



Welcome,
Guest. Please
login or register.
Did you miss
your activation
email?
March 17, 2019,
10:26:39 AM



Famicom World
 Family Computer
 Famicom / Disk System (Moderators: manuel,
L E T)
 Famicom Disk System Development

0 Members and 1 Guest are viewing
this topic.

« previous
next »

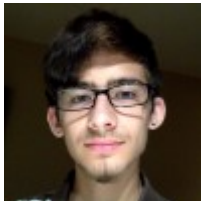
Pages: [1] 2 3 4



Author **Topic: Famicom Disk System Development (Read 10672 times)**

aguerrero810

Famicom
☆☆☆☆
Gender:
 United States
Posts: 32



Famicom Disk System Development
« on: September 25, 2015, 01:36:29 AM »

What do I need to start developing games for the Famicom Disk System. I have minor knowledge on making programs for the NES. Is it similar? Are there build tools i should be aware of? I have FDSLOADR for testing and development and I would like to eventually write programs to disks to play them on original hardware.

Thanks!
Anthony



Logged

P
FamicomBox
☆☆☆☆
Gender:
 Sweden
Posts: 3296

Re: Famicom Disk System Development
« Reply #1 on: September 25, 2015, 06:00:45 PM »

As long as you don't deal with loading files or changing disk side, it's pretty much the same except you need file headers for each file. You have three NMIs you can use. More details here:

http://wiki.nesdev.com/w/index.php/Family_Computer_Disk_System.

Here is an example I made which loads without errors but I couldn't get my program to load for some reason:

Edit: Got it to work at last: <http://www.famicomworld.com/forum/index.php?topic=12209.msg166113#msg166113>

Code:

```
;Famicom Disk System Program Template v0.7
;By Pokun
;
;Assembles in ASM6

;FDS Disk Image

;Games are stored on one or multiple disk sides (every disk has 2 sides).
;The FDS BIOS is used to load data from disks to PRG RAM or VRAM,
;and games can execute from there.
```

```

=====
;
; Constants
=====
;Header constants
REVISION = $00 ;revision number used in the header (starts at 0)
SIDE_COUNT = $01 ;number of disk sides (1, 2 or 4)
FILE_COUNT1 = $02 ;total number of non-hidden files on disk side 1
FILE_COUNT2 = $00 ;total number of non-hidden files on disk side 2
FILE_COUNT3 = $00 ;total number of non-hidden files on disk side 3
FILE_COUNT4 = $00 ;total number of non-hidden files on disk side 4

;FDS defines
FDS_CRAM = $0000 ;where CHR-RAM starts
FDS_PRAM = $6000 ;where PRG-RAM starts
FDS_BIOS = $E000 ;where the BIOS resides

;FDS BIOS calls
FDS_Delay132 = $E149 ;132 clock cycle delay

=====
;
; Variables
=====
;Zero Page ($0000-$00FF)
.enum $0000

.ende
;Page 1 ($0100-$01FF)
;(Stack)

;Page 2 ($0200-$02FF)
;(Shadow OAM)

;Page 3-7 ($0300-$07FF)
.enum $0300

.ende

=====
; fwNES FDS header
=====
;16 byte header for the .FDS Quick Disk format
;This header is optional as newer emulators may not require
;it. Number of disk sides can be figured out by looking at the
;FDS image file size.

.org $0000
; .db "FDS", $1A ;identification of the fwNES header
; .db SIDE_COUNT ;number of disk sides
; .pad 16, $00 ;clear the remaining bytes
;HEADEROFFSET = $ ;add this to compensate for the added header.

;Note about blocks
;Each disk side must be structured into blocks as follows:
;First a disk header block and a file amount block then one file header block
;and file data block for each file on the disk. Finally the disk sides ends
;with an end of disk side indicator. Each block has a block code:
;$01 = Disk Header Block
;$02 = File Amount Block
;$03 = File Header Block
;$04 = File Data Block

;$FF = End of Disk Side Indicator

;From the last file, fill the side with zeroes so that the disk side gets
;exactly 65500 bytes.

;On real disks there are also gaps and CRCs not present in
;FDS images. These must also be considered when calculating true
;disk capacity. The actual disc capacity in bytes is:
;65500 - 28300/8 - (2*N + 1)*(16 + 976)/8
;N = number of files
=====

```

; Disk Side 1 (65500 byte)

;=====

;Disk Header Block

.db \$01 ;Block code (1 = disk header block)
;db ""NINTENDO-HVC"" ;Used by BIOS to verify legitimate image.
;If the string is wrong the disk will fail to boot.
.db \$00 ;Maker code (similar codes as in Game Boy headers)
;\$00 is unlicensed, \$01 is Nintendo \$08 is Capcom...
.db "HMB" ;3-letter ASCII code for title (e.g. LNK for Zelda no Densetsu)
.db " " ;Game type
;\$20 = " " — Normal disk
;\$45 = "E" — Event (e.g. Japanese national DiskFax tournaments)
;\$52 = "R" — Reduction in price via advertising
.db REVISION ;Revision number (starts at \$00, increments per revision)
.db \$00 ;Side number
;\$00 = Side A, \$01 = Side B. Single-sided disks use \$00
.db \$00 ;Disk number (first disk is \$00, second is \$01, etc.)
.db \$00 ;Disk type
;\$00 = FMC ("normal card"), \$01 = FSC ("card with shutter")
.db \$00 ;Unknown
.db \$0F ;Boot read file code (file code/number to load upon boot)
.db \$FF,\$FF,\$FF,\$FF,\$FF ;Unknown
.db \$00,\$00,\$00 ;Manufacturing date
;Stored in BCD, subtract the Shouwa starting year 1925 from the year
;Example: 1988-11-28 becomes \$63 \$11 \$28
.db \$49 ;Country code (\$49 = Japan)
.db \$61 ;Unknown
.db \$00 ;Unknown
.db \$00,\$02 ;Unknown
.db \$00,\$00,\$00,\$00,\$00 ;Unknown
.db \$00,\$00,\$00 ;"Rewritten disk" date (supposedly by the Disk Writer)
.db \$00 ;Unknown
.db \$80 ;Unknown
.db \$00,\$00 ;Disk Writer serial number
.db \$07 ;Unknown
.db \$00 ;Disk rewrite count (value in BCD, \$00 = Original (no copies))
.db \$00 ;Actual disk side (\$00 = Side A, \$01 = Side B)
.db \$00 ;Unknown
.db \$00 ;Price
;db \$00,\$00 ;CRC - Not used in the .fds file format.

;Disk sides note

;If the FDS is started with a disk whose side number and disk number
;aren't both \$00, it will be prompted to insert the first disk side.
;However, some games make this number \$00, even for the second disk
;to make it bootable too.

;File Amount block

.db \$02 ;Block code (2 = file amount block)
;db FILE_COUNT1 ;Total number of files recorded on disk (side?)
;If more files exist on disk they will be considered hidden and the
;BIOS will ignore them. A loading routine on disk is required to
;load them.

;File codes note

;All files with IDs smaller or equals to the boot read file code
;will be loaded when the game is booting.

;-----

;File "KYODAKU-"

;The first file on the disk must be the "KYODAKU-" file (approval in Japanese)
;that contains the message that scrolls up on screen ("THIS PRODUCT
;IS MANUFACTURED AND SOLD BY NINTENDO...") at boot, and must match
;the same data in the BIOS at \$ED37. It's a nametable type of file that is
;loaded into VRAM address \$2800.

;File Header Block

.db \$03 ;Block code (3 = file header block)
.db \$00 ;File Number
;must go in increasing order, first file is 0
.db \$00 ;File ID

```

;ID specified at disk-read function call
;This is the number which will decide which file is
;loaded from the disk (instead of the file number).
;An ID smaller than the boot number means the file is
;a boot file, and will be loaded on first start up.
.db "KYODAKU-" ;File Name (8 uppercase ASCII characters)
.db $00,$28 ;File Address (16-bit little endian)
;the destination address when loading
.db $E0,$00 ;File Size (16-bit little endian)
.db $02 ;File Type
;0:Program (PRAM)
;1:Character (CRAM)
;2:Nametable (VRAM)

;File Data Block
.db $04 ;Block code
.include "KYODAKU-.650" ;file data
;If this file doesn't match the 224-byte text approval data in the BIOS the
;disk won't run.

;-----
;File "MAINGAME"

;File Header Block
.db $03 ;Block code
.db $01 ;File Number
.db $01 ;File ID
.db "MAINGAME" ;File Name (8 uppercase ASCII characters)
.dw FDS_PRAM ;Where to load file to (16-bit little endian)
.dw (MainGameEnd - MainGameStart) ;File Size (16-bit little endian)
.db $00 ;File type
;0:Program (PRAM)
;1:Character (CRAM)
;2:Nametable (VRAM)

;File Data Block
.db $04 ;Block code
MainGameStart:
.include "MAINGAME.650" ;file data
MainGameEnd:
;This file contains the main program and data.

;-----
;End of Disk Side 1
.db $FF ;End of disk side indicator
.pad 65500, $00 ;zero out rest of disk

;-----
;Next disk side goes here. A header for this side is needed but no KYODAKU-file
;unless it's bootable I think.

```

And the "KYODAKU-.650" file needs to contain this:

Code:

```

;"KYODAKU-" file

HEX 24 24 24 24 24 24 24 24 24 24 17 12 17 1D 0E
HEX 17 0D 18 24 28 24 24 24 24 24 24 24 24 24 24
HEX 24 24 24 24 24 24 24 0F 0A 16 12 15 22 24 0C 18
HEX 16 19 1E 1D 0E 1B 24 1D 16 24 24 24 24 24 24 24
HEX 24 24 24 24 24 24 24 24 24 24 24 24 24 24 24
HEX 24 24 24 24 24 24 24 24 24 24 24 24 24 24 24
HEX 24 24 1D 11 12 1C 24 19 1B 18 0D 1E 0C 1D 24 12
HEX 1C 24 16 0A 17 1E 0F 0A 0C 1D 1E 1B 0E 0D 24 24
HEX 24 24 0A 17 0D 24 1C 18 15 0D 24 0B 22 24 17 12
HEX 17 1D 0E 17 0D 18 24 0C 18 27 15 1D 0D 26 24 24
HEX 24 24 18 1B 24 0B 22 24 18 1D 11 0E 1B 24 0C 18
HEX 16 19 0A 17 22 24 1E 17 0D 0E 1B 24 24 24 24 24
HEX 24 24 15 12 0C 0E 17 1C 0E 24 18 0F 24 17 12 17
HEX 1D 0E 17 0D 18 24 0C 18 27 15 1D 0D 26 26 24 24

;Translates to this in SMB/Zelda encoding:

```

```

,"      NINTENDO R      "
,"      FAMILY COMPUTER TM      "
,"      "
," THIS PRODUCT IS MANUFACTURED "
," AND SOLD BY NINTENDO CO,LTD. "
," OR BY OTHER COMPANY UNDER  "
," LICENSE OF NINTENDO CO,LTD.. "

```

The "MAINGAME.650" file I haven't got to load yet so there's not much point in posting it. But it should look like something like this I believe:

Code:

```

;File "MAINGAME" data
;-----
NMI1:
jsr FDS_HOOK
rti

NMI2:
jsr FDS_HOOK
rti

NMI3:
jsr FDS_HOOK
;NMI code here
rti

;if you are not using all three NMIs you can have them share the same label
;-----
RESET:
;init code (although I believe the BIOS does most of the init at boot)
jsr FDS_HOOK
main:
;main program loop here
jmp main

;Subroutines:

FDS_HOOK:
;Prevent the program to suddenly reset to
;the FDS menu on NMI, IRQ, or RESET.
pha
lda #$C0
sta $0100 ;set NMI
lda #$80
sta $0101 ;set IRQ
lda #$35
sta $0102 ;set RESET vector
lda #$53
sta $0103 ;set boot type to "reset by user"
pla
rts

IRQ:
jsr FDS_HOOK
rti

;=====
; Interrupt Vector Table
;=====
.org $DFF6
.dw NMI1
.dw NMI2
.dw NMI3
.dw RESET
.dw IRQ
;-----

```

The FDS_HOOK I learned from Chris Covell. Just call it at the start of each interrupt handler.


Translation projects:


Kore ga Family Computer da!
Family Basic
Family Basic V3

aguerrero810

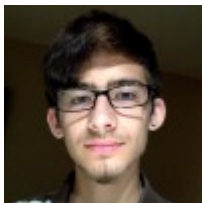
Famicom



Gender: 

 United States

Posts: 32



Re: Famicom Disk System Development

« Reply #2 on: September 27, 2015, 09:10:49 PM »


This is great! Thanks! Although I noticed that some pirate software like Disk Hacker II and Brad Taylors Graphics Demo, don't have the KYODAKU file, as they don't display that message. Is this file even necessary for booting?

P

FamicomBox



Gender: 

 Sweden

Posts: 3296



Re: Famicom Disk System Development

« Reply #3 on: September 28, 2015, 05:46:35 AM »

It's normally necessary. I have no idea how those programs tricks the BIOS. I think some Super Pig games also display another message there or skips it entirely so there's definitely a way around it.

Translation projects:

Kore ga Family Computer da!
Family Basic
Family Basic V3

famiac

Twin Famicom



Gender: 

 United States

Posts: 907



Re: Famicom Disk System Development

« Reply #4 on: September 28, 2015, 04:25:37 PM »

Oh interesting. Maybe i'll write for the fds for my next project.

member #844 and 565

aguerrero810

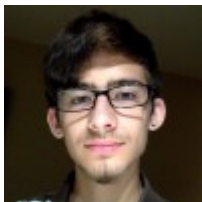
Famicom



Gender: 

 United States

Posts: 32



Re: Famicom Disk System Development

« Reply #5 on: October 03, 2015, 12:55:24 AM »

So trying your program files out.
I found that once the file loads it constantly uses
Code:

```
PC  OP  ARG
-----
$0000 JSR $0000
```

it just jumps to the same line it jumps from. no matter what i put in MAINGAME, that's all it does.


80sFREAK

Sharp C1



Re: Famicom Disk System Development

« Reply #6 on: October 03, 2015, 05:03:31 AM »

Gender: 
Posts: 1393

You forgot about IRQ and NMI

Hardware freak


 Logged


I don't buy, sell or trade at moment.

HVC-Man

AV Famicom



Gender: 

 Canada

Posts: 305



Re: Famicom Disk System Development

« Reply #7 on: October 03, 2015, 05:34:09 AM »

I was under the impression that FDS programming was basically like any other ASIC mapper, only significant difference is having to manually load and save data on the floppy disk, whereas on a cartridge game, addresses can easily be requested.


 Logged

P

FamicomBox



Gender: 

 Sweden

Posts: 3296



Re: Famicom Disk System Development

« Reply #8 on: October 03, 2015, 05:42:57 PM »

I'm under that impression too.

I tried adding a CHR data file and see if it loaded at boot and it did. So there's either something very wrong with my MAINGAME file or there something small in the header that isn't quite right. Maybe I misunderstood something about file IDs or something.

 Logged

Translation projects:

Kore ga Family Computer da!

Family Basic

Family Basic V3

80sFREAK

Sharp C1



Gender: 

Posts: 1393



Re: Famicom Disk System Development

« Reply #9 on: October 03, 2015, 05:57:27 PM »

address to start?

Hardware freak

 Logged


I don't buy, sell or trade at moment.

P

FamicomBox



Gender: 

 Sweden

Posts: 3296



Re: Famicom Disk System Development

« Reply #10 on: October 03, 2015, 06:48:48 PM »

What address? I start the image at .org \$0000 and the interrupt vector table is at \$DFF6. Is there something I missed?

The MAINGAME file is set to load to beginning of PRAM in the RAM Adapter.

 Logged

Translation projects:

Kore ga Family Computer da!

Family Basic

Family Basic V3

80sFREAK

Sharp C1



Gender: 

Posts: 1393



Re: Famicom Disk System Development

« Reply #11 on: October 03, 2015, 07:03:28 PM »

Quote from: P on October 03, 2015, 06:48:48 PM

Hardware freak



What address? I start the image at .org \$0000 and the interrupt vector table is at \$DFF6. Is there something I missed?

The MAINGAME file is set to load to beginning of PRAM in the RAM Adapter.

And PRAM is \$6000.

 Logged

I don't buy, sell or trade at moment.

P
FamicomBox
★★★★★
Gender: 
 Sweden
Posts: 3296



Re: Famicom Disk System Development


« Reply #12 on: October 03, 2015, 07:58:59 PM »

Yes it is. Do you mean the image file start address should start at \$6000 as well, using .org?

 Logged

Translation projects:

Kore ga Family Computer da!
Family Basic
Family Basic V3

80sFREAK
Sharp C1
★★★★★
Gender: 
Posts: 1393

Hardware freak



Re: Famicom Disk System Development



« Reply #13 on: October 04, 2015, 04:09:33 AM »

I read wiki page about FDS BIOS subroutines, but can't see clear howto start loaded file. If there is no info in the header(address to start), there might be two ways to start loaded programm.

1. Jump to the first loaded byte.
2. Use interrupt handler. NMI or IRQ.

 Logged

I don't buy, sell or trade at moment.

P
FamicomBox
★★★★★
Gender: 
 Sweden
Posts: 3296



Re: Famicom Disk System Development

« Reply #14 on: October 04, 2015, 09:18:16 AM »

How would I jump to the first loaded byte or use NMI/IRQ handlers? I can't jump since my code never starts in the first place. I tried adding a jmp RESET as a first thing in the MAINGAME file but it doesn't work.

I assume that the program starts executing where the reset vector at \$DFFC points at, but my program never seems to start. It gets stuck at a JSR \$0000 as Aguerro says.

 Logged

Translation projects:

Kore ga Family Computer da!
Family Basic
Family Basic V3

Pages: [1] 2 3 4



« previous next »

Jump to:

