

Claude 3.7 Sonnet

Please code a Prodos interrupt handler in 6502 assembly language that can be compiled by EDASM

I'll create a ProDOS interrupt handler in 6502 assembly language that can be compiled by EDASM.

prodos-interrupt-handler.txt

This ProDOS interrupt handler is written for the 6502 processor and is compatible with EDASM (Apple's Editor/Assembler). Here's a breakdown of what it does:

1. Handler Structure:

- Saves all processor registers (A, X, Y) and processor status
- Preserves zero page locations that might be used
- Calls your custom interrupt code
- Restores all saved state
- Returns from interrupt with RTI

2. Key Components:

- The main handler at `INTHAND`
- A placeholder for your custom code at `USERINT`
- Installation routine at `INSTALL` that hooks into the system
- Uninstallation routine at `UNINSTL` to restore the original handler

3. Usage:

- Replace the `USERINT` routine with your actual interrupt processing code
- Call `INSTALL` once to hook your handler into the system
- Call `UNINSTL` when you want to remove your handler

The code uses standard 6502 assembly conventions and ProDOS system entry points. You may need to adjust the `ORG` directive to place the code in an appropriate memory location for your specific Apple II configuration.

Please use the Prodos MLI to allocate and deallocate the interrupt, and follow the rules for the handling decimal flag and carry under Prodos

I'll modify the interrupt handler to properly use the ProDOS MLI for allocation and deallocation of the interrupt, and ensure it correctly handles the decimal flag and carry according to ProDOS requirements.

prodos-interrupt-handler (1).txt

I've revised the interrupt handler to properly follow ProDOS conventions and use the MLI (Machine Language Interface) for interrupt allocation and deallocation. Here are the key improvements:

ProDOS MLI Integration:

Now uses the proper ALLOC_INT (\$40) and DEALLOC_INT (\$41) MLI calls

Creates appropriate parameter blocks for these calls

Handles potential errors during installation/uninstallation

ProDOS Requirements:

Added CLD instruction to clear the decimal flag as required by ProDOS

Added CLC before exiting to indicate the interrupt was handled

Properly preserves and restores processor status

System Integration:

Saves and restores the system BRK vector

Uses interrupt number 0 for system interrupt handling

Properly structures the parameter blocks according to MLI requirements

Error Handling:

Checks for errors after MLI calls using the carry flag

Provides entry points for error handling routines

This implementation follows the official ProDOS rules for interrupt handlers, including:

Clearing the decimal mode flag

Clearing the carry flag before returning

Using the MLI to properly allocate and deallocate interrupts instead of directly modifying vectors

Preserving all registers and processor status

To use this handler, you would:

Replace the code in the USERINT routine with your interrupt processing logic

Call INSTALL to register the handler with ProDOS

Call UNINSTL when you want to remove your handler

What does an Apple 6502 assembler developer need to know about using the 80 column screen in a program that has interrupts?

When developing Apple II 6502 assembly programs that use both the 80-column screen and interrupts, there are several critical considerations:

1. **Hardware Page Switching Complications**
 - The 80-column card uses auxiliary memory and soft switches that can be affected during interrupts
 - The 80-column firmware routines are not all interrupt-safe because they might be interrupted mid-operation
2. **Critical Timing Issues**
 - Screen accesses have precise timing requirements
 - Interrupts can occur at any time, potentially disrupting screen update timing
 - The video circuitry continues displaying even during interrupt handling
3. **Screen Firmware State Preservation**
 - The 80-column firmware maintains internal state (cursor position, display mode, etc.)
 - Interrupts must save and restore this state if they use screen functions
4. **Safe Practices for Interrupt Handlers:**
 - Save all soft-switch states at the beginning of your interrupt handler
 - If accessing the screen during an interrupt, use only ROM routines known to be interrupt-safe
 - Always restore the original memory configuration before returning from the interrupt
 - Use indirect addressing for screen access to ensure proper memory bank selection
5. **Memory Bank Considerations:**
 - The 80-column display uses both main and auxiliary memory for text storage
 - Interrupts can occur when either bank is selected
 - Your interrupt handler must identify and preserve the active memory bank
 - Use the RDMAINRAM (C
 - 002
 -)
 - ,
 - R
 - D
 - C
 - A
 - R
 - D
 - R
 - A

M

(

C002), RDCARDRAM (

C002),RDCARDRAM(C003), WRMAINRAM (C

004

)

,

a

n

d

W

R

C

A

R

D

R

A

M

(

C004), and WRCARDRAM (

C004),andWRCARDRAM(C005) soft switches carefully

6. **Safe ROM Entry Points:**

- Use COUT (\$FDED) for character output (it's generally interrupt-safe)
- Avoid direct calls to 80-column firmware routines during interrupts
- The Monitor ROM routines are generally safer than direct 80-column card access

7. **Protecting Against Corruption:**

- Create a flag to indicate when screen operations cannot be interrupted
- In critical sections, temporarily disable interrupts with SEI/CLI
- Keep interrupt handlers short to minimize timing disruptions

8. **Memory-Mapped Screen Access:**

- Be very careful with direct memory-mapped screen access during interrupts
- Screen memory in 80-column mode spans across both main and auxiliary memory
- Characters in even columns reside in main memory (\$400-\$7FF)
- Characters in odd columns reside in auxiliary memory (\$400-\$7FF)

9. **Alternate Solutions:**

- Consider double buffering techniques where your interrupt routines write to an off-screen buffer
- Update the actual screen only during vertical blanking periods
- Use a flag system to defer screen updates until safe

10. **ProDOS Specific Requirements:**

- ProDOS itself uses interrupts for timing
- Always follow ProDOS interrupt handler requirements (clearing decimal mode, preserving registers)
- Be aware that ProDOS may switch memory banks itself during system calls

These considerations are essential because improper handling can lead to screen corruption, system crashes, or intermittent bugs that are difficult to track down.

Please write a ProDOS compliant interrupt handler in EDASM format that runs on an Apple 2GS and reads activity from the ADB connected mouse and calculates X and Y coordinates for the mouse position clamped to the range 0-79 for X and 0-23 for Y. Store the X coordinate in zero page location 06 and the Y coordinate in zero page location 07.

I'll create a ProDOS-compliant interrupt handler for the Apple IIGS that tracks mouse position via the ADB (Apple Desktop Bus) and stores the coordinates in zero page.

mouse-tracking-handler.txt

This interrupt handler is specifically designed for the Apple IIGS to track mouse position via the Apple Desktop Bus (ADB) and store the coordinates in the zero page locations you requested. Here's a breakdown of the key features:

Key Features

ProDOS Compliance:

Uses proper MLI calls for interrupt allocation/deallocation

Clears decimal mode flag (CLD) as required

Clears carry flag (CLC) before returning

Preserves all registers and processor status

IIGS-Specific Handling:

Detects and handles 65816 native/emulation mode

Uses IIGS-specific ADB registers and protocols

Handles the 16-bit environment properly

Mouse Coordinate Processing:

Reads raw mouse movement data from ADB

Tracks accumulated movement in temporary variables

Clamps final X coordinate to 0-79 range (stored in \$06)

Clamps final Y coordinate to 0-23 range (stored in \$07)

Installation/Removal:

INSTALL routine hooks the handler into ProDOS

UNINSTALL routine properly removes the handler

Initializes mouse tracking variables

Usage Instructions

Assembly: Compile this code with EDASM.

Installation: Call the INSTALL routine once to activate mouse tracking.

Access: Your main program can read the current mouse position from:

X position (0-79): Zero page location \$06

Y position (0-23): Zero page location \$07

Removal: Call UNINSTALL when you no longer need mouse tracking.

Technical Notes

The handler reads the ADB bus for mouse data and processes the relative movement

It accumulates movement in temporary variables and applies clamping

It properly handles the IIGS environment register for native/emulation mode switching

The coordinate system maps to an 80×24 text screen with (0,0) at the upper left

This implementation should work reliably under ProDOS on the Apple IIGS while maintaining compatibility with other interrupt-driven processes.

Retry

Claude can make mistakes.

Please double-check responses.