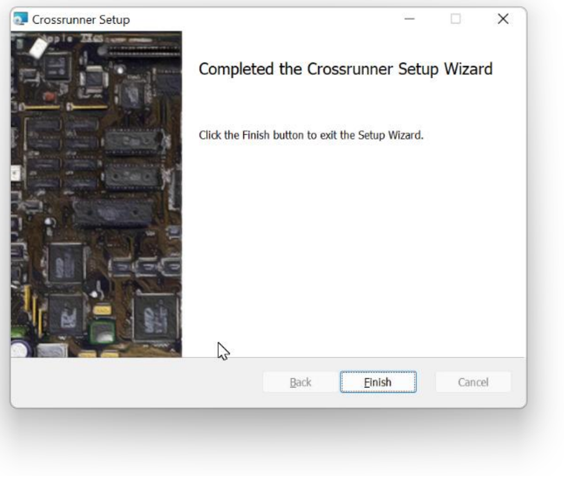
Click the “Install” button to begin the install.



Ready to install Crossrunner

Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.

Click the “Finish” button to exit the installer.



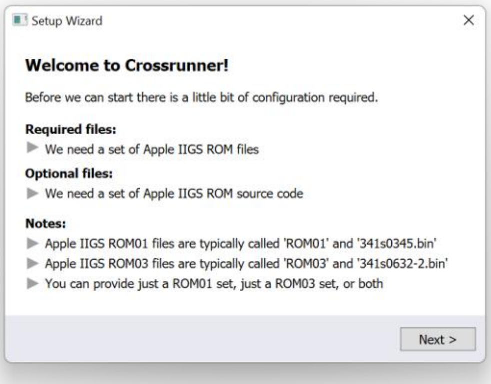
Completed the Crossrunner Setup Wizard

Click the Finish button to exit the Setup Wizard.

Crossrunner should now be installed, and you should be able to launch it from the Start button.

Starting Crossrunner for the first time

When you launch Crossrunner for the first time, it presents you with a Setup Wizard. At a minimum, you need to provide it with a set of ROM files for either a ROM01 Apple IIGS or a ROM03 Apple IIGS.



Setup Wizard

Welcome to Crossrunner!

Before we can start there is a little bit of configuration required.

Required files:

- We need a set of Apple IIGS ROM files

Optional files:

- We need a set of Apple IIGS ROM source code

Notes:

- Apple IIGS ROM01 files are typically called 'ROM01' and '341s0345.bin'
- Apple IIGS ROM03 files are typically called 'ROM03' and '341s0632-2.bin'
- You can provide just a ROM01 set, just a ROM03 set, or both

You can optionally provide it with both ROM01 and ROM03 ROM files, which will allow you to easily switch between the two whenever you want.

You can optionally provide it with the source code to the Apple IIGS ROM's. This is a feature primarily of benefit to programmers. If the source code files are provided then the built-in debugger inside Crossrunner will display the actual Apple IIGS source code for the firmware, instead of just a raw disassembly. This makes it much more pleasant to debug inside ROM code.

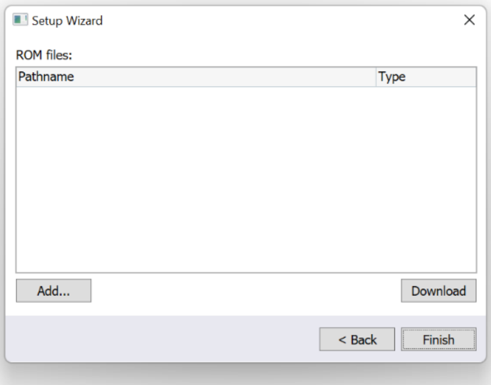
❖ ROM source code is currently only available for the ROM03 Apple IIGS. Therefore, you will only see source code when your current machine is set to ROM03.

Apple IIGS ROM01 files are typically called 'ROM01', which is a 128KB file, and '341s0345.bin', which is a 3KB file.

Apple IIGS ROM03 files are typically called 'ROM03', which is a 256KB file, and '341s0632-2.bin', which is a 4KB file.

❖ Crossrunner emulates the Apple Desktop Bus (ADB) subsystem, which is a whole "machine" in itself – with its own 6502 based processor, its own ROM, and its own RAM. This differs from other emulators including KEGS and Sweet 16 which provide a higher-level abstraction of the machine and therefore do not require ADB ROMs. With Crossrunner you need to provide an ADB ROM file (341s0xxx.bin) in addition to the main Apple IIGS ROM files.

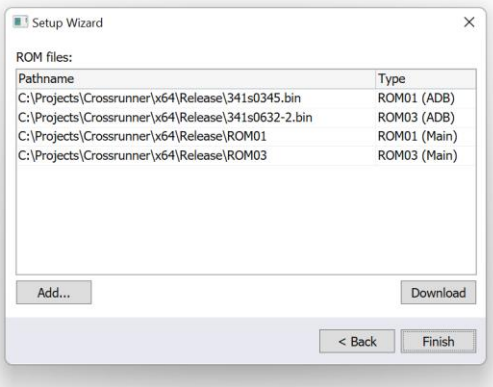
Click the "Next >" button to begin.



Click the “Add…” button and navigate to a directory where the ROM files can be located. Click the “Select Folder” button when you are in the correct directory. The list of ROM files will be populated with all matching files.

Next to each ROM file that it finds it will display the type of ROM that it detected. This will be either ROM01 (Main), ROM01 (ADB), ROM03 (Main) or ROM03 (ADB).

At a minimum you need one pair of ROM01 files or one pair of ROM03 files.



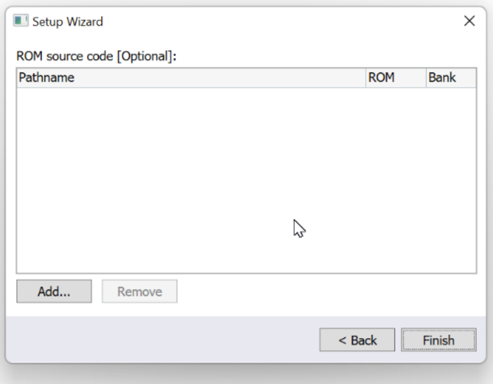
Pathname	Type
C:\Projects\Crossrunner\x64\Release\341s0345.bin	ROM01 (ADB)
C:\Projects\Crossrunner\x64\Release\341s0632-2.bin	ROM03 (ADB)
C:\Projects\Crossrunner\x64\Release\ROM01	ROM01 (Main)
C:\Projects\Crossrunner\x64\Release\ROM03	ROM03 (Main)

If your ROM files are spread over multiple directories, then you can click the “Add…” button again and pick another directory. Crossrunner will then add any additional ROM files it finds to the list.

Alternatively, you can click on the “Download” button to automatically download the ROM’s from the Internet.

❖ Please comply with the laws of the country in which you are located. Some countries may not permit you to download ROM’s even if you own an Apple IIGS with the matching ROM’s.

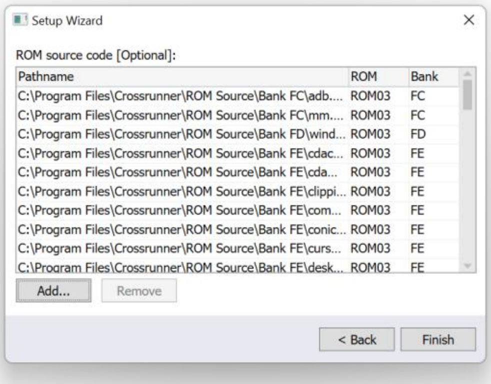
Once you have added all the ROM files click the “Next >” button.



This next stage is completely optional. If you want to specify a directory containing the Apple IIGS ROM source code then click the “Add…” button and select a directory containing the source code.

Otherwise, click “Finish” to complete the Setup Wizard.

The screenshot below shows what the list should look like if you choose to add the Apple IIGS ROM source code.

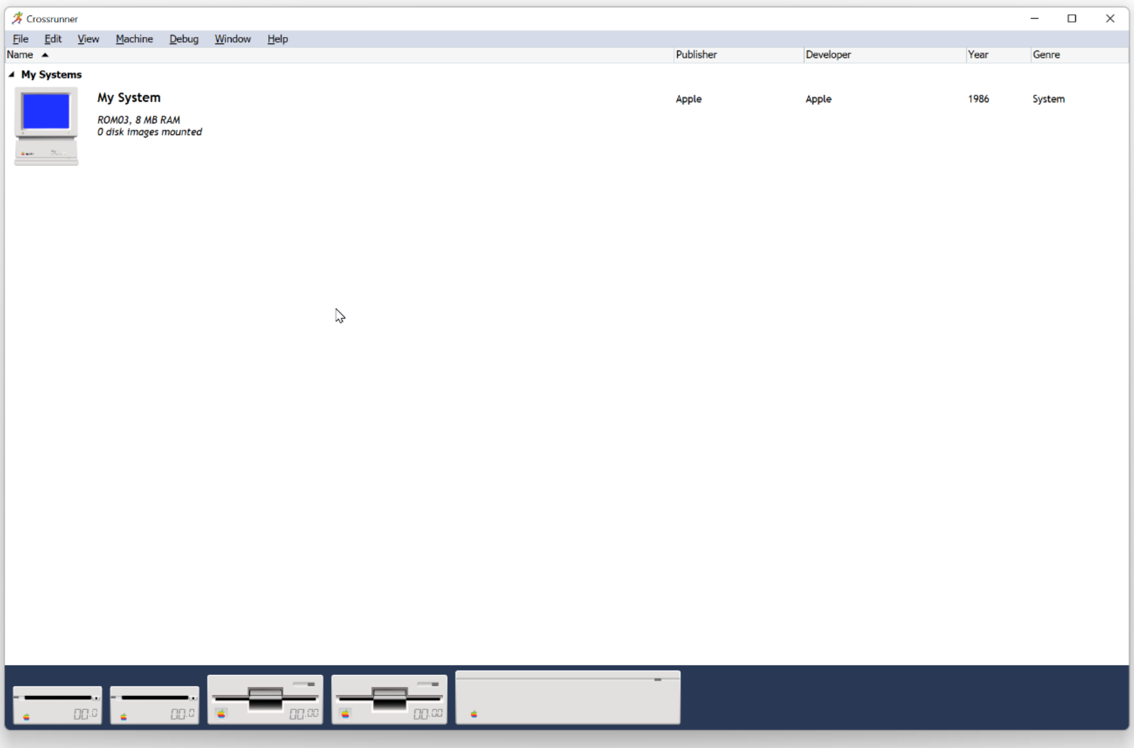

Setup Wizard

ROM source code [Optional]:

Pathname	ROM	Bank
C:\Program Files\Crossrunner\ROM Source\Bank FC\adb....	ROM03	FC
C:\Program Files\Crossrunner\ROM Source\Bank FC\mm....	ROM03	FC
C:\Program Files\Crossrunner\ROM Source\Bank FD\wind...	ROM03	FD
C:\Program Files\Crossrunner\ROM Source\Bank FE\cdac...	ROM03	FE
C:\Program Files\Crossrunner\ROM Source\Bank FE\cda...	ROM03	FE
C:\Program Files\Crossrunner\ROM Source\Bank FE\clippi...	ROM03	FE
C:\Program Files\Crossrunner\ROM Source\Bank FE\com...	ROM03	FE
C:\Program Files\Crossrunner\ROM Source\Bank FE\conic...	ROM03	FE
C:\Program Files\Crossrunner\ROM Source\Bank FE\curs...	ROM03	FE
C:\Program Files\Crossrunner\ROM Source\Bank FE\desk...	ROM03	FE

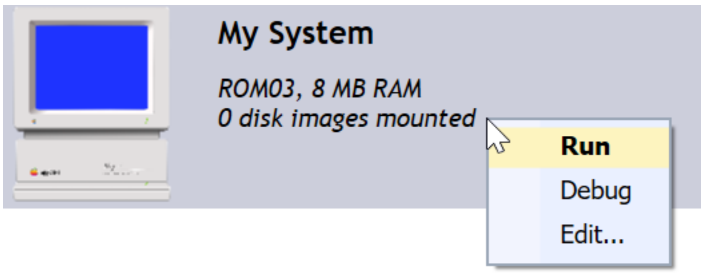
Configuring your first system

Crossrunner should now display its main window, with a single system displayed.



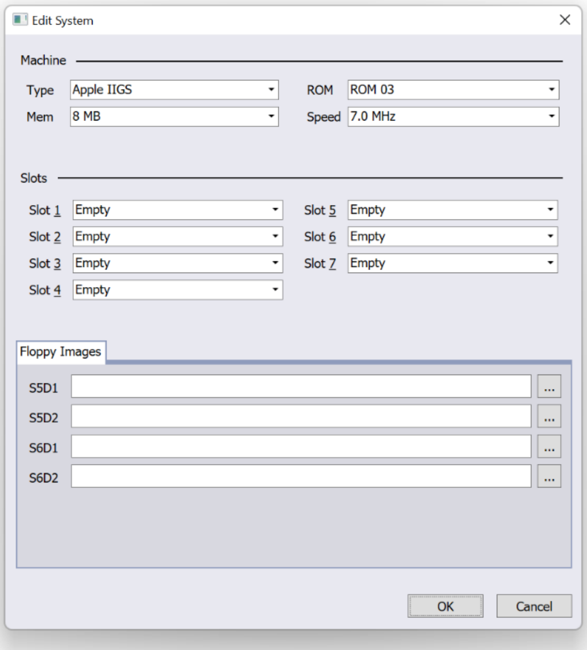
Name	Publisher	Developer	Year	Genre
My Systems				
My System ROM03, 8 MB RAM 0 disk images mounted	Apple	Apple	1986	System

You can now configure “My System” how you like it. Right-click on “My System” and select “Edit…” from the popup menu.



This will bring up the Edit System dialog. Here you can configure where it is a ROM01 or ROM03 machine, how much memory is in the machine, and whether the machine runs at standard IIGS speeds (2.8 MHz) or whether it runs at accelerated speeds (7 MHz, 14 MHz) as would be the case if you had a TransWarp GS or ZIP GSX accelerator card in a real machine.

All seven slots are listed. By default these slots will be empty, but you can insert expansion cards into slots as you desire.


Edit System

Machine

Type: Apple IIGS
ROM: ROM 03
Mem: 8 MB
Speed: 7.0 MHz

Slots

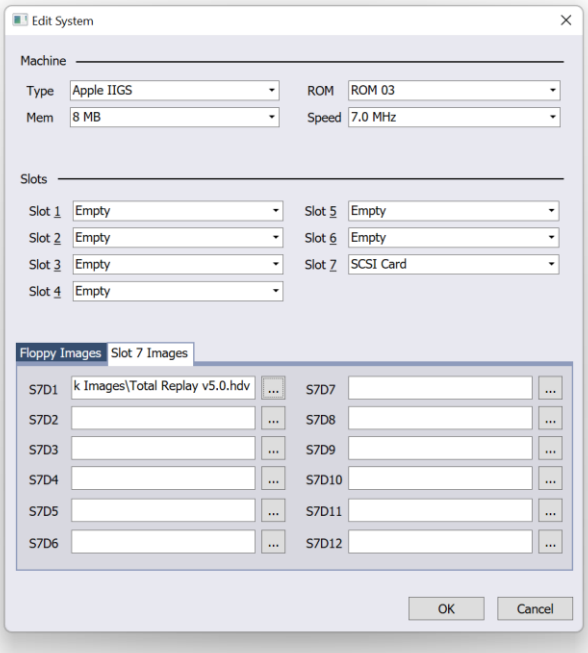
Slot 1: Empty
Slot 2: Empty
Slot 3: Empty
Slot 4: Empty
Slot 5: Empty
Slot 6: Empty
Slot 7: Empty

Floppy Images

S5D1:
S5D2:
S6D1:
S6D2:

It is recommended to start by adding a SCSI card to Slot 7. Go to the Slot 7 combo box and select “SCSI card” from the list of options.

You will notice that a new tab appears at the bottom of the dialog labelled “Slot 7 Images”.


Edit System

Machine

Type: Apple IIGS
ROM: ROM 03
Mem: 8 MB
Speed: 7.0 MHz

Slots

Slot 1: Empty
Slot 2: Empty
Slot 3: Empty
Slot 4: Empty
Slot 5: Empty
Slot 6: Empty
Slot 7: SCSI Card

Floppy Images | Slot 7 Images

S7D1: k Images\Total Replay v5.0.hdv
S7D2:
S7D3:
S7D4:
S7D5:
S7D6:
S7D7:
S7D8:
S7D9:
S7D10:
S7D11:
S7D12:

You can mount up to 12 hard drive images in each slot with a SCSI card. These are labelled S7D1 (Slot 7 drive 1) through to S7D12 (Slot 7 drive 12). Click on the "…" button next to each drive to select a disk image to mount in that spot.

In the screenshot above, we have mounted the popular "Total Replay" disk image into S7D1 as an example.

❖ Hard drive images have a file extension of either ".po", ".2mg", or ".hdv".

Click "OK" to save the configuration for "My System".

Uthernet II emulation

You can optionally add an "Uthernet II" card in the Edit System dialog. Software that uses the Uthernet II can use the card it in two different ways – a high level TCP or UDP socket interface, or a low level Raw Ethernet interface. Crossrunner supports both, but requires Npcap (www.npcap.com) to be installed if you require the low level Raw Ethernet interface. Note that both Marinetti's Uthernet II link layer and A2OSX require the low level interface.

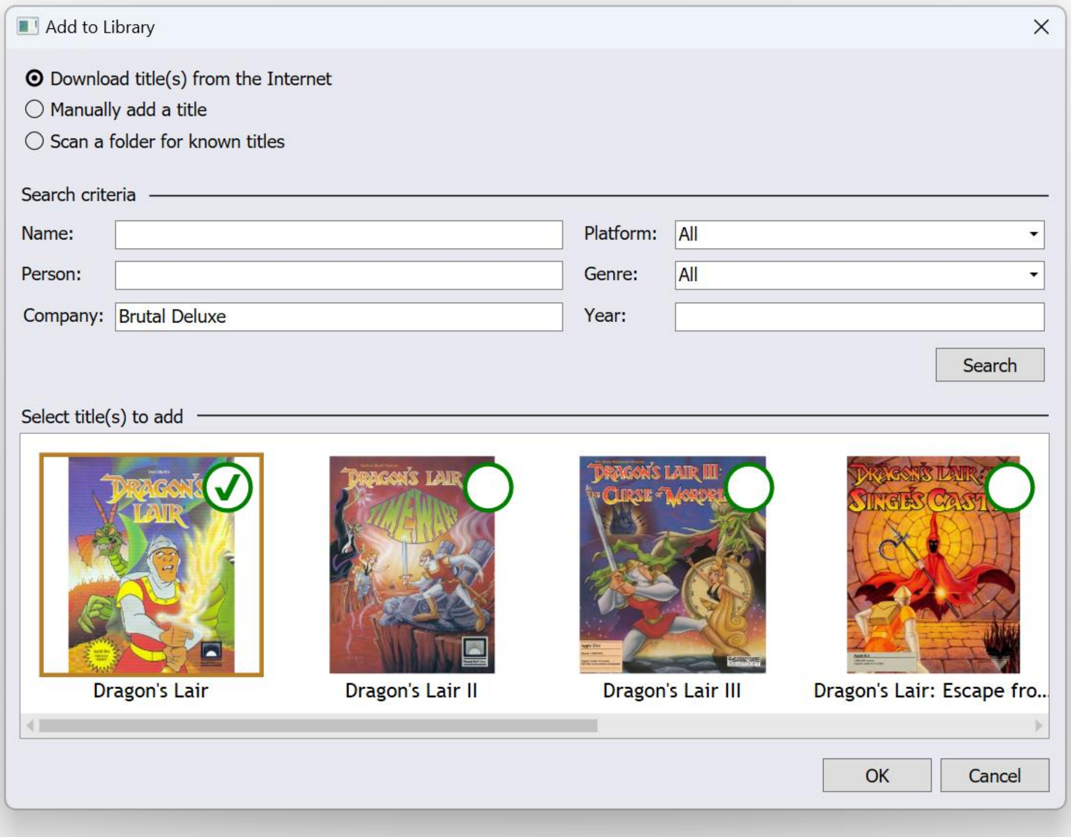
Click the "…" button next to the Uthernet II in the Edit System dialog to configure which network adapter Npcap will use for the low level Ethernet interface.

Adding disk images to your library

You can optionally add disk images to your library. There are several reasons you may choose to do this:

- It allows you to quickly launch some of your favourite games using the correct settings for the game.
- It allows you to quickly switch between disks for multi-disk games.
- It allows you to view the beautiful box art that most Apple II software titles had.

Go to the "File" menu and select "Add to Library…".


Add to Library

[✓ Download title(s) from the Internet] Manually add a title Scan a folder for known titles

Search criteria

Name:		Platform:	All
Person:		Genre:	All
Company:	Brutal Deluxe	Year:	

There are three different ways of adding titles to your library:

- You can download titles from the Internet.
- You can manually add a title to your library.
- You can scan a folder on your hard drive to find known titles.

Downloading title(s) from the Internet

The first thing you must do is to enter some search criteria for your search. Fill in one or more of the search criteria fields. You must fill in at least one field. In the example above, we have searched for all titles from “Brutal Deluxe”. Click the “Search” button to search for matches.

Matching titles will be then be displayed. Next to each title there will be a green circle. Click on the green circle to mark that title for downloading. In the example above, “Dragon’s Lair” is selected for downloading but the sequels are not. Click the “OK” button to begin downloading.

❖ Please comply with the laws of the country in which you are located. Some countries may not permit you to download software than you already own.

Manually adding a title

Select the “Manually add a title” radio button. You will have to enter the Name for the Title, and then select which Platform it was made for using the drop down. Next, you will need to click the “Add…” button and pick the disk images associated with that title.

Click the “OK” button to add the title to the library. The title will be added with a default image, and all other title attributes will be blank. You can right-click on the title in the library to edit the title and change all these values if you want to.

Scan a folder for known titles

If you already have disk images on your hard drive, then you can select the “Scan a folder for known titles” option. Click the “Scan Folder…” to pick a folder to scan. Crossrunner will now search for disk images in the folder you have selected, and also all subfolders of that folder. It will then attempt to match those disk images against its database of known disk images.

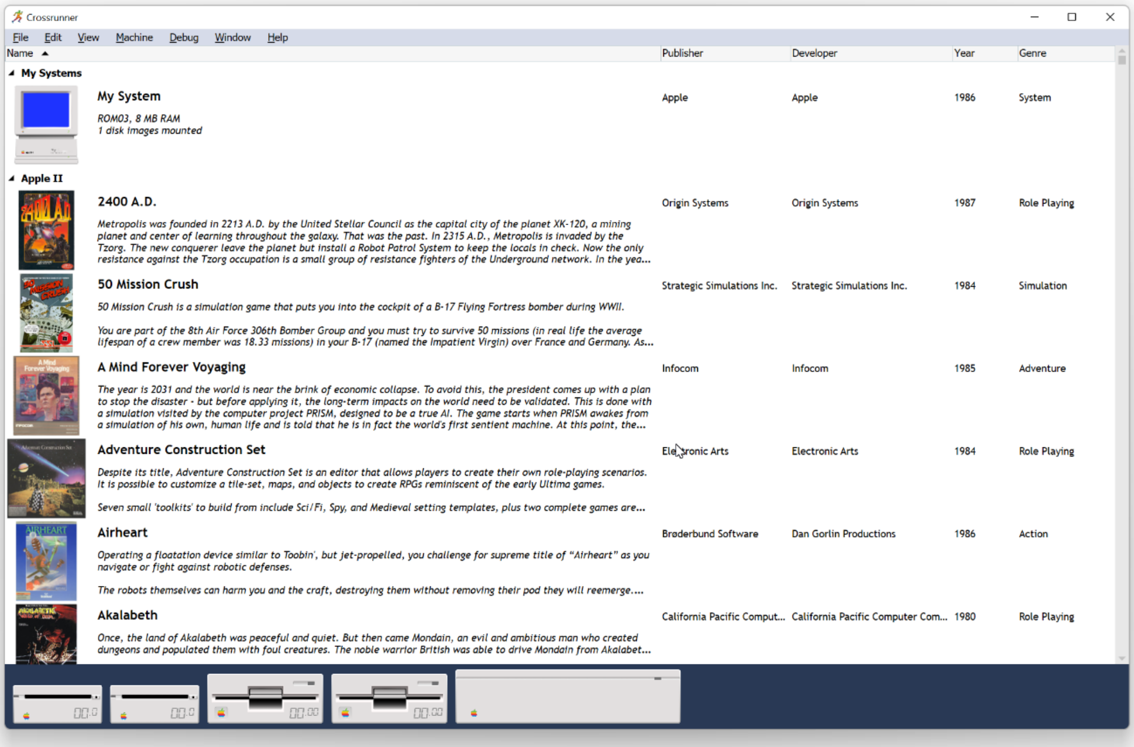
The Scan Results will show all disk images that were found in the folder, and the “Status” column will show either “Known” or “Unknown”. If the title is Unknown then Crossrunner does not know what to do with it, and these titles will not be added. You will have to manually add these titles.

Click the “OK” button to add all titles that have disk images with a “Known” status.

Regardless of which of the three different ways to add titles to the library you have selected, the main window will be updated with a list of all software titles that have been added. The titles will be grouped into one of two categories – "Apple II" for 8 bit software titles, and "Apple IIGS" for 16 bit software titles.

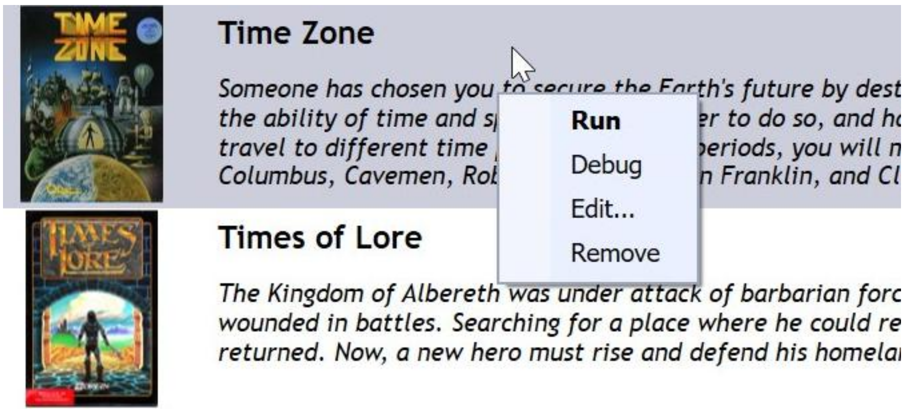
Next to each software title it displays a short description of the title, the company that developed and published the title, the year the title was released, and the genre for the title.

You can sort the list by any of these fields.



Name ▲	Publisher	Developer	Year	Genre
My Systems				
My System *ROM03, 8 MB RAM 1 disk images mounted*	Apple	Apple	1986	System
Apple II				
2400 A.D. *Metropolis was founded in 2213 A.D. by the United Stellar Council as the capital city of the planet XK-120, a mining planet and center of learning throughout the galaxy. That was the past. In 2315 A.D., Metropolis is invaded by the Tzorg. The new conquerer leave the planet but install a Robot Patrol System to keep the locals in check. Now the only resistance against the Tzorg occupation is a small group of resistance fighters of the Underground network. In the yea...*	Origin Systems	Origin Systems	1987	Role Playing
50 Mission Crush *50 Mission Crush is a simulation game that puts you into the cockpit of a B-17 Flying Fortress bomber during WWII. You are part of the 8th Air Force 306th Bomber Group and you must try to survive 50 missions (in real life the average lifespan of a crew member was 18.33 missions) in your B-17 (named the Impatient Virgin) over France and Germany. As...*	Strategic Simulations Inc.	Strategic Simulations Inc.	1984	Simulation
A Mind Forever Voyaging *The year is 2031 and the world is near the brink of economic collapse. To avoid this, the president comes up with a plan to stop the disaster - but before applying it, the long-term impacts on the world need to be validated. This is done with a simulation visited by the computer project PRISM, designed to be a true AI. The game starts when PRISM awakes from a simulation of his own, human life and is told that he is in fact the world's first sentient machine. At this point, the...*	Infocom	Infocom	1985	Adventure
Adventure Construction Set *Despite its title, Adventure Construction Set is an editor that allows players to create their own role-playing scenarios. It is possible to customize a tile-set, maps, and objects to create RPGs reminiscent of the early Ultima games. Seven small 'toolkits' to build from include Sci/Fi, Spy, and Medieval setting templates, plus two complete games are...*	Electronic Arts	Electronic Arts	1984	Role Playing
Airheart *Operating a floatation device similar to Toobin', but jet-propelled, you challenge for supreme title of "Airheart" as you navigate or fight against robotic defenses. The robots themselves can harm you and the craft, destroying them without removing their pod they will reemerge....*	Brøderbund Software	Dan Gorlin Productions	1986	Action
Akalabeth *Once, the land of Akalabeth was peaceful and quiet. But then came Mondain, an evil and ambitious man who created dungeons and populated them with foul creatures. The noble warrior British was able to drive Mondain from Akalabet...*	California Pacific Comput...	California Pacific Computer Com...	1980	Role Playing

You can right-click on any of these titles to bring up a popup menu. If you select "Edit…" from the popup menu it will allow you to make changes to the selected title.



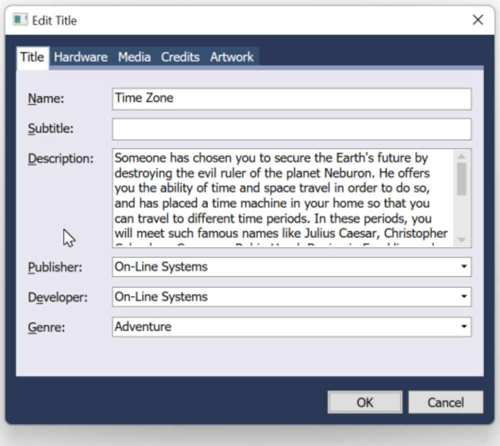
Time Zone

Someone has chosen you to secure the Earth's future by dest the ability of time and sp er to do so, and h travel to different time periods, you will n Columbus, Cavemen, Rob n Franklin, and Cl

Times of Lore

The Kingdom of Albereth was under attack of barbarian forc wounded in battles. Searching for a place where he could re returned. Now, a new hero must rise and defend his homela

The “Edit Title” dialog is shown below. On the first tab you can configure the fields that will be displayed in the list including the name of the title and a description of the title.



Name: Time Zone

Subtitle:

Description: Someone has chosen you to secure the Earth's future by destroying the evil ruler of the planet Neburon. He offers you the ability of time and space travel in order to do so, and has placed a time machine in your home so that you can travel to different time periods. In these periods, you will meet such famous names like Julius Caesar, Christopher

Publisher: On-Line Systems

Developer: On-Line Systems

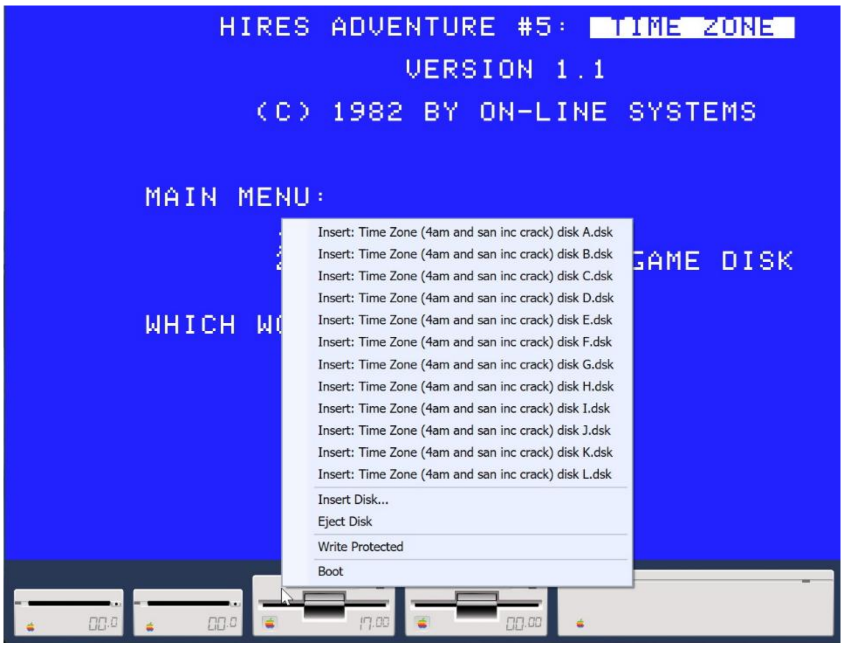
Genre: Adventure

In the “Hardware” tab you can configure the hardware configuration that will be used when running that title. For example, some titles may run better on an accelerated machine while others may run better at 1 MHz or 2.8 MHz. Some 8-bit software titles may take advantage of a Mockingboard card, in which case you can configure the machine to always include a Mockingboard card when running that title.

Click “OK” when you have finished making changes.

While you can just insert disks into floppy drives in the emulator on an ad-hoc basis instead of adding software titles to a library, you will miss out on some of the features.

In the screenshot below, we are running Time Zone from our software library. Time Zone is a 12 disk game with lots of disk swapping required! When you right-click on the floppy drive while running it, it gives you a “quick list” of all the disk images that make up that title – allowing you to swap disks quickly!


HIRES ADVENTURE #5: TIME ZONE
VERSION 1.1
(C) 1982 BY ON-LINE SYSTEMS
MAIN MENU:
GAME DISK
WHICH W
Insert: Time Zone (4am and san inc crack) disk A.dsk
Insert: Time Zone (4am and san inc crack) disk B.dsk
Insert: Time Zone (4am and san inc crack) disk C.dsk
Insert: Time Zone (4am and san inc crack) disk D.dsk
Insert: Time Zone (4am and san inc crack) disk E.dsk
Insert: Time Zone (4am and san inc crack) disk F.dsk
Insert: Time Zone (4am and san inc crack) disk G.dsk
Insert: Time Zone (4am and san inc crack) disk H.dsk
Insert: Time Zone (4am and san inc crack) disk I.dsk
Insert: Time Zone (4am and san inc crack) disk J.dsk
Insert: Time Zone (4am and san inc crack) disk K.dsk
Insert: Time Zone (4am and san inc crack) disk L.dsk
Insert Disk...
Eject Disk
Write Protected
Boot

Chapter 2
Menu Options

File menu

Add to Library… – Adds titles to your software library. See Chapter 1 for details on how to add titles to your library.

Close – This menu item will be enabled when the emulator is running. This will close the current emulation session and return you to the main screen.

Load State… – Prompts you for a Save State file and then restores the emulator state from that file.

Save State… – Prompts you for a file to save the current emulator state into. You can then use the Load State menu item to restore the emulator back to this state at any time you want.

Quick Load State – Restores the emulator state from a global snapshot file without any prompting.

Quick Save State – Saves the emulator state to a global snapshot file without any prompting.

Edit menu

Go To – When editing a file on a disk image or in the script editor it will prompt you for the line number to go to. When using the Debugger, this will prompt you for the memory address to go to. This functionality is described in more detail in Chapter 3.

Find and Replace > Quick find – When editing a file on a disk image or in the script editor this will bring up the Quick Find overlay where you can type in the text that you wish to search for. When using the Debugger, this will allow you to search for source code associated with the current bank of memory. This functionality is described in more detail in Chapter 3.

Find and Replace > Quick replace – When editing a file on a disk image or in the script editor this will bring up the Quick Replace overlay where you can type in the text that you wish to search for, the text you want to replace it with and then you can either replace each occurrence on a case-by-case basis, or choose to replace all matches.

Undo, Redo, Copy, Cut, Paste, Delete, Select All – these are the standard clipboard actions.

Cancel Paste – This is enabled when you attempt to paste a large amount of data from the clipboard into the emulated machine, as pasting can take a long time depending on what you are running and how much data you are pasting. This will cancel the paste operation currently in process.

Make Uppercase – This will convert the currently selected text to uppercase. This is enabled when editing files on disk images or when using the script editor.

Make Lowercase – This will convert the currently selected text to lowercase. This is enabled when editing files on disk images or when using the script editor.

View White Space – When enabled white space (spaces, tabs) will be displayed using symbols making it easier to count the number of spaces or to identify where tabs are used instead of spaces and vice versa. This is enabled when editing files on disk images or when using the script editor.

Increase Line Indent – This will increase the indentation for the currently selected text. This is enabled when editing files on disk images or when using the script editor.

Decrease Line Indent – This will decrease the indentation for the currently selected text. This is enabled when editing files on disk images or when using the script editor.

Comment Selection – This will comment out the currently selected text. This will use the commenting sequence associated with the current file type – for example “//” for C and Wren, “(* *)” for Pascal, and “*” for assembler. This is enabled when editing files on disk images or when using the script editor.

Uncomment Selection – This will uncomment the currently selected text. This is enabled when editing files on disk images or when using the script editor.

View menu

Solution Explorer – This functionality is disabled in the current version of Crossrunner.

Output – This functionality is disabled in the current version of Crossrunner.

Shell – This will bring up the Shell pane, which is described in Chapter 4.

Scripting – This will bring up the Script editor, which is described in Chapter 6.

Script Log – This will bring up the Script Log, which is described in Chapter 6.

Monitor > Integral Scaling – This will ensure that the monitor is only scaled using an integral scaling factor – for example – 1x, 2x, 3x, 4x and so on. This will ensure the sharpest display quality but it will mean that the monitor may not take up all available space on the screen.

Monitor > Stretch to fit – This will expand the monitor to fit the available screen real estate.

Monitor > No forced Aspect Ratio – When combined with Stretch to fit it will allow the monitor to use all available screen real estate regardless of aspect ratio. This may result in a display that is either too wide or too skinny. If Integral Scaling is enabled, that will take precedence over this setting.

Monitor > 1:1 Aspect Ratio – When combined with Stretch to fit this will make the monitor use the same aspect ratio as the host system. If Integral Scaling is enabled, that will take precedence over this setting.

Monitor > 1:1.25 Aspect Ratio – When combined with Stretch to fit this will make the monitor use an aspect ratio that is slightly taller than the aspect ratio of modern displays. This will closely match the aspect ratio of the Apple II. If Integral Scaling is enabled, that will take precedence over this setting.

Monitor > Show borders – When enabled this will display the border area of the Apple IIGS monitor. Some Apple IIGS software (mainly demos) used the border area for graphics. Disabling the border area will free up this space.

Monitor > Landscape orientation – The default monitor orientation.

Monitor > Portrait orientation – This will rotate the monitor 90 degrees into Portrait mode. There are a very small number of Apple II games that require you to run the monitor this way – for example, early versions of Alien Rain and Snoggle. These games ran in portrait mode because they were based on Arcade games that ran with the CRT rotated into Portrait mode. However, later versions were re-released in Landscape orientation – presumably after Broderbund received complaints from users.

Monitor > Green monochrome LCD monitor – This is not a real world monitor, but this simulates a monochrome (Green) monitor connected through the Apple IIGS RGB video port, through a RGB to HDMI conversion box.

Monitor > Amber monochrome LCD monitor – This is not a real world monitor, but this simulates a monochrome (Amber) monitor connected through the Apple IIGS RGB video port, through a RGB to HDMI conversion box.

Monitor > Colour LCD monitor – This is a LCD monitor connected through the Apple IIGS RGB video port, through a RGB to HDMI conversion box. This option will generate the sharpest image, while preserving the characteristics of dithered 640 colours which appear as a proper colour instead of lines of alternating colours.

Monitor > Green monochrome CRT monitor – This is a monochrome (Green) monitor connected through the Apple IIGS composite video port.

Monitor > Amber monochrome CRT monitor – This is a monochrome (Amber) monitor connected through the Apple IIGS composite video port.

Monitor > AppleColor Composite CRT monitor – This is an AppleColor Composite CRT monitor connected to an 8-bit Apple II composite video port. This monitor was typically used with Apple IIe systems.

There are a small number of 8 bit Apple II games that look better in composite (with NTSC artefacts) than in RGB. NTSC quirks mean that there will be all kinds of colour artefacts being displayed – such as yellows and pinks in High Res mode. Note that the Apple IIGS hardware is, unlike earlier machines, an RGB native system and it therefore generates a composite image in a different manner to earlier Apple II systems. This composite image quality is widely regarded as inferior to the 8 bit Apple II machines. Crossrunner, therefore does not attempt to emulate the Apple IIGS version of Composite but rather the Apple IIe version of Composite. This can be considered similar to putting a VidHD card into the Apple IIGS, which makes the IIGS output 8 bit graphics in a manner closer to the original 8 bit machines.

Monitor > AppleColor RGB CRT monitor – This will change the monitor into an AppleColor RGB monitor connected through the Apple IIGS RGB video port. This will emulate the characteristics of that monitor including its shadow mask, geometry and sharpness. This was the standard monitor sold with an Apple IIGS computer system.

Monitor > Custom RGB CRT monitor – This will change the monitor into a custom RGB monitor connected through the Apple IIGS RGB video port. Currently this is configured to emulate a Sony PVM monitor including its aperture grill, geometry and sharpness. In the future, this will be user configurable for users that want to tweak CRT settings.

Monitor > Fullscreen mode – This will toggle Crossrunner into Fullscreen mode. The menu bar and the drive bar will disappear, and the full screen will be used for the monitor. Use Alt+Enter to leave fullscreen mode.

Debugging > Registers – This will show the Registers pane. This is described in more detail in Chapter 3.

Debugging > Watch – This will show the Watch pane. This is described in more detail in Chapter 3.

Debugging > Ensoniq – This will show the Ensoniq pane. This is described in more detail in Chapter 3.

Debugging > Breakpoints – This will show the Breakpoints pane. This is described in more detail in Chapter 3.

Debugging > Memory 1 – This will show the first instance of the Memory pane. This is described in more detail in Chapter 3.

Debugging > Memory 2 – This will show a second instance of the Memory pane. This is described in more detail in Chapter 3.

Debugging > Memory Map – This will show the Memory Map pane. This is described in more detail in Chapter 3.

Debugging > Monitor 1 – This will show the first instance of the Monitor pane. This is described in more detail in Chapter 3.

Debugging > Monitor 2 – This will show a second instance of the Memory pane. This is described in more detail in Chapter 3.

View As > Tiles – This will change the Library view to a Tile based layout. This maximises the size of box art and removes all text other than the game title.

View As > Details – This will change the Library view to a List based layout, with a description of each title and details about the publisher, developer, year and genre.

Group By > Platform, Publisher, Developer, Year, Genre – This will group the Library view by the selected category. For example, grouping by Platform will group all the Apple II games into a group and all the Apple IIGS games into another group. Whereas grouping by Genre will group all the Action games into a group regardless of whether they are 8 bit or 16 bit titles.

Sort By > Name, Publisher, Developer, Year, Genre – This will sort the Library view using the selected criteria.

Machine menu

1 MHz, 2.8 MHz, 7 MHz, 14 MHz, Unlimited Speed – This will set the speed of the emulated machine. 1 MHz will match the speed of the 8 bit Apple II's. 2.8 MHz will match the speed of a stock Apple IIGS. 7 MHz will match the speed of an Apple IIGS with a Transwarp GS accelerator card. 14 MHz will match the speed of an Apple IIGS with the latest Applesqueezer accelerator card. Unlimited speed will allow the emulated machine to run as fast as possible.

Reset – This is equivalent to pressing Ctrl+Reset on the keyboard.

Reboot – This is equivalent to pressing Ctrl+Open Apple+Reset on the keyboard. This will reboot the machine.

Control Panel – This is equivalent to pressing Ctrl+Open Apple+Escape on the keyboard. It will bring up the Apple IIGS Control Panel.

Pause – This will pause the emulated machine. Select this option again to unpause the machine.

Mute Audio – This will mute audio generated by the emulated machine. Select this option again to unmute the audio.

Capture Mouse – In Apple IIGS desktop applications Crossrunner will try to automatically synchronise the host machine's mouse position with the emulated machine's mouse position for a seamless experience. However, it may be preferable in some titles for the emulated machine to take full control over mouse input. This is called "capturing the mouse". When the mouse is captured you will be unable to use the mouse in Windows, as all mouse input will be sent to the emulated machine. This is perfect for games like Crystal Quest. To remove the mouse capture, hold down the Control key while clicking the right mouse button.

Take Screenshot – This will take a screenshot of the emulated machine's screen and save it to a PNG file on disk. The file will always be saved at native, unscaled, resolution. Screenshots will be saved in the default "Pictures" folder that has been configured in Windows inside a folder called "Screenshots".

Debug menu

Build – This functionality is disabled in the current version of Crossrunner.

Clean – This functionality is disabled in the current version of Crossrunner.

Start Debugging – This will change the emulated machine/program to a running state inside the Debugger. It will keep running until it either hits a breakpoint, or you manually Break execution.

Start Without Debugging – This will change the emulated machine/program to a running state, but not inside the Debugger. This will maximise screen real estate for the monitor.

Break – This will halt the emulated machine if it is running, and change Crossrunner to Debugging mode.

Stop Debugging – This will close the current emulation session and return you to the main screen.

Step Into – When the emulated machine is halted in the Debugger, this option will execute a single machine language instruction and then halt. If the instruction is a Jump to Subroutine (JSR) then the machine will enter that subroutine and halt on the first instruction within that subroutine.

Step Over – When the emulated machine is halted in the Debugger, this option will execute a single machine language instruction and then halt. However, if the instruction is a Jump to Subroutine (JSR) then the Debugger will execute that entire subroutine as though it were a single instruction. The machine will then halt on the instruction after the JSR.

Toggle Breakpoint – This will toggle a breakpoint at the selected address within the Debugger.

Reset Profiling Data – Resets all internal profiling information, including the cycle counts associated with each memory address. You will typically reset the profiling data when you have hit a breakpoint where you wish to begin profiling. This option will then clear all data prior to that point.

Help menu

About… – This will display the About Crossrunner screen, with the Crossrunner mascot!

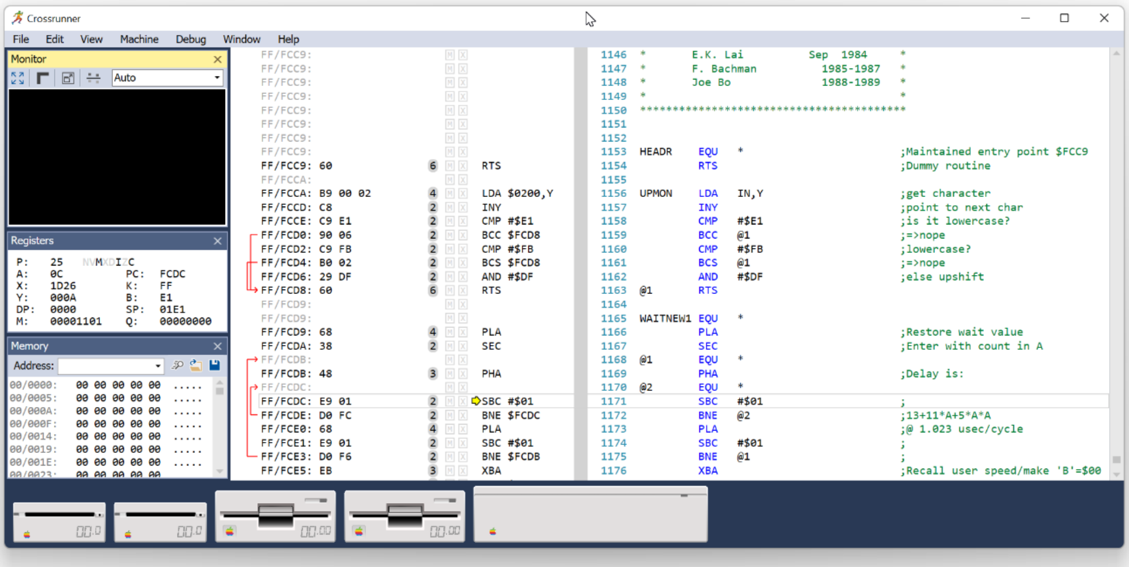
Chapter 3
Debugging

Activating the Debugger

There are several ways to activate the Debugger:

1. While something is running you can go to the "Debug" menu and select "Break". The debugger will be set to the line that was currently executing.
2. You can right-click on a System or a Title and select "Debug" from the popup menu. The debugger will be set to 00/FA62, which is the Apple II's reset vector.
3. If a previously configured breakpoint is hit then the debugger will appear and will be set to the line where the breakpoint was triggered.

The Debugger is shown below.



```
P:    25   NVMXDIZC
A:    0C          PC:   FCDC
X:    1D26        K:    FF
Y:    000A        B:    E1
DP:   0000        SP:   01E1
M:    00001101    Q:    00000000
```

```
00/0000:   00 00 00 00 00   .....
00/0005:   00 00 00 00 00   .....
00/000A:   00 00 00 00 00   .....
00/000F:   00 00 00 00 00   .....
00/0014:   00 00 00 00 00   .....
00/0019:   00 00 00 00 00   .....
00/001E:   00 00 00 00 00   .....
00/0023:   00 00 00 00 00
```

```
FF/FCC9:                     M X
FF/FCC9:                     M X
FF/FCC9:                     M X
FF/FCC9:                     M X
FF/FCC9:                     M X
FF/FCC9:                     M X
FF/FCC9:                     M X
FF/FCC9:                     M X
FF/FCC9: 60               6  M X  RTS
FF/FCCA:                     M X
FF/FCCA: B9 00 02         4  M X  LDA $0200,Y
FF/FCCD: C8               2  M X  INY
FF/FCCE: C9 E1            2  M X  CMP #$E1
FF/FCD0: 90 06            2  M X  BCC $FCD8
FF/FCD2: C9 FB            2  M X  CMP #$FB
FF/FCD4: B0 02            2  M X  BCS $FCD8
FF/FCD6: 29 DF            2  M X  AND #$DF
FF/FCD8: 60               6  M X  RTS
FF/FCD9:                     M X
FF/FCD9:                     M X
FF/FCD9: 68               4  M X  PLA
FF/FCDA: 38               2  M X  SEC
FF/FCDB:                     M X
FF/FCDB: 48               3  M X  PHA
FF/FCDC:                     M X
FF/FCDC: E9 01            2  M X  SBC #$01
FF/FCDE: D0 FC            2  M X  BNE $FCDC
FF/FCE0: 68               4  M X  PLA
FF/FCE1: E9 01            2  M X  SBC #$01
FF/FCE3: D0 F6            2  M X  BNE $FCDB
FF/FCE5: EB               3  M X  XBA
```

```
*       E.K. Lai          Sep  1984     *
*       F. Bachman          1985-1987   *
*       Joe Bo              1988-1989   *
*                                       *
*****************************************


HEADR    EQU   *                          ;Maintained entry point $FCC9
         RTS                              ;Dummy routine

UPMON    LDA   IN,Y                       ;get character
         INY                              ;point to next char
         CMP   #$E1                       ;is it lowercase?
         BCC   @1                         ;=>nope
         CMP   #$FB                       ;lowercase?
         BCS   @1                         ;=>nope
         AND   #$DF                       ;else upshift
@1       RTS

WAITNEW1 EQU   *
         PLA                              ;Restore wait value
         SEC                              ;Enter with count in A
@1       EQU   *
         PHA                              ;Delay is:
@2       EQU   *
         SBC   #$01                       ;
         BNE   @2                         ;13+11*A+5*A*A
         PLA                              ;@ 1.023 usec/cycle
         SBC   #$01                       ;
         BNE   @1                         ;
         XBA                              ;Recall user speed/make 'B'=$00
```

In the example above on the left-hand side of the screen, we have docked a number of panes – a Monitor pane, a Register pane, and a Memory pane. These are all optional panes and can be removed or moved around the screen as you wish. These panes are described later in this chapter.

On the right-hand side of the screen is the source code that has been matched to the current location. In this example below, we are within the ROM area so the source code is the Apple IIGS ROM 03 source code. If you were debugging a RAM area which contained your code then your source code would be displayed here. If there is no source code available for the memory location you are viewing then this area will be blank.

In the middle, there is the disassembly view. This divided into columns.

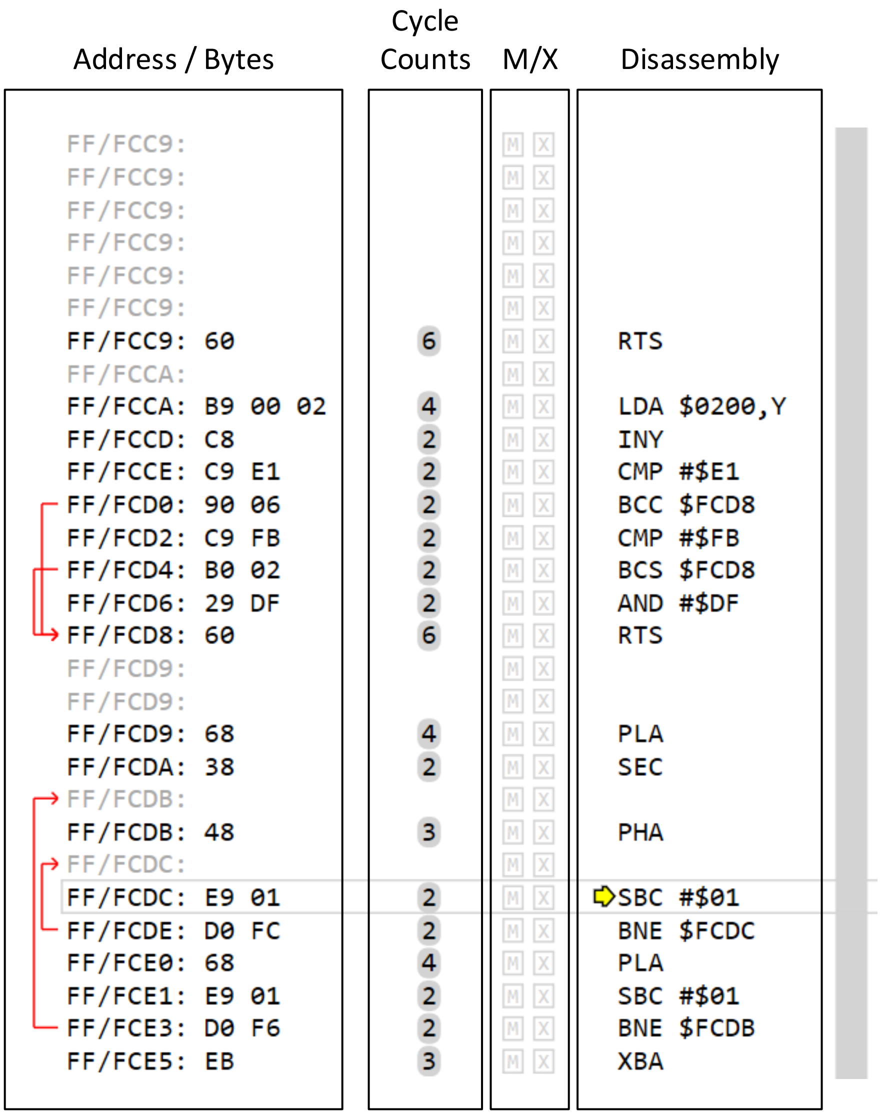
1. **Address / Bytes** – this first column contains the address in memory where the code/data is located and the bytes at that location.
   There is a branch indicator, if the line contains a branch instruction, which shows where the branch destination is. If you move the cursor to a line with a branch instruction then that branch path will be highlighted.
2. **Cycle Counts –** the next column contains the number of cycles that the instruction will take to execute.
3. **M/X indicators –** the next column contains the M and X processor flags that were used when disassembling that address.

An uppercase ‘M’ indicates 16-bit accumulator/memory. A lowercase ‘m’ indicates 8-bit accumulator/memory. Similarly, an uppercase ‘X’ indicates 16-bit index registers. A lowercase ‘x’ indicates 8-bit index registers. Areas that have not been disassembled, or are identified as data, will have a ‘?’ instead.

If the disassembly uses the wrong M/X flags then you can change them by clicking on the ‘M’ or the ‘X’ – this will toggle the flag between 8-bit and 16-bit. This will also affect all lines that follow up until a branch or a return.

For areas where the disassembler has source code, and is therefore confident about the M/X flags, then these will be greyed out and will not be able to be toggled.

4. **Disassembly –** the next column contains the disassembly. If the Program Counter (PC) is at that address then a yellow area will indicate the current execution point. This is shown in the example below.



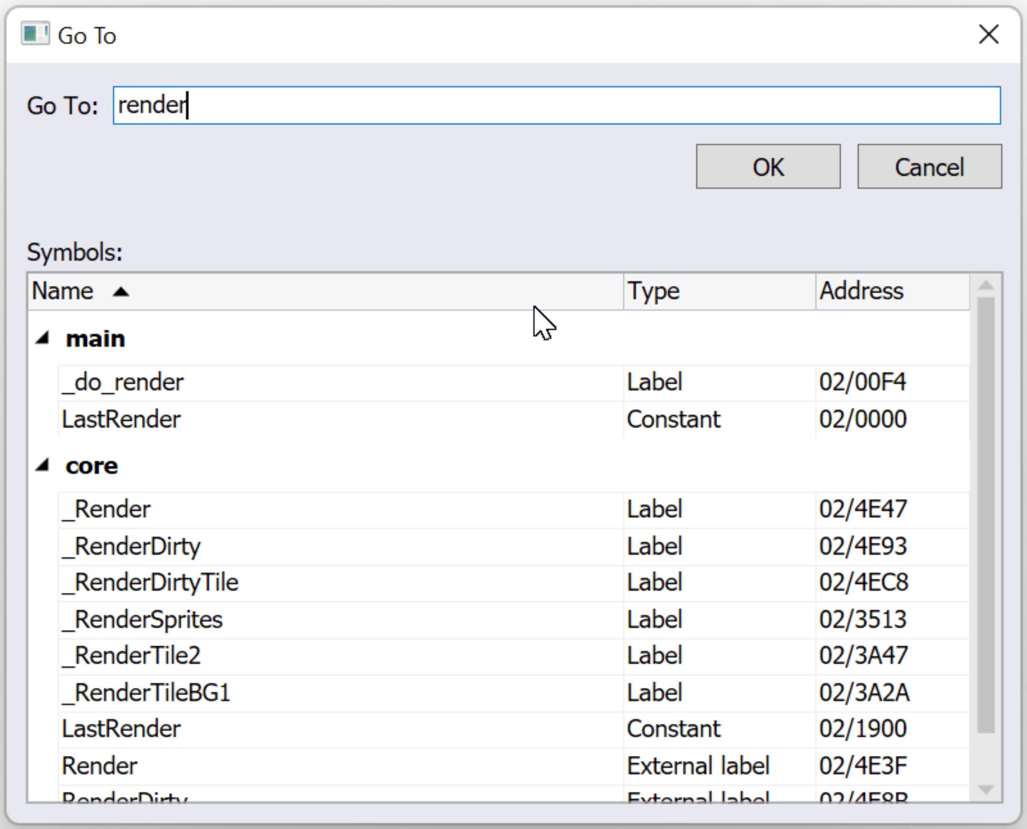
```
FF/FCC9:                M X
FF/FCC9:                M X
FF/FCC9:                M X
FF/FCC9:                M X
FF/FCC9:                M X
FF/FCC9:                M X
FF/FCC9: 60         6   M X   RTS
FF/FCCA:                M X
FF/FCCA: B9 00 02   4   M X   LDA $0200,Y
FF/FCCD: C8         2   M X   INY
FF/FCCE: C9 E1      2   M X   CMP #$E1
FF/FCD0: 90 06      2   M X   BCC $FCD8
FF/FCD2: C9 FB      2   M X   CMP #$FB
FF/FCD4: B0 02      2   M X   BCS $FCD8
FF/FCD6: 29 DF      2   M X   AND #$DF
FF/FCD8: 60         6   M X   RTS
FF/FCD9:                M X
FF/FCD9:                M X
FF/FCD9: 68         4   M X   PLA
FF/FCDA: 38         2   M X   SEC
FF/FCDB:                M X
FF/FCDB: 48         3   M X   PHA
FF/FCDC:                M X
FF/FCDC: E9 01      2   M X  ⇨SBC #$01
FF/FCDE: D0 FC      2   M X   BNE $FCDC
FF/FCE0: 68         4   M X   PLA
FF/FCE1: E9 01      2   M X   SBC #$01
FF/FCE3: D0 F6      2   M X   BNE $FCDB
FF/FCE5: EB         3   M X   XBA
```

❖ You select one or more lines that have not been disassembled, and press the “C” key (for “Code”) to disassemble the selection.

❖ You select one or more lines that have been disassembled as code, and press the “U” key (for “Undefine”) to remove the disassembly.

Go To Address / Go To Symbol

If you want to quickly jump to a different location in memory within the Debugger, then you can either press Ctrl+G, or you can go to the “Edit” menu > “Go To…”.



Name	Type	Address
main		
_do_render	Label	02/00F4
LastRender	Constant	02/0000
core		
_Render	Label	02/4E47
_RenderDirty	Label	02/4E93
_RenderDirtyTile	Label	02/4EC8
_RenderSprites	Label	02/3513
_RenderTile2	Label	02/3A47
_RenderTileBG1	Label	02/3A2A
LastRender	Constant	02/1900
Render	External label	02/4E3F
RenderDirty	External label	02/4E8B

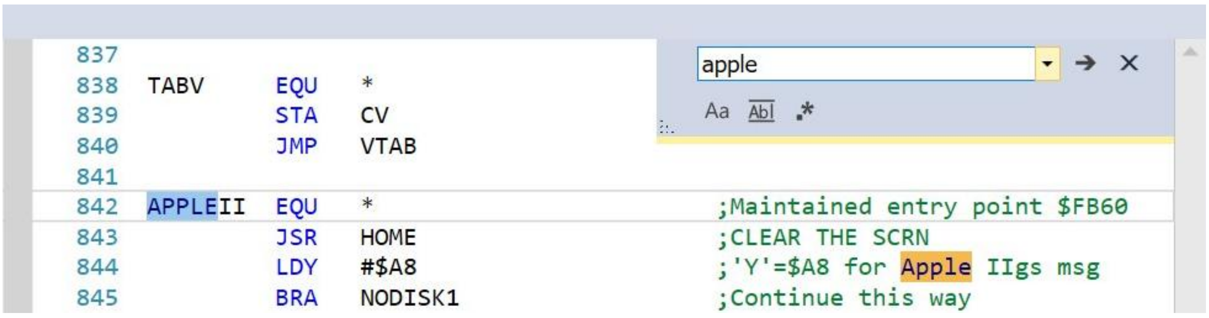
At the top of the dialog is a text field where you can enter the address that you wish to jump to.

If are currently debugging an ORCA or Merlin application for which you have provided source code (see Chapter 5) then there will also be a list of symbols shown in the dialog. You can either double-click on a symbol in the list, to go to the address where the symbol is located, or you can type the symbol’s name in the text field.

If you are not debugging an ORCA or Merlin application, but your emulated machine has booted into GS/OS, then a list of GS/OS system memory allocations will be shown instead. This allows you to quickly locate, in memory, where a given application resides.

Find

If you want to find some text within the source code that you are currently debugging then you can either press Ctrl+F to bring up the Find overlay, or you can go to the “Edit” menu > “Find and replace” > “Quick Find”.



```

TABV     EQU   *
         STA   CV
         JMP   VTAB

APPLEII  EQU   *                ;Maintained entry point $FB60
         JSR   HOME             ;CLEAR THE SCRN
         LDY   #$A8             ;'Y'=$A8 for Apple IIgs msg
         BRA   NODISK1          ;Continue this way
```

Simply type the text that you wish to search for and then hit Enter. If there are multiple matches then you can just keep hitting Enter. An alternate to using the Enter key is to click on the → button within the Find overlay.

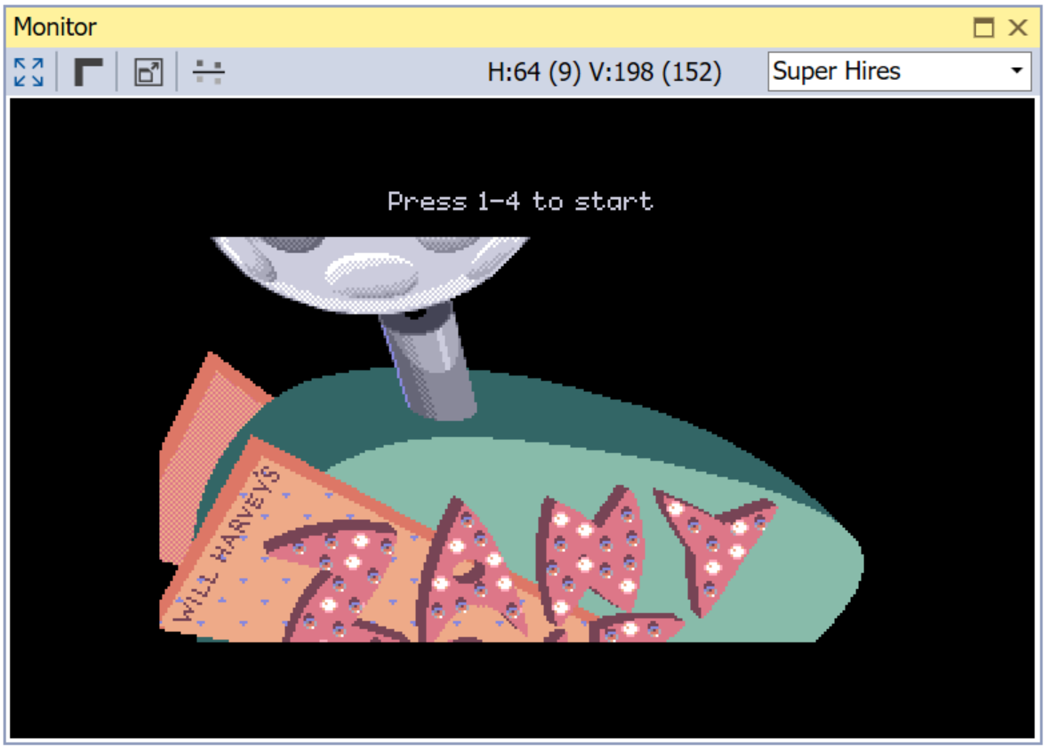
At the bottom of the find overlay are three buttons that can be used to customise your search. From left to right they are:

1. **Match case** – If selected this will perform a case sensitive search. Otherwise it the search will be case insensitive.
2. **Whole words only** – If selected this will only exclude partial matching. Otherwise partial matches will be returned. An example of a partial match would be to find “secretary” when searching for “secret”.
3. **Regular expressions** – If selected then the search string will be treated as a standard regular expression. Otherwise the search string will be treated as a literal string.

Debugging screen output

The Monitor pane displays screen output. By default the Monitor pane is displayed when the debugger is activated. If it is not display, then you can show the main monitor pane through the “View” menu > “Debugging” > “Monitor 1”.

There is a second monitor pane available. This is not displayed by default, but can be shown by selecting “View” > “Debugging” > “Monitor 2”. The second monitor pane is useful, for example, when debugging software that does page flipping – you could use Monitor Pane 1 to display Hires Page 1 and Monitor Pane 2 to display Hires Page 2.



In the top-right of the Monitor pane there is a drop down, which allows you to specify the screen that is displayed. The default option, "Auto", displays the screen as seen by the user. The other options let you force specific screens to be displayed, for example "Super Hires" in the screenshot above. In this example, the contents of the Super Hires space in memory will be displayed even if the program is currently displaying the text screen (for example).

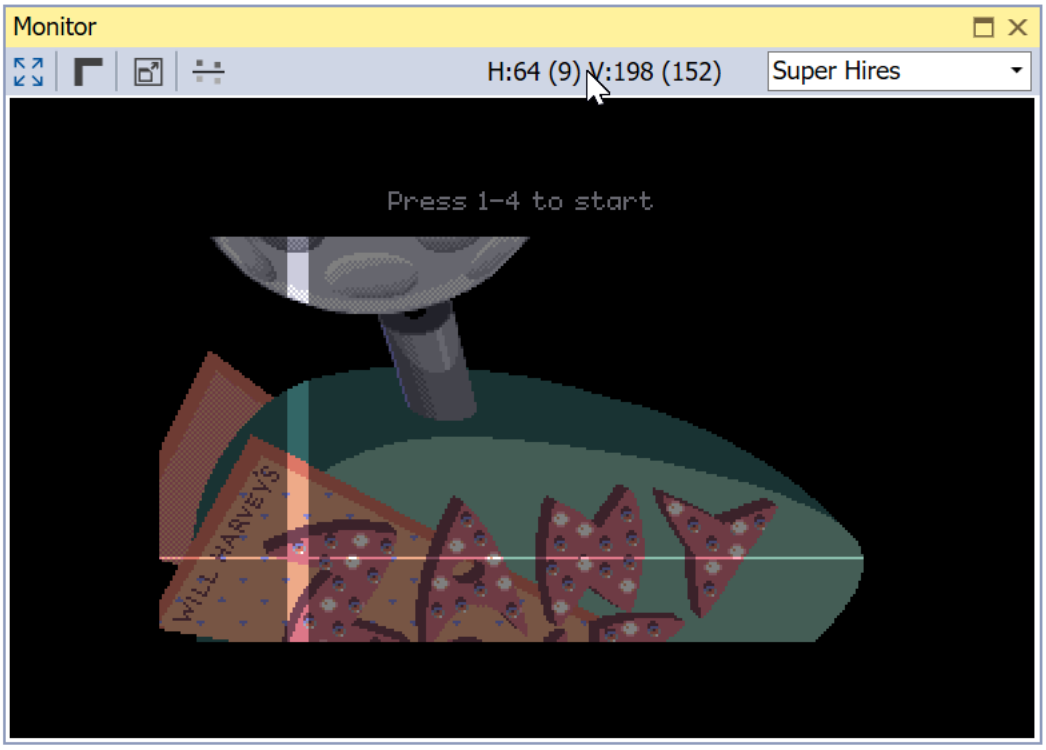
In the top-left corner of the Monitor pane are toolbar options that control how the screen is displayed. The options are, from left to right:

- **Toggle integral scaling on/off** – If selected the screen will be expanded to fill the entire monitor pane. If deselected the screen will only use 1:1, 2:1, 4:1, etc scaling which may mean that the screen does not fill the entire monitor pane.
- **Toggle borders** – This will toggle the border area of the screen on or off. If your focus is just on the screen's contents and you are not interested in the border area then you may want to toggle it off. If you are debugging software that alters the border area then will want to ensure that borders are displayed.
- **Enter fullscreen mode** – This will enable fullscreen mode.
- **Show changes ahead of the beam** – By default the monitor shows the screen as the user would see it. This means that if you change memory in an area where the beam hasn't reached it will not be displayed until the point in time where the beam reaches that part of the screen. If you select this option, then the screen that is displayed will exactly match what is currently held in memory. This can be useful if you want to modify memory and see the results immediately.

In the middle of the monitor pane, the video counters are displayed. The first value is the horizontal video counter (that is typically accessed through $C02F). The second value is the vertical video counter (that is typically accessed through $C02E and $C02F).

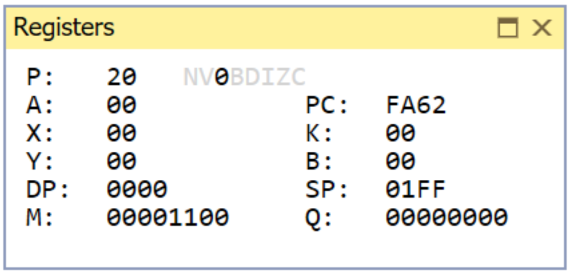
Both of these video counters have their actual hardware value displayed first, and then a user friendly value displayed second. In this example below, the vertical video counter has a hardware value of 198 which is scanline 152 on the screen.

If you hover the mouse over the video counters then the Monitor pane will highlight where the beam is currently up to on the screen.



Viewing and changing register values

The Registers pane allows you to view, and change, the CPU's register values. The Register pane can be displayed through the "View" menu > "Debugging" > "Registers".


Registers

```
P:    20   NV0BDIZC
A:    00              PC:   FA62
X:    00              K:    00
Y:    00              B:    00
DP:   0000            SP:   01FF
M:    00001100        Q:    00000000
```

Displayed are the Processor Flags (P), accumulator (A), X register (X), Y register (Y), Direct Page register (DP), the Machine State pseudo-register (M), Program Counter (PC), Program Bank register (K), Data Bank register (B), Stack Pointer (SP) and the Quagmire pseudo-register (Q).

Next to the hexadecimal value held in the Processor Flags (P) is a breakdown of the individual bits that make up the Processor Flag. If the bit is displayed in light grey it is unset. If the bit is displayed in black it is set.

Bit	Letter	Meaning
7	N	Negative flag (Set = Result negative)
6	V	Overflow flag (Set = Overflow occurred)
5	M / 0	Memory/Accumulator Select (Set = 16-bit, Unset = 8-bit). This is "0" in emulation mode.
4	X / B	Index Register Select (Set = 16-bit, Unset = 8-bit). This is the Break flag in emulation mode.
3	D	Decimal mode (Set = BCD mode, Unset = Binary mode)
2	I	IRQ disable (Set = Interrupts disabled, Unset = Interrupts enabled)
1	Z	Zero flag (Set = Result zero)
0	C	Carry (Set = Carry)

The Machine State and Quagmire pseudo-registers are the same as what are displayed in GSBug.

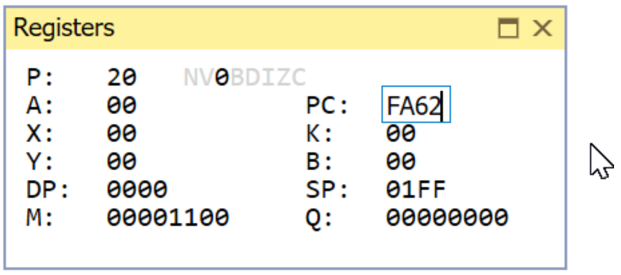
The Machine State pseudo-register is located at $C068 (in any of banks $00, $01, $E0, or $E1) and is used to access a variety of different Mega II softswitches. This register is described in detail in the Apple IIGS Hardware Reference. The table below lists the bits that comprise this pseudo-register.

Bit	Name	Meaning
7	ALTZP	If this bit is 0, bank-switched memory, stack, and zero page are in main memory; if it is 1, they are in auxiliary memory.
6	PAGE2	If this bit is 0, text page 1 is selected; if it is 1, text page 2 is selected.
5	RAMRD	If this bit is 0, main-memory RAM is read-enabled; if it is 1, auxiliary-memory RAM is read-enabled.
4	RAMWRT	If this bit is 0, main-memory RAM is write-enabled; if it is 1, auxiliary-memory RAM is write-enabled.
3	RDROM	If this bit is 0, language-card RAM is read-enabled; if it is 1, language-card ROM is read-enabled.
2	LCBNK2	If this bit is 0, bank 2 language-card RAM (at $D000 through $DFFF) is selected; if it is 1, bank 1 language-card RAM is selected. Switching banks with this bit does not write-enable language-card RAM. Use the L flag both to write-enable RAM and to switch language-card banks.
1	ROMBANK	This bit is reserved; it must equal 0.
0	INTCXROM	If this bit is 0, external ROM (that is, ROM on the circuit board at $Cx00) is active; if it is 1, internal ROM at $Cx00 is active.

The Quagmire pseudo-register is comprised of the lower seven bits of the Shadow register at $C035 and the high bit of the Configuration register at $C036. The table below lists the bits that comprise this pseudo-register.

Bit	Meaning
7	If this bit is 1, high-speed operation is on.
6	If this bit is 1, IOLC (I/O and language card) shadowing is off.
5	This bit is reserved; it must equal 1.
4	If this bit is 1, auxiliary-memory Hi-Res graphics shadowing is off.
3	If this bit is 1, Super Hi-Res graphics shadowing is off.
2	If this bit is 1, Hi-Res graphics page 2 shadowing is off.
1	If this bit is 1, Hi-Res graphics page 1 shadowing is off.
0	If this bit is 1, text page 1 shadowing is off.

You can edit the contents of any register by clicking on the register value. In the example below, the user has clicked on the Program Counter value and can now type in a new value. The value takes effect after you hit Enter.



```
P:   20   NV0BDIZC
A:   00              PC:   FA62
X:   00              K:    00
Y:   00              B:    00
DP:  0000            SP:   01FF
M:   00001100        Q:    00000000
```

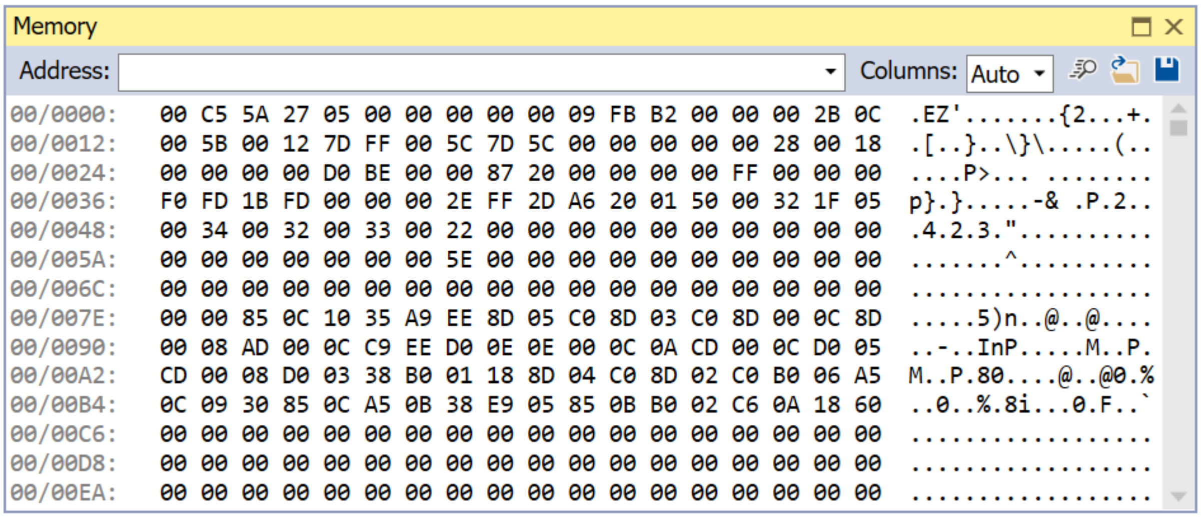
The Processor Flags can be toggled by clicking on the individual letters. For example, if you click the “C” letter it will toggle between Carry Set and Carry Clear.

Viewing and changing memory

The Memory Pane can be used to view the contents of memory, and to modify memory. There are two Memory Panes, which can be enabled through the “View” menu > “Debugging” > “Memory 1” and “Memory 2”.

The Memory Pane displays the hexadecimal contents of the selected memory area on the left, and displays an ASCII decoding of those values on the right. High bits are ignored when decoding ASCII, as is the standard for the Apple II.

The Address edit at the top of the pane can be used to jump to a specific location in memory. For example, typing "04/1234" will jump to the address 04/1234.



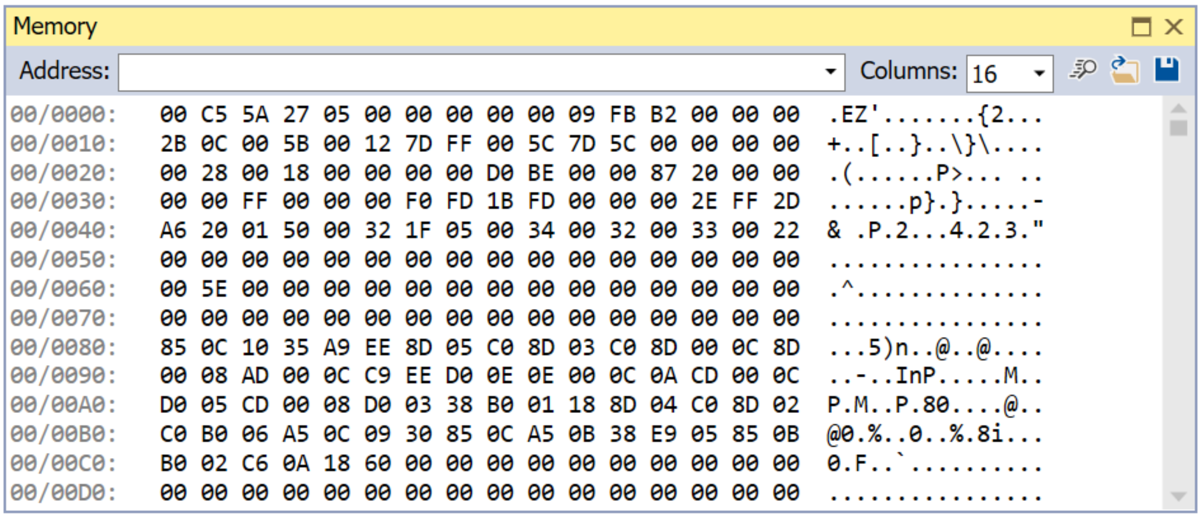
```
00/0000:   00 C5 5A 27 05 00 00 00 00 00 09 FB B2 00 00 00 2B 0C  .EZ'.......{2...+.
00/0012:   00 5B 00 12 7D FF 00 5C 7D 5C 00 00 00 00 00 28 00 18  .[..}..\}\.....(..
00/0024:   00 00 00 00 D0 BE 00 00 87 20 00 00 00 00 FF 00 00 00  ....P>... ........
00/0036:   F0 FD 1B FD 00 00 00 2E FF 2D A6 20 01 50 00 32 1F 05  p}.}.....-& .P.2..
00/0048:   00 34 00 32 00 33 00 22 00 00 00 00 00 00 00 00 00 00  .4.2.3."..........
00/005A:   00 00 00 00 00 00 00 5E 00 00 00 00 00 00 00 00 00 00  .......^..........
00/006C:   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..................
00/007E:   00 00 85 0C 10 35 A9 EE 8D 05 C0 8D 03 C0 8D 00 0C 8D  .....5)n..@..@....
00/0090:   00 08 AD 00 0C C9 EE D0 0E 0E 00 0C 0A CD 00 0C D0 05  ..-..InP.....M..P.
00/00A2:   CD 00 08 D0 03 38 B0 01 18 8D 04 C0 8D 02 C0 B0 06 A5  M..P.80....@..@0.%
00/00B4:   0C 09 30 85 0C A5 0B 38 E9 05 85 0B B0 02 C6 0A 18 60  ..0..%.8i...0.F..`
00/00C6:   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..................
00/00D8:   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..................
00/00EA:   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..................
```

The Memory Pane will update in real-time. As memory values change, they will be displayed.

You can edit memory values by clicking on a hexadecimal value and then typing a new hexadecimal value. The cursor will move as you type, so you can enter in as many bytes as you want.

If you prefer to edit using ASCII values then you can click on the ASCII values on the right-hand side and type replacement values – e.g. typing the letter 'A' on the keyboard will store the hexadecimal value 41 into memory. Just like the hexadecimal edits, the cursor will move within the ASCII section of the pane as you type, so you can enter in as many ASCII characters as you want.

By default, the memory pane will adjust the number of columns it displays to fill the width of the pane. Sometimes it is easier to deal with memory with fixed widths. You can use the Columns drop down to specify a fixed number of columns. In the example below, we have set the Column count to 16, and therefore the memory pane has nicely aligned addresses of 0000, 0010, 0020, and so on.



```
00/0000:   00 C5 5A 27 05 00 00 00 00 00 09 FB B2 00 00 00  .EZ'.......{2...
00/0010:   2B 0C 00 5B 00 12 7D FF 00 5C 7D 5C 00 00 00 00  +..[..}..\}\....
00/0020:   00 28 00 18 00 00 00 00 D0 BE 00 00 87 20 00 00  .(......P>... ..
00/0030:   00 00 FF 00 00 00 F0 FD 1B FD 00 00 00 2E FF 2D  ......p}.}.....-
00/0040:   A6 20 01 50 00 32 1F 05 00 34 00 32 00 33 00 22  & .P.2...4.2.3."
00/0050:   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00/0060:   00 5E 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .^..............
00/0070:   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00/0080:   85 0C 10 35 A9 EE 8D 05 C0 8D 03 C0 8D 00 0C 8D  ...5)n..@..@....
00/0090:   00 08 AD 00 0C C9 EE D0 0E 0E 00 0C 0A CD 00 0C  ..-..InP.....M..
00/00A0:   D0 05 CD 00 08 D0 03 38 B0 01 18 8D 04 C0 8D 02  P.M..P.80....@..
00/00B0:   C0 B0 06 A5 0C 09 30 85 0C A5 0B 38 E9 05 85 0B  @0.%..0..%.8i...
00/00C0:   B0 02 C6 0A 18 60 00 00 00 00 00 00 00 00 00 00  0.F..`..........
00/00D0:   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

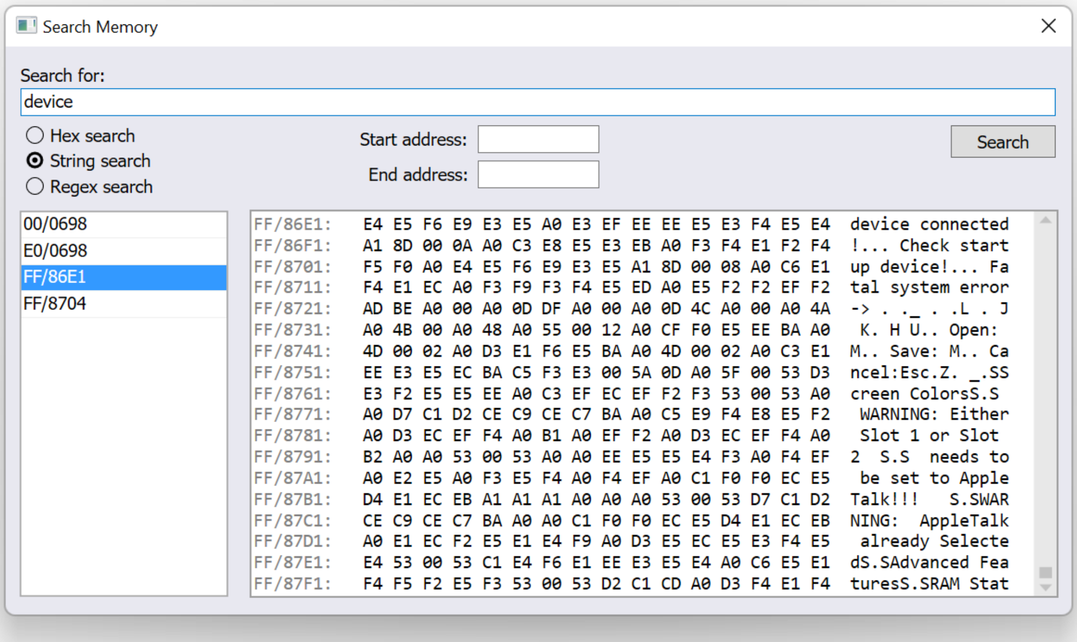
There are three toolbar buttons on the top-right of the memory pane:

- **Search memory**
- **Load memory from disk**

- **Save memory to disk**

Search Memory

Clicking the Search Memory button will bring up a dialog which allows you to search the contents of main memory for the specified sequence of bytes, a specified string, or for memory that matches a regular expression.



Search for: device

( ) Hex search
(●) String search
( ) Regex search

Start address:
End address:

```
00/0698
E0/0698
FF/86E1
FF/8704
```

```
FF/86E1:   E4 E5 F6 E9 E3 E5 A0 E3 EF EE EE E5 E3 F4 E5 E4  device connected
FF/86F1:   A1 8D 00 0A A0 C3 E8 E5 E3 EB A0 F3 F4 E1 F2 F4  !... Check start
FF/8701:   F5 F0 A0 E4 E5 F6 E9 E3 E5 A1 8D 00 08 A0 C6 E1  up device!... Fa
FF/8711:   F4 E1 EC A0 F3 F9 F3 F4 E5 ED A0 E5 F2 F2 EF F2  tal system error
FF/8721:   AD BE A0 00 A0 0D DF A0 00 A0 0D 4C A0 00 A0 4A  -> . ._ . .L . J
FF/8731:   A0 4B 00 A0 48 A0 55 00 12 A0 CF F0 E5 EE BA A0   K. H U.. Open: 
FF/8741:   4D 00 02 A0 D3 E1 F6 E5 BA A0 4D 00 02 A0 C3 E1  M.. Save: M.. Ca
FF/8751:   EE E3 E5 EC BA C5 F3 E3 00 5A 0D A0 5F 00 53 D3  ncel:Esc.Z. _.SS
FF/8761:   E3 F2 E5 E5 EE A0 C3 EF EC EF F2 F3 53 00 53 A0  creen ColorsS.S 
FF/8771:   A0 D7 C1 D2 CE C9 CE C7 BA A0 C5 E9 F4 E8 E5 F2   WARNING: Either
FF/8781:   A0 D3 EC EF F4 A0 B1 A0 EF F2 A0 D3 EC EF F4 A0   Slot 1 or Slot 
FF/8791:   B2 A0 A0 53 00 53 A0 A0 EE E5 E5 E4 F3 A0 F4 EF  2  S.S  needs to
FF/87A1:   A0 E2 E5 A0 F3 E5 F4 A0 F4 EF A0 C1 F0 F0 EC E5   be set to Apple
FF/87B1:   D4 E1 EC EB A1 A1 A1 A0 A0 A0 53 00 53 D7 C1 D2  Talk!!!   S.SWAR
FF/87C1:   CE C9 CE C7 BA A0 A0 C1 F0 F0 EC E5 D4 E1 EC EB  NING:  AppleTalk
FF/87D1:   A0 E1 EC F2 E5 E1 E4 F9 A0 D3 E5 EC E5 E3 F4 E5   already Selecte
FF/87E1:   E4 53 00 53 C1 E4 F6 E1 EE E3 E5 E4 A0 C6 E5 E1  dS.SAdvanced Fea
FF/87F1:   F4 F5 F2 E5 F3 53 00 53 D2 C1 CD A0 D3 F4 E1 F4  turesS.SRAM Stat
```

The search can, optionally, be restricted to a given address range. If the Start Address is populated then the search will begin starting at the specified address. If the End Address is populated then the search will terminate at the specified address. If neither address is specified then the entire contents of memory will be searched.

If you search for a sequence of bytes then you must enter the hexadecimal byte values into the search field. You can separate bytes using whitespace, but this is not required. The '?' character can be used as a wildcard.

For example, "11 2? ?3 ?? 44" will search for "11" followed by any byte with a high nibble of '2', followed by any byte with a low nibble of '3', followed by any byte, followed by "44".

If you search for a string then a case sensitive search of memory is performed. The high bit is ignored when performing string searches.

Search results are displayed in a list on the left-hand side of the window. If there are over 1000 matches then the search will terminate after the 1000th match. You can click on each address in the search results to see a preview of the memory at that address.

Load Memory

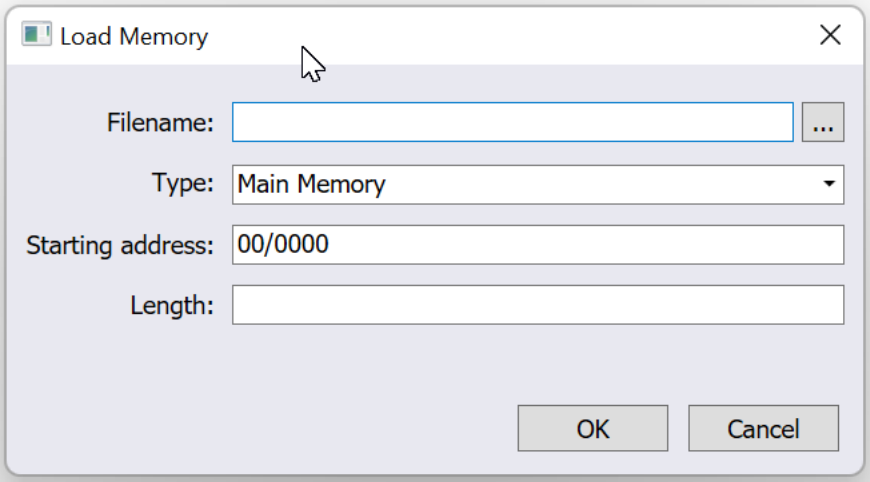
Clicking the Load Memory button will bring up a dialog which allows you to specify the file to load into memory and the address where the file's contents should be stored.

Firstly, either enter the filename of the file to load, or alternatively click on the "…" button to select the file.

Next, select which memory area the file should be loaded into – the computer's main memory, the 64KB of sound RAM used by the Ensoniq, or the 96 bytes of memory used by the ADB microcontroller.

Next, enter the starting address in memory where the file should be loaded.

The final field contains the length of data to be stored. By default, this will be the length of the file being loaded, but this can be edited to reduce the number of bytes being loaded.


Load Memory

Filename:

Type: Main Memory

Starting address: 00/0000

Length:

The length field is treated as a decimal number, unless you prefix the length with the dollar sign (‘$’). For example, “1234” is 1234 bytes while “$1234” is 4660 bytes.

Save Memory

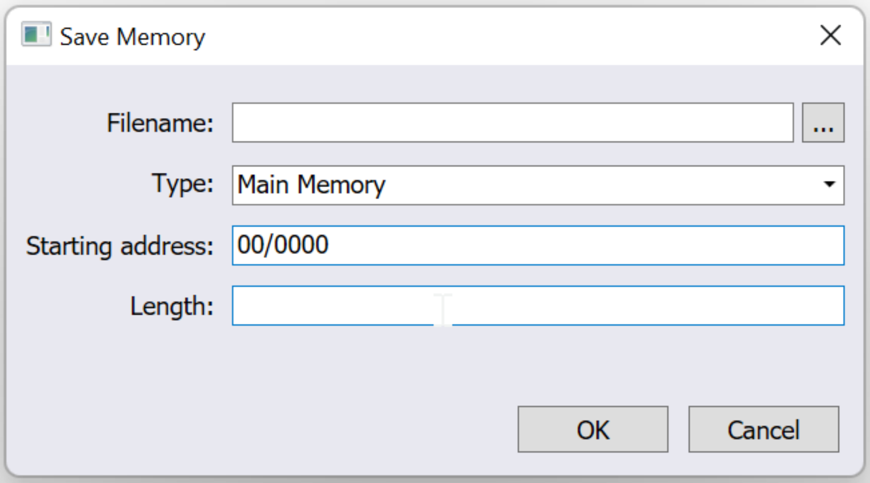
Clicking the Save Memory button will bring up a dialog which allows you to specify a file on disk where memory will be written to.

Firstly, either enter the filename of the file to save to, or alternatively click on the “…” button to select a file.

Next, select which memory area should be saved.

Next, select the starting address in memory which will be written to the file.

The final field contains the number of bytes to be written to the file.


Save Memory

Filename:

Type: Main Memory

Starting address: 00/0000

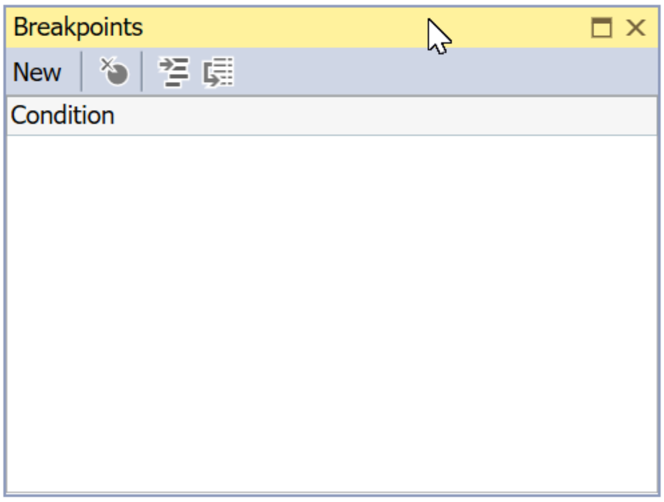
Length:

Setting breakpoints

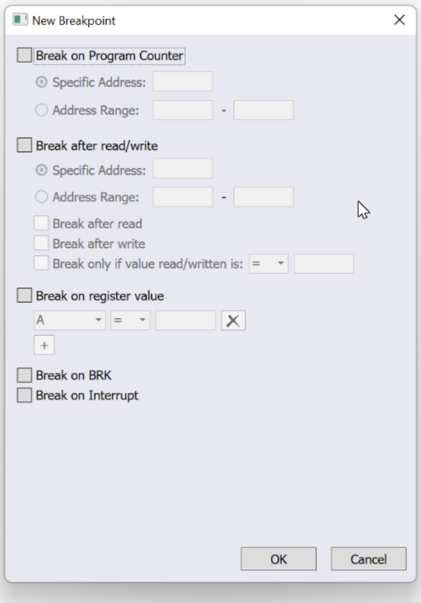
There are two main ways of setting breakpoints.

Firstly, you can select a line in either the disassembly view, or in the source code view, and press F9 on the keyboard to toggle a breakpoint at that address. The “Toggle breakpoint” option is also available in the popup menu that is presented if you right-click on a line in either of these views.

Secondly, you can use the Breakpoints Pane to create more complex breakpoints. The Breakpoints Pane can be enabled through the “View” menu > “Debugging” > “Breakpoints”.



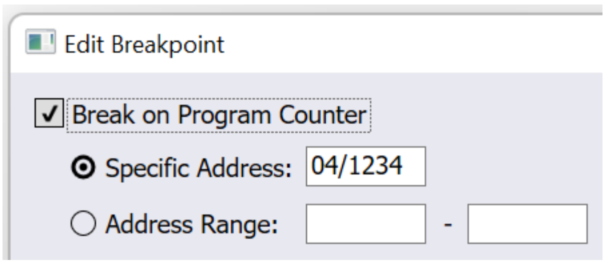
In the Breakpoints Pane it will display a list of all active breakpoints. Click the “New” button in the top left of the pane to bring up the “New Breakpoint” dialog.



There are a number of different conditions that can be specified using this dialog. You can specify a single condition, or you can specify a combination of conditions. All conditions must be met for the breakpoint to be triggered.

Break on Program Counter

The “Break on Program Counter” option creates a breakpoint that will trigger when the Program Counter (PC) register in the CPU is set to a specific value (before executing code at that address), or alternatively when the PC is within a specific address range.

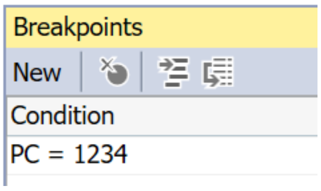

Edit Breakpoint

[✓] Break on Program Counter

(●) Specific Address: 04/1234

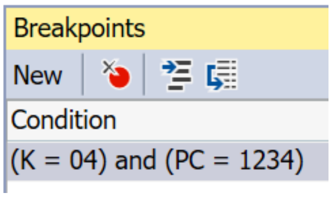
( ) Address Range: ___ - ___

If you specify a 16 bit address (for example, “1234”) then the breakpoint will be triggered on any bank. So regardless of whether we are running in bank 00 at address 1234, or if we are running in bank 04 at address 1234, it will trigger.



Condition
PC = 1234

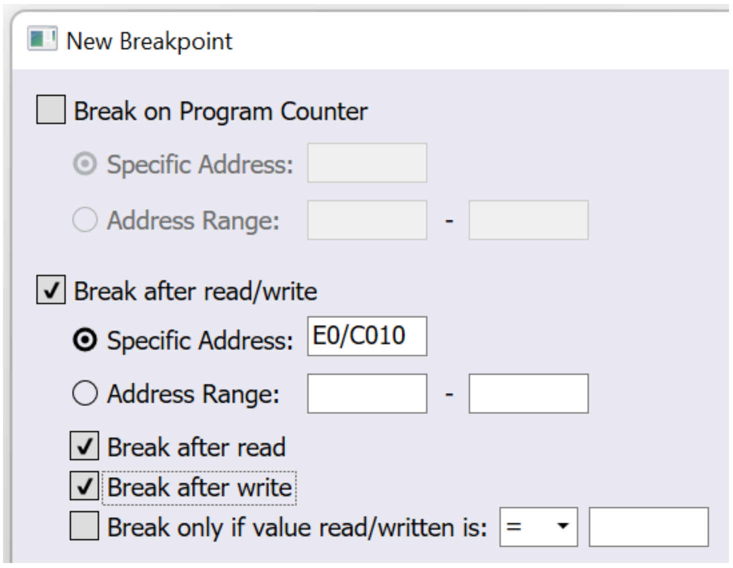
If you specify a 24 bit address (for example, “04/1234”) then the breakpoint will only be triggered for that specific bank and address.



Condition
(K = 04) and (PC = 1234)

Break after read/write

The “Break after read/write” option creates a breakpoint that will trigger when memory is accessed at a specific address, or when memory is accessed from a specified address range. Unlike the “Break on Program Counter” option, this breakpoint triggers *after* the code that has performed a read and/or write.


New Breakpoint

[ ] Break on Program Counter

(●) Specific Address: ___

( ) Address Range: ___ - ___

[✓] Break after read/write

(●) Specific Address: E0/C010

( ) Address Range: ___ - ___

[✓] Break after read

[✓] Break after write

[ ] Break only if value read/written is: = ___

You must at a minimum tick either “Break after read” or “Break after write”. You can tick both if you want. By ticking both, it will break after either a read or a write to the address.

The final checkbox (“Break only if value read/written is”) can restrict this breakpoint further. If this option is checked then the breakpoint will only trigger if memory was accessed at the specified address *and* the value that was read from that address, or written to that address, meets specific additional criteria.

You can specify that the address that is read/written must be equal to a given value, not equal to a given value, or less than or greater than a given value. The value that you specify can be either an 8-bit value, or a 16-bit value.

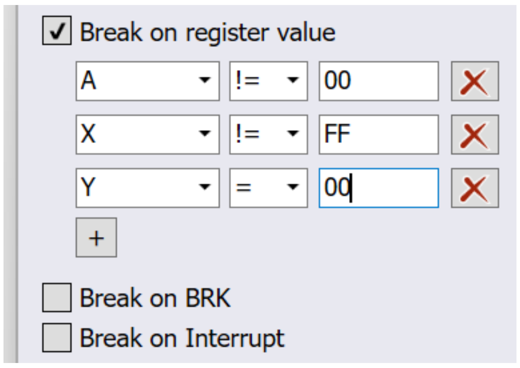
Break on register value

The “Break on register value” option creates a breakpoint that will trigger only when one or more registers have values that meet specified criteria.

The leftmost drop down picks the register that you want to test. The middle drop down picks the comparison type (the register must equal the specified value, must be different from the specified value, or must be less than or greater than the specified value). The rightmost drop down picks the value that the register will be compared against. The value that you specify can be either an 8-bit value, or a 16-bit value.

You can click the “+” button to add additional register conditions. All register conditions must be met for the breakpoint to trigger.

If you want to remove a register condition, then click the “X” button on the right of the register condition you wish to delete.


Break on register value
A != 00
X != FF
Y = 00
Break on BRK
Break on Interrupt

“Break on register value” conditions are often combined with “Break on Program Counter” conditions. In this case, the breakpoint will only trigger if the PC is at the specified address *and* the registers hold the specified values at that point in time.

Break on BRK

The “Break on BRK” option will create a breakpoint that will trigger when a BRK opcode is encountered.

Break on Interrupt

The “Break on Interrupt” option will create a breakpoint that will trigger when an interrupt is triggered.

Deleting breakpoints

Breakpoints can be deleted in the Breakpoints Pane. Select the breakpoint you want to delete and then either press the Delete key, or click on the “Delete breakpoint” button in the toolbar.

Memory Map

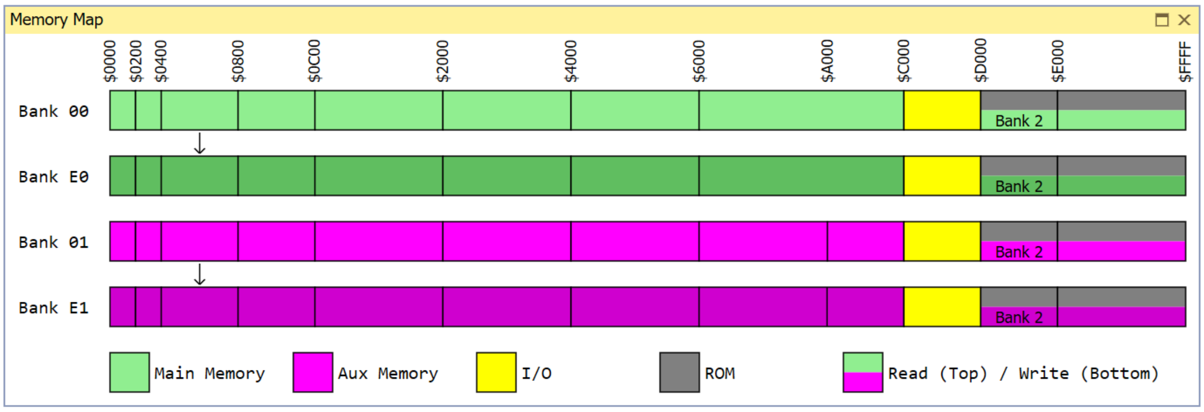
The Apple II series of computer, like other computers that began life with a 64K address space, has a configurable memory map. Selected ranges of memory can be swapped between main memory, auxiliary memory, language card banks, I/O space and ROM. Selected ranges can even be configured to be asymmetric – i.e. reads access different memory to writes. This allows an Apple II+ to access 64K of RAM and 12K of ROM. The Apple IIe builds on this in order to access 128K of RAM and 12K of ROM.

The Apple IIGS takes this one step further by having 128K of fast RAM (bank 0 and bank 1) and 128K of slow RAM (bank E0 and bank E1), with all of the same configurable memory map options across all 4 banks plus a few new features – for example:

- The $C000-$CFFF I/O space can be replaced by RAM.
- Reverse mapping can happen – so auxiliary memory can be remapped to main memory.
- Memory can be "shadowed" – so that a write to bank 0 will also result in a write to bank E0 and/or a write to bank 1 will result in a write to bank E1.

All of this can make it non-trivial to determine what memory map is active when you break CPU execution at a random point in time. The Memory Map pane aims to give you a way to view the entire mapping quickly in a glance.

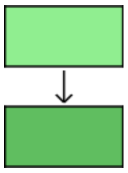
The Memory Map pane can be displayed through the "View" menu > "Debugging" > "Memory Map".



The four banks are shown as 4 rows. Each row is divided into a series of chunks from $0000 to $FFFF. Each chunk will be colour coded to indicate its current mapping.

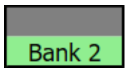
For example, the first chunk of memory is $0000-$0200, which is the zero page and stack space. This chunk can be switched between main memory and aux memory using the SETALTZP ($C009) softswitch. If this chunk is Green it indicates that the chunk is mapped to main memory. If the chunk is Magenta it indicates that the chunk is mapped to auxiliary memory.

Shadowing is indicated by an arrow from a chunk of memory in Bank 00 to Bank E0, or from Bank 01 to Bank E1. If there is no arrow then it indicates that Shadowing has been disabled for that chunk of memory.



In the example above, there are two chunks of memory that are being shadowed – Text Page 1 ($0400-$07FF) in main memory and the same chunk in auxiliary memory. Any writes to address range $0400-$07FF in bank 00 will be shadowed to address range $0400-$07FF in bank E0, and any writes to address range $0400-$07FF in bank 01 will be shadowed to address range $0400-$07FF in bank E1.

When a chunk of memory is configured differently for reads and writes it is displayed as a split chunk. The top half of the chunk indicates where reads will be mapped to, and the bottom half of the chunk indicates where writes will be mapped to.



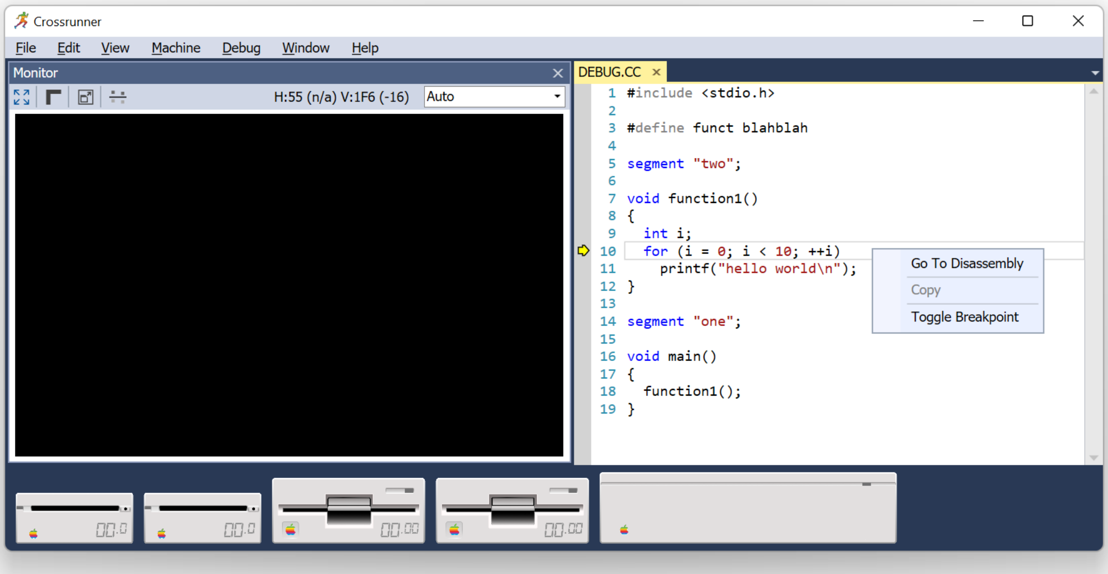
In the example, above the top half of the chunk is coloured grey indicating that reads will come from ROM. The bottom half of the chunk is coloured green indicating that writes will be mapped to RAM.

In this particular example, the memory space is $D000-$DFFF which can be mapped to two different areas in RAM - Language Card Bank 1 or Language Card Bank 2. The chunk is therefore labelled to indicate which Language Card Bank is current mapped.

Debugging High Level Languages

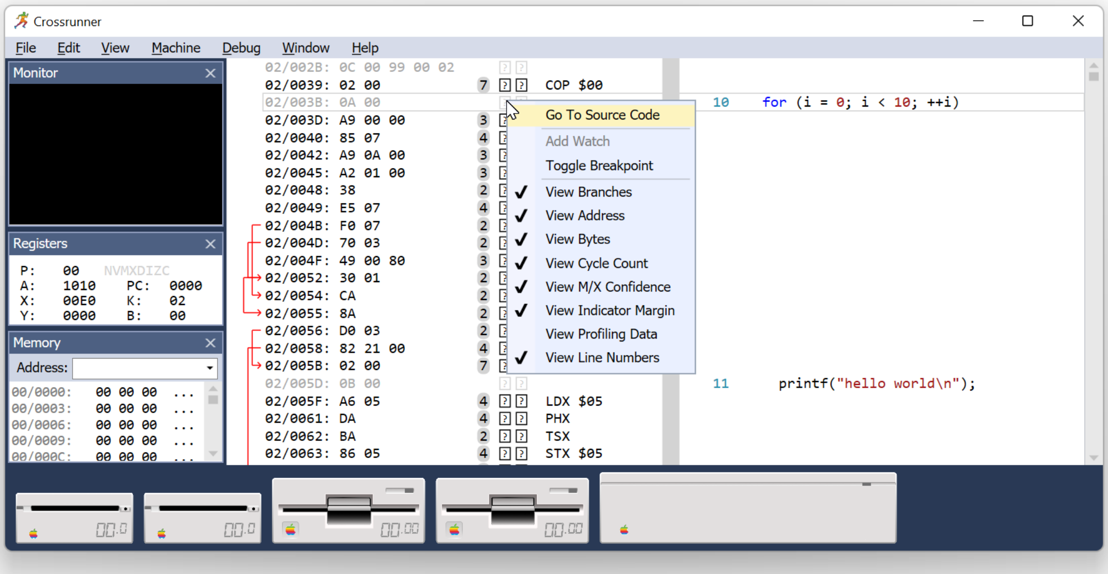
Crossrunner also supports a higher level debugging mode for languages such as ORCA/C and ORCA/Pascal. In this mode you debug using the C or Pascal source code, and step through it line-by-line rather than instruction-by-instruction.

You can at any time switch between the high-level mode and the disassembly-level mode. Simply right-click on a line of C/Pascal and select “Go To Disassembly”. This will switch to the disassembly-level mode and highlight the first instruction that corresponds to that line of C/Pascal. For each line of C/Pascal there will be many assembly language instructions.



```
#include <stdio.h>

#define funct blahblah

segment "two";

void function1()
{
  int i;
  for (i = 0; i < 10; ++i)
    printf("hello world\n");
}

segment "one";

void main()
{
  function1();
}
```

You can also go in the opposite direction. Simply right-click on a line containing C/Pascal source code, and select “Go To Source”. This will switch to the source-level mode and highlight the line of C/Pascal source code that begins with that COP instruction.



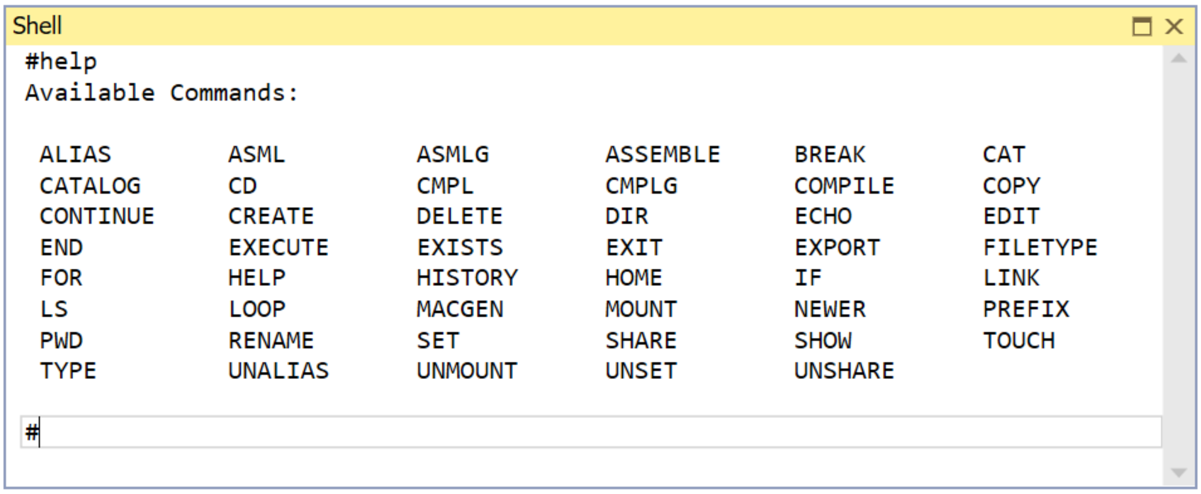
```
P:    00    NVMXDIZC
A:    1010    PC:   0000
X:    00E0    K:    02
Y:    0000    B:    00
```

```
00/0000:   00 00 00   ...
00/0003:   00 00 00   ...
00/0006:   00 00 00   ...
00/0009:   00 00 00   ...
00/000C:   00 00 00   ...
```

```
02/002B: 0C 00 99 00 02
02/0039: 02 00         7    COP $00
02/003B: 0A 00                              for (i = 0; i < 10; ++i)
02/003D: A9 00 00      3
02/0040: 85 07         4
02/0042: A9 0A 00      3
02/0045: A2 01 00      3
02/0048: 38            2
02/0049: E5 07         4
02/004B: F0 07         2
02/004D: 70 03         2
02/004F: 49 00 80      3
02/0052: 30 01         2
02/0054: CA            2
02/0055: 8A            2
02/0056: D0 03         2
02/0058: 82 21 00      4
02/005B: 02 00         7
02/005D: 0B 00                                printf("hello world\n");
02/005F: A6 05         4    LDX $05
02/0061: DA            4    PHX
02/0062: BA            2    TSX
02/0063: 86 05         4    STX $05
```

Chapter 4
Shell

The Shell pane will be familiar to anyone who has used ORCA/Shell. The Shell pane can be opened through the “View” menu > “Shell”.



```
#help
Available Commands:

 ALIAS        ASML         ASMLG        ASSEMBLE     BREAK        CAT
 CATALOG      CD           CMPL         CMPLG        COMPILE      COPY
 CONTINUE     CREATE       DELETE       DIR          ECHO         EDIT
 END          EXECUTE      EXISTS       EXIT         EXPORT       FILETYPE
 FOR          HELP         HISTORY      HOME         IF           LINK
 LS           LOOP         MACGEN       MOUNT        NEWER        PREFIX
 PWD          RENAME       SET          SHARE        SHOW         TOUCH
 TYPE         UNALIAS      UNMOUNT      UNSET        UNSHARE

#
```

There are two main uses for the Shell:

1. To be able to build existing software that relied on ORCA/Shell build scripts.
2. Reducing the time to perform certain disk image management tasks.

Getting Help on commands

The first command you should familiar yourself with is the `HELP` command. Typing HELP by itself will list all commands available within the shell. You can then type `HELP <command>` to get more detailed help on a particular command.

Mounting disk images

The next command that you should familiarise yourself with is the `MOUNT` command, which is a new command not found in ORCA/Shell. The MOUNT command allows you to mount disk images. Typing `MOUNT <filename>` will mount the specified disk image. For example, if you have a ProDOS disk image called “Source.2MG” which contains the ProDOS volume SOURCE, you can type “MOUNT Source.2MG” to mount the volume :SOURCE.

The Shell lets you mix and match between the native filesystem (e.g. Windows filesystem), ProDOS filesystems and DOS 3.3 filesystems. The Windows filesystem can be referred to using GS/OS pathnames, which are colon separated. For example, “C:\Windows\filename” would be referenced using “:C:Windows:filename”.

This mixing and matching means that you can use the ORCA/Shell `COPY` command to copy files from the native filesystem to a disk image, from a disk image to the native filesystem or from a disk image to another disk image. It allows you to run Apple IIGS command line utilities and redirect the output to a file on the native filesystem. It allows you to run a native (e.g. Windows) command line utility and redirect the output to a file on a ProDOS disk image.

If you want to unmount a disk image, then you can type `UNMOUNT <volume>`.

Wildcards

The Shell support ORCA/Shell wildcards. The ‘=’ character is a wildcard that matches zero or more characters.

Because the Shell allows wildcards, this can make certain bulk tasks like changing the filetype and auxtype of a large number of files achievable in less time than other tools (such as Ciderpress).

Command history

You can use the up and down arrow keys to up through your command history. This makes it easier to reissue commands you have previously typed.

You can also use the `HISTORY` command to display the last 20 commands that were executed.

Filename completion

The Shell supports filename completion using the tab key. This allows you to type the first few letters of a filename and then press ‘tab’ to complete the filename, saving you some typing. If there are multiple possible matches for the completion then you can keep pressing the tab key to cycle through the options.

Prefixes

The Shell supports GS/OS prefixes. The ORCA compilers and linkers use these GS/OS prefixes to specify the location for supporting files that they rely on.

Prefixes can be very useful to standardise on directory structures, and to save you keystrokes. For example, to open up the C header “limits.h”, you can type `EDIT 13:ORCACDefs:limits.h`.

Prefix	Description	Default value
@	User’s folder	Native filesystem documents directory
*	Boot prefix	Native filesystem boot directory (e.g. :C:)
8	Current prefix	Native filesystem current working directory
9	Application	Native filesystem directory of the Crossrunner application
10	Standard Input	.CONSOLE
11	Standard Output	.CONSOLE
12	Error Output	.CONSOLE
13	ORCA library	Configured under “Libraries Path” in the Preferences dialog
14	ORCA work	The parent directory of 16:
15	ORCA shell	Configured under “Shell Path” in the Preferences dialog
16	ORCA language	Configured under “Languages Path” in the Preferences dialog
17	ORCA utility	Configured under “Utilities Path” in the Preferences dialog

You can use the `SHOW PREFIX` command to display all prefix values that are set.

You can use the `PREFIX <number> <directory>` command to change the prefix assigned to a given number.

Device Numbers

The Shell supports GS/OS device numbers. Every volume, whether it is a native volume or a disk image, is assigned a volume number. You can get a list of all device numbers by typing the command `SHOW UNITS`.

Similar to prefixes, device number can be used in any command, for example `EDIT .d5:filename`.

Chapter 5
Merlin and ORCA Integration

Crossrunner allows you to debug programs you have written using either the Merlin assembler (Merlin-32) or using the ORCA suite of compilers and assemblers (ORCA/M, ORCA/C and ORCA/Pascal).

Using this feature you will be able to see your source code inside Crossrunner, complete with all labels, variable names and comments. This makes it a lot easier to debug than just using the standard disassembly output.

❖ Merlin-32 is a command line assembler that is available (for free) on Windows, MacOS and Linux. It is a modern version of the popular Apple II Merlin 16+ assembler originally by Glen Bredon.

  Merlin-32 v1.0 can be downloaded at
  http://www.brutaldeluxe.fr/products/crossdevtools/merlin/index.html

  Later versions of Merlin-32 can be downloaded at https://github.com/apple2accumulator/merlin32

❖ ORCA/M, ORCA/C and ORCA/Pascal are still commercial products.

  They can be purchased as part of the "Opus][: The Software" collection from https://juiced.gs/store/opus-ii-software/

Merlin-32

Compiling

When compiling using Merlin-32 you must specify the "-V" (verbose) build switch. This will cause Merlin-32 to generate the xxx_Output.txt files which Crossrunner uses.

For example:

```
Merlin32 -V App.s
```

Running

You can now run your program using Crossrunner. You must use the following command line switches:

1. The filename for your program
2. `-source,` followed by all assembler outputs
3. `-map,` followed by the *Master* source file (or the *Link* file)
4. `-Debug`
5. `-CompatibilityLayer`

In the example above, Merlin32 has output the following files:

- GTEZelda_S02_MAINSEG_Output.txt
- GTEZelda_S03__Output.txt
- GTEZelda_S04__Output.txt
- GTEZelda_S05__Output.txt
- GTEZelda_S06__Output.txt
- GTEZelda_S07__Output.txt

And it has generated an executable called:

- GTEZelda

Therefore our command line would be:

```
    Crossrunner.exe GTEZelda -source GTEZelda_S02_MAINSEG_Output.txt
GTEZelda_S03__Output.txt GTEZelda_S04__Output.txt GTEZelda_S05__Output.txt
GTEZelda_S06__Output.txt GTEZelda_S07__Output.txt -map App.s -Debug -
CompatibilityLayer
```

ORCA/M

Compiling

When compiling using ORCA/M you must use the `+L +S` switches, and you must redirect the output to a file.

For example (using Golden Gate):

```
    iix assemble +L +S game.asm keep=obj:game > out\game.asm
```

Using Crossrunner's Shell:

```
    assemble +L +S game.asm keep=obj:game > out\game.asm
```

Using Crossrunner's standalone Compile command line utility:

```
    compile +L +S game.asm keep=obj:game > out\game.asm
```

❖ Golden Gate is a compatibility layer for ORCA and GNO/ME command line programs to allow you to run them natively from Windows, MacOS or Linux.

Linking

Next, you must specify the `+L` switch when linking, and you must redirect the output to a file.

For example (using Golden Gate):

```
    iix link +L obj:game keep=mygame > out\game.map
```

Using Crossrunner's Shell or Crossrunner's standalone Link command line utility:

```
    link +L obj:game keep=mygame > out\game.map
```

Running

You can now run your program using Crossrunner. You must use the following command line switches:

1. The filename for your program
2. `-source`, followed by all assembler outputs
3. `-map`, followed by the linker output
4. `-Debug`
5. `-CompatibilityLayer`

Continuing the examples above (under Compiling and Linking) this would result in the following command line:

```
    Crossrunner.exe MYGAME -source out\game.asm -map out\game.map -Debug
-CompatibilityLayer
```

An example with more source code files would look like:

```
    Crossrunner.exe MYPROGRAM -source out\file1.asm out\file2.asm
out\file3.asm -map out\program.map -Debug -CompatibilityLayer
```

ORCA/C and ORCA/Pascal

Compiling

When compiling using ORCA/C or ORCA/Pascal you must use the `+D` switch.

For example (using Golden Gate):

```
iix compile +D game.cc keep=obj:game
```

Using Crossrunner's Shell or Crossrunner's standalone Compile command line utility:

```
compile +D game.cc keep=obj:game
```

Linking

Next, you must specify the `+L` switch when linking, and you must redirect the output to a file.

For example (using Golden Gate):

```
iix link +L obj:game keep=mygame > out\game.map
```

Using Crossrunner's Shell or Crossrunner's standalone Link command line utility:

```
link +L obj:game keep=mygame > out\game.map
```

Running

You can now run your program using Crossrunner. You must use the following command line switches:

1. The filename for your program
2. `-source`, followed by all C source code files
3. `-map`, followed by the linker output
4. `-DebugSource`
5. `-CompatibilityLayer`

Continuing the examples above (under Compiling and Linking) this would result in the following command line:

```
Crossrunner.exe MYGAME -source game.cc -map out\game.map -DebugSource
-CompatibilityLayer
```

An example with more source code files would look like:

```
Crossrunner.exe MYPROGRAM -source file1.cc file2.cc file3.cc -map
out\program.map -DebugSource -CompatibilityLayer
```

Chapter 6
Scripting

Crossrunner has a built-in scripting engine that allows you to hook into the emulator in order to add custom actions or to leverage core functionality to perform tasks.

The scripting engine uses Wren for the scripting language. The Wren website (http://wren.io) has an introductory guide to the scripting language if you are not familiar with it.

After you type in a script into the Script pane, you need to click the "Run Script" button in the toolbar of the Script pane to execute the script. If your script is a simple, linear, script – for example:

```
System.print(1 + 1)
```

then the script will finish immediately and output the result immediately. After running this script you can click the "Stop" button in the toolbar, but this does not impact the script as the script has already finished all its processing.

However, as we will discuss later (under "Handling Events") you will need to leave scripts that install event handlers in a "Running" state in order for them to receive events.

Accessing the Emulator State

CPU Registers

In order to access the CPU state, the built-in `Register` class can be used. This class has the following fields:

- `A` – 16 bit accumulator. Note that this will always return both the A and B accumulator values as a single 16 bit result regardless of whether emulation mode is enabled, or whether the processor flags are set to 8 bit or 16 bit.
- `A8` – 8 bit accumulator. Note that this will always return only the A accumulator (low 8 bits) regardless of whether emulation mode is enabled, or whether the processor flags are set to 8 bit or 16 bit.
- `X` – X register.
- `Y` – Y register.
- `PC` – the Program Counter
- `SP` – the Stack Pointer
- `P` – the processor flags
- `DP` – the Direct Page register
- `K` – the Program Bank Register. This can also be accessed through the `PBR` alias.
- `DBR` – the Data Bank Register.
- `emulationMode` – a `Bool` value that will be `true` if emulation mode is enabled.

All of these fields support both reading the current values of the register, as well as modifying the current value of the register.

An example of using the `Register` class is shown below:

```
if (Register.PC == 0x1234) {
  Register.X = Register.Y
}
```

In the example above, the Program Counter is tested to see if it is at address 1234 (in hexadecimal). If it is, then the X register is set to be the value contained in the Y register.

Video Graphics Controller state

In order to access the Video Graphics Controller (VGC) state, the built-in `Video` class can be used. This class has the following fields:

- `scanline` – the current scanline that the electron beam is on. This will be a value between 0 and 199 when the beam is scanning the standard screen area. The value will be negative if the beam is scanning the border region above the screen area, and will be >= 200 when scanning the border region below the screen area.
- `horizCounter` – the horizontal video counter. This is the same as the 7-bit value held in the `$C02F` softswitch.
- `vertCounter` – the vertical video counter. This is the same as the 9-bit value held in the `$C02E` and `$C02F` softswitches.

All of these fields are read only.

Accessing memory

The Apple IIGS has three distinct areas of memory – main memory, sound (Ensoniq DOC) RAM, and the memory within the ADB microcontroller.

The `Memory` class is used to read/write main memory. The `SoundRAM` class is used to read/write to sound RAM. And finally, the `ADBMemory` class is used to read/write to the M50740/M50741 microcontroller's memory.

An example of reading and writing to memory is shown below:

```
// copy a byte from address 00/4000 to address 00/2000
Memory[0x2000] = Memory[0x4000]

// set the first byte in Sound RAM to the hexadecimal value 12
SoundRAM[0x0000] = 0x12
```

Accessing softswitches

The `Softswitch` class can be used to read softswitches. The following table lists the field names, and the I/O address that it corresponds to. For example, `Softswitch.KBD` is equivalent to reading softswitch $C000.

All softswitches are read-only and have no side effects. For example, `Softswitch.KBDSTRB` will return the "Any Key Down" flag and key information, but it will not clear the keystrobe.

KBD	$C000
KBDSTRB	$C010
RDLCBNK2	$C011
RDLCRAM	$C012
RDRAMRD	$C013
RDRAMWRT	$C014
RDCXROM	$C015
RDALTZP	$C016
RDC3ROM	$C017
RD80COL	$C018
RDVBLBAR	$C019
RDTEXT	$C01A
RDMIX	$C01B
RDPAGE2	$C01C
RDHIRES	$C01D
ALTCHARSET	$C01E
RD80VID	$C01F
TBCOLOR	$C022
VGCINT	$C023
MOUSEDATA	$C024

SLTROMSEL	$C02D
VERTCNT	$C02E
HORIZCNT	$C02F
DISKREG	$C031
CLOCKDATA	$C033
CLOCKCTL	$C034
SHADOW	$C035
CYAREG	$C036
DMAREG	$C037
SOUNDCTL	$C03C
SOUNDDATA	$C03D
SOUNDADRL	$C03E
SOUNDADRH	$C03F
INTEN	$C041
DIAGTYPE	$C046
BUTN3	$C060
BUTN0	$C061
BUTN1	$C062
BUTN2	$C063
PADDL0	$C064

KEYMODREG	$C025
DATAREG	$C026
KMSTATUS	$C027
NEWVIDEO	$C029
LANGSEL	$C02B

PADDL1	$C065
PADDL2	$C066
PADDL3	$C067
STATEREG	$C068

Changing the emulator state

Currently the only operation that is available is to change the emulator from a halted state to a running state. This is accomplished with the following code:

```
Emulator.run()
```

This is useful when combined with an `onBreakpoint` event handler. This allows you to react to the breakpoint and then either conditionally break or continue running.

Examining the CPU instruction history

Crossrunner keeps track of the last 1,000,000 CPU instructions that were executed. You can use the `History` class to examine this data. The class has the following operators, methods and fields:

- `History[index]` – the index operator allows you to access the previous CPU instructions. An index value of 0 will return the last CPU instruction that was executed, while an index value of 999,999 will access the instruction that was executed 999,999 instructions previously. You can only read from the CPU instruction history – you cannot modify previous instructions.
- `size` – this field will return the total number of instructions available in the history. After you have been running for a short period this will max out at 1,000,000 instructions, but shortly after boot this field will be a smaller value right until the point when the history buffer reaches its limit (at which point older instructions will be replaced with newer instructions).
- `markedEntry` – this field is both readable and writeable. This allows you to place a marker within the instruction history, and then to later read the offset where that marker has moved to. Because the instruction history changes as new instructions are issued this is useful, for example, to mark the point where you have processed up to. Then, later on, you can continue processing from that point onwards.

When using the index operator to access a historical entry, the following fields can be read:

- `A` – 16 bit accumulator. Note that this will always return both the A and B accumulator values as a single 16 bit result regardless of whether emulation mode was enabled, or whether the processor flags were set to 8 bit or 16 bit.
- `X` – X register.
- `Y` – Y register.
- `PC` – the Program Counter
- `SP` – the Stack Pointer
- `P` – the processor flags
- `DP` – the Direct Page register
- `K` – the Program Bank Register. This can also be accessed through the `PBR` alias.
- `DBR` – the Data Bank Register.
- `emulationMode` – a `Bool` value that will be `true` if emulation mode was enabled.
- `Q` – the Quagmire pseudo-register
- `RDLCBNK2` – the RDLCBNK2 softswitch ($C011)
- `RDLCRAM` – the RDLCRAM softswitch ($C012)
- `RDRAMRD` – the RDRAMRD softswitch ($C013)
- `RDRAMWRT` – the RDRAMWRT softswitch ($C014)
- `RDCXROM` – the RDCXROM softswitch ($C015)

- `RDALTZP` – the RDALTZP softswitch ($C016)
- `SHADOW` – the Shadow Register ($C035)
- `STATEREG` – the State Register ($C068)
- `bytes` – a raw byte array containing the CPU opcode and operand.
- `cycles` – the number of 65816 cycles the instruction takes to issue – irrespective of clock speeds.
- `opcode` – a string containing the opcode name.
- `operand` – a string containing the operand, if any, for the instruction.
- `instruction` – a string representing the full instruction (opcode + operand).
- `horizCounter` – the horizontal video counter. This is the same as the 7-bit value held in the `$C02F` softswitch.
- `vertCounter` – the vertical video counter. This is the same as the 9-bit value held in the `$C02E` and `$C02F` softswitches.
- `timestamp` – time since the machine started (measured in 1 MHz cycles)
- `timestampFixed` – same as timestamp but padded to an output width of 17 characters if necessary

In addition to these fields there is a useful, and performant, shortcut for building a custom string from the fields:

- `formatString(string)` – this will generate a string with any embedded keywords replaced with their actual values (in hexadecimal). Embedded keywords are specified within braces.

An example of `formatString` would be:

```
History[i].formatString("{K}/{PC} - A={A}, X={X}, Y={Y}"
```

Putting this all together, here is a simple script that will display the last 20 instructions that were executed. It will output the result to the Script Log. Note that the for loop iterates from 19 down to 0, rather than the other way around. This is because we want this script to display the oldest instruction first and the newest instruction last.

```
for (i in 19..0) {
  System.print(History[i].formatString(
    "{K}/{PC} - A={A}, X={X}, Y={Y} - {instruction}"))
}
```

Handling Events

If your script contains an `EventHandler` class then Crossrunner will callback into your class when certain events happen. A skeleton for handling different events is shown below:

```
class EventHandler {
  static onBreakpoint() {
  }

  static onFrameComplete() {
  }
}
```

`onBreakpoint` will be called whenever a breakpoint is triggered in the emulator.

`onFrameComplete` will be called 60 times a second, at the completion of each video frame. A frame is complete when all visible scanlines have been displayed – this includes the scanlines in the border region.

Adding overlays on top of the display area

The `Overlay` class can be used to overlay rectangles and text over the screen area. The uses for this are limitless – such as exposing internal game variables for all to see, highlighting hit boxes for characters in a game, or displaying the number of frames per second that a game engine is outputting.

This class has the following methods:

- `drawRectangle(x, y, width, height, color)` – draws a rectangle at the specified coordinates with the specified width/height. The outline of the rectangle is drawn using the specified colour.
- `fillRectangle(x, y, width, height, color)` – this is similar to drawRectangle, but this method instead fills the rectangle with the specified colour.
- `drawLine(startX, startY, endX, endY, color)` – draws a line from the specified starting point to the specified end point using the specified colour.
- `drawText(x, y, color, text)` – draws a string of text using the specified colour at the given coordinates.

When specifying color, you can either use one of the predefined names (`Color.Black`, `Color.White`, `Color.Red`, `Color.Green`, `Color.Blue`, `Color.Yellow`, `Color.Magenta`, `Color.Aqua`, `Color.Lime`, `Color.Orange`) or you can specify a custom colour using `Color.new(r, g, b, a)` where you specify the red, green, blue and alpha channel values using a number between 0 and 1. For example, `Color.new(1,0.65,0,1)` will generate the same shade of orange as the built-in `Color.Orange`.

The `Overlay` class also has the following field:

- `coordinateSystem` – this field is by default `Coordinates.Native`, which means that any coordinates specified for drawing rectangles and lines are specified using the host system's resolution and are relating to the top-left corner of the pane.

  You can set this field to, for example, `Coordinates.Apple280` which allows you to specify coordinates where 0, 0 will be the top left of the Hires screen and 279, 191 will be the bottom left of the Hires screen. Other options include `Coordinates.Apple140`, `Coordinates.Apple320`, `Coordinates.Apple560` and `Coordinates.Apple640`.

The example below adds an overlay over the game Rescue Raiders, showing internal game variables for the computer AI.

```
class EventHandler {
  static onFrameComplete() {
    var fuel = Memory[0x6108]
    var missiles = Memory[0x6102]
    Overlay.fillRectangle(10,10,200,60,Color.Red)
    Overlay.drawText(15,15,Color.Black,"Enemy fuel: %(fuel)")
    Overlay.drawText(15,30,Color.Black,"Enemy missiles: %(missiles)")
  }
}
```

The series *Behind the Code* on the YouTube channel *Displaced Gamers* shows many examples of this type of emulator scripting technique for a variety of popular Nintendo games.

Reading and Writing to disk

Opening a file and then either creating or writing to it involves the use of two classes. Firstly, the `Filesystem` class is used to identify the filesystem which will be used – which may be your computer's native filesystem, or it may be a ProDOS filesystem within a disk image.

The constructor for filesystem requires a pathname – which either identifies a folder on your hard drive which becomes the base path for all file actions, or specifies the full pathname of a disk image file. For example, the following code specifies that the native filesystem should be used:

```
var fs = Filesystem.new("C:\\myfolder")
```

Whereas the following code specifies that a disk image should be used:

```
var fs = Filesystem.new("C:\\myimage.2mg")
```

The `Filesystem` class has the following methods:

- `create(path, filetype, auxtype)` – this will create a new file on the filesystem using the specified relative pathname and with the given ProDOS filetype and auxtype. When using the native filesystem the filetype and auxtype are encoded using the Ciderpress convention of #FFAAAA.
- `createFolder(path)` – this will create a new folder on the filesystem using the specified relative pathname.
- `delete(path)` – this will delete the file at the specified location.
- `rename(path, newFilename)` – this will rename the file at the specified location to the new filename that was specified.
- `getFileInfo(path, fileInfo)` – this will return information about the specified file into the specified `FileInfo` instance.
- `setFileInfo(path, fileInfo)` – this will update the file's attributes to match the attributes contained in the `FileInfo` instance.

The `Filesystem` class also has the following fields:

- `blocksFree` – returns the number of free blocks on the filesystem.
- `blocksUsed` – returns the number of blocks that are in use on the filesystem.
- `totalBlocks` – returns the total number of blocks on the filesystem.
- `volumeName` – returns the ProDOS volume name of the filesystem.

The `FileInfo` class has the following fields:

- `accessType` – the access permissions for the file. This is either `AccessType.NoAccess`, `AccessType.ReadOnly`, `AccessType.WriteOnly` or `AccessType.ReadAndWrite`.
- `isVisible` – true if the file is visible.
- `backupNeeded` – true if the backup needed flag is set for the file.
- `renameAllowed` – true if the file is permitted to be renamed.
- `destroyAllowed` – true if the file is permitted to be deleted.
- `fileType` – the ProDOS filetype of the file.
- `auxType` – the ProDOS auxtype of the file.
- `storageType` – returns the storage type of the file. This is either `StorageType.File`, `StorageType.ExtendedFile` or `StorageFile.Directory`.
- `endOfFile` –the end-of-file marker for the file, or for the data fork within the file.
- `blocksUsed` – returns the number of blocks used for the file, or for the data fork within the file.
- `resourceEndOfFile` – the end-of-file marker for the resource fork within the file.
- `resourceBlocksUsed` – returns the number of blocks used for the resource fork within the file.

There are a number of built-in filetype enumerations that can be used when creating files, and comparing the filetype returned by `getFileInfo` against. These are `Filetype.TXT`, `Filetype.BIN`, `Filetype.SRC`, `Filetype.S16`, `Filetype.PNT`, `Filetype.PIC`, `Filetype.ANI`, `Filetype.SND` and `Filetype.SYS`.

Once an instance of the `Filesystem` class has been created and, if writing, an empty file has been created on that filesystem we can now construct an instance of the `File` class. The constructor takes the filesystem as the first parameter, and the pathname within that filesystem as the second parameter. For example:

```
var f = File.new(fs, "output.log")
```

If `fs` in the example above uses the native filesystem with a base path of "C:\myfolder" then the file referenced will have a path of "C:\myfolder\output.log". However, if `fs` in the example above uses a disk image then the file referenced will be in the root directory of that disk image.

If you wish to open the resource fork of a file then you need to use the constructor that accepts the fork type as the third parameter. For example:

```
var f = File.new(fs, ":MYFOLDER:myfile", Fork.Resource)
```

Pathnames must be specified using the GS/OS convention of colon separators.

The `File` class has the following methods:

- `read(numberOfBytes)` – this will attempt to read the specified number of bytes. It will return a byte array containing the data that was read.
- `write(data)` – this will attempt to write the specified data to the file.
- `getDirectoryEntry(offset, directoryEntry)` – this will return information about the directory entry contained at the specified offset. This will only work if the file that was opened was a directory. This method will return true if there is an entry at the specified offset, and false when it reaches the end of the directory.

The `DirectoryEntry` class has the same fields as the `FileInfo` class, plus two additional fields:

- `filename` – returns the filename for the directory entry.
- `entryOffset` – returns the offset of the filename within the directory.

Saving and restoring snapshots

The `Snapshot` class can be used to save a snapshot of the current emulator state to disk, and to also restore the emulator state from disk. The class has the following methods:

- `load(file)` – this will load the snapshot from the specified file. The file can be either an instance of the `File` class, or it can be a `String` containing the pathname of the snapshot on the native filesystem.
- `save(file)` – this will save a snapshot of the emulator state to specified file. The file can be either an instance of the `File` class, or it can be a `String` containing the pathname of the snapshot on the native filesystem.

Examples

Logging every instruction to a file on disk

The following script installs an event handler that will log every CPU instruction executed to a file on disk:

```
class Logger {
  construct new() {
    _fs = Filesystem.new("C:\\")
    _fs.create("output.log",Filetype.TXT,0)
    _f = File.new(_fs, "output.log")
  }

  logHistory() {
    var end = History.size - 1
    if (History.markedEntry != -1) {
      end = History.markedEntry - 1
    }
    for (i in end..0) {
      _f.write(History[i].formatString(
         "{K}/{PC} - A={A}, X={X}, Y={Y} - {instruction}\n"))
    }
    History.markedEntry = 0
  }
}

class EventHandler {
  static init() {
    __logger = Logger.new()
  }

  static onFrameComplete() {
    __logger.logHistory()
  }

  static onBreakpoint() {
  }
}

EventHandler.init()
```

Listing the contents of a ProDOS disk image

The following script will open a disk image and list the filenames contained in the root directory:

```
var fs = Filesystem.new("C:\\Emulation\\emutest.po")
var f = File.new(fs, ":%(fs.volumeName):")
var di = DirectoryEntry.new()
var offset = 0
while (f.getDirectoryEntry(offset,di)) {
  System.print(di.filename)
  offset = offset + 1
}
```

Printing out Applesoft statements as they run

The following script will print out Applesoft statements as they are executed. Note that it requires a breakpoint to be set at address $D7D2 in order to work.

```
class EventHandler {
  static init() {
    __tokens = [
      "END",      "FOR",     "NEXT",    "DATA",    "INPUT",   "DEL",
      "DIM",      "READ",    "GR",      "TEXT",    "PR#",     "IN#",
      "CALL",     "PLOT",    "HLIN",    "VLIN",    "HGR2",    "HGR",
      "HCOLOR=",  "HPLOT",   "DRAW",    "XDRAW",   "HTAB",    "HOME",
      "ROT=",     "SCALE=",  "SHLOAD",  "TRACE",   "NOTRACE", "NORMAL",
      "INVERSE",  "FLASH",   "COLOR=",  "POP",     "VTAB",    "HIMEM:",
      "LOMEM:",   "ONERR",   "RESUME",  "RECALL",  "STORE",   "SPEED=",
      "LET",      "GOTO",    "RUN",     "IF",      "RESTORE", "&",
      "GOSUB",    "RETURN",  "REM",     "STOP",    "ON",      "WAIT",
      "LOAD",     "SAVE",    "DEF",     "POKE",    "PRINT",   "CONT",
      "LIST",     "CLEAR",   "GET",     "NEW",     "TAB",     "TO",
      "FN",       "SPC(",    "THEN",    "AT",      "NOT",     "STEP",
      "+",        "-",       "*",       "/",       "^",       "AND",
      "OR",       ">",       "=",       "<",       "SGN",     "INT",
      "ABS",      "USR",     "FRE",     "SCRN",    "PDL",     "POS",
      "SQR",      "RND",     "LOG",     "EXP",     "COS",     "SIN",
      "TAN",      "ATN",     "PEEK",    "LEN",     "STR$",    "VAL",
      "ASC",      "CHR$",    "LEFT$",   "RIGHT$",  "MID$"]
  }

  static onFrameComplete() {
  }

  static buildLine(i) {
    var line = ""
    var inQuote = false
    while (Memory[i] != 0) {
      var b = Memory[i]
      if ((b == 0x3A) && !inQuote) break
      if (b == 0x22) inQuote = !inQuote
      if (b < 0x80) {
        line = line + String.fromByte(b)
      } else {
        line = line + " " + __tokens[b-0x80] + " "
      }
      i = i + 1
      if (i == 255) return "error"
    }
    return line
  }

  static onBreakpoint() {
    var address = Memory[0xB8] + (Memory[0xB9] << 8)
    if (Memory[address] == 0x3A) {
      var line = "   %(buildLine(address + 1))"
      System.print("%(Hex.Str16(address)) - %(line)")
    } else {
      var lineNumber = Memory[address + 3] + (Memory[address + 4] << 8)
      var line = "%(lineNumber) %(buildLine(address + 5))"
      System.print("%(Hex.Str16(address)) - %(line)")
    }
    Emulator.run()
  }
}

EventHandler.init()
```

Chapter 7
Command Line arguments

Crossrunner optionally supports command line arguments. If no command line arguments are specified then Crossrunner will start in Library mode – which is the mode where it presents you with a user interface to select a System or Disk Image to boot. If command line arguments are specified then their usage is as follows:

```
Crossrunner.exe [-ROM01 | -ROM03 | -Snapshot <filename>]
      [-Speed=<speed>] [-Debug | -DebugSource] [-CompatibilityLayer]
      [-Source <filename 1> … <filename n>]
      [-Map <filename>]
      <filename>
```

The switches have the following behaviour:

`-ROM01`	Emulate a ROM01 machine
`-ROM03`	Emulate a ROM03 machine
`-Snapshot <filename>`	Initialise the starting machine state from the specified file. The ROM01/ROM03 switches are ignored if a snapshot is specified.
`-Speed=<speed>`	Set the machine speed – valid options are: • `Slow` – 1 MHz operation • `Normal` – 2.8 MHz operation • `Fast` – 7 MHz operation • `VeryFast` – 14 MHz operation • `Unlimited` – Uncapped operation
`-Debug`	Crossrunner will startup in low-level (assembly language) Debug mode in a Stopped state. This switch cannot be combined with -DebugSource.
`-DebugSource`	Crossrunner will startup in high-level (C/Pascal) Debug mode and only run until the first high-level line of code is encountered. This switch cannot be combined with -Debug.
`-CompatibilityLayer`	Crossrunner will operate in compatibility layer mode. In this mode, GS/OS calls are redirected to the host's filesystem and Toolbox calls are run natively instead of being emulated from ROM.
`-Source <filenames>`	Specify one or more filenames of source code (C or Pascal) or assembler output (ORCA/M or Merlin) to be debugged. See Chapter 5 for more information.
`-Map <filename>`	Specify the linker map output (ORCA) or link file (Merlin) to be debugged. See Chapter 5 for more information.
`filename`	The program to run. • This can be a disk image (.dsk, .po, .do, .nib, .hdv, .2mg or .woz) that will be booted from. • This can be a GS/OS OMF file that will be executed.

Thanks

Thanks to John Brooks. Nobody understands Apple IIGS video on a cycle-by-cycle basis better than John and Doug Snyder. John patiently answered my many questions, and saved me countless hours.

Thanks to Alex Lee. Alex did the Apple IIGS system image and the drive icons at the bottom of the screen. He has collated and clean up a huge library of Apple II and IIGS box art which he has kindly shared with this project.

Thanks to 4am and qkumba. They have done, and continue to do, an amazing amount of preservation and clean cracking of Apple II software. The Crossrunner library is built upon their hard work.

Thanks to Larry Greene. He is the person who rewound time to turn the Mark Twain source code into the ROM 03 source code, which can be used within Crossrunner.

Thanks to Kent Dickey. He was one of the blazing pioneers in the Apple IIGS emulation space. After development of KEGS stopped for several years, it gave me the incentive to create Crossrunner. Now that KEGS is in active development again, Apple II enthusiasts have more options for quality emulators than ever before.

Thanks to all the beta testers. This project has been in development, and in beta, for far longer than I could ever have imagined at the start.

Third Party Libraries

Pugixml (https://github.com/zeux/pugixml)

Miniaudio (https://github.com/mackron/miniaudio)

SQLite (https://www.sqlite.org/)

All of the code and documentation in SQLite has been dedicated to the public domain by the authors.

Cathode-Retro (http://github.com/DeadlyRedCube/Cathode-Retro)