Using breakpoints to find graphics data

by Bunkai (Jun 2023)

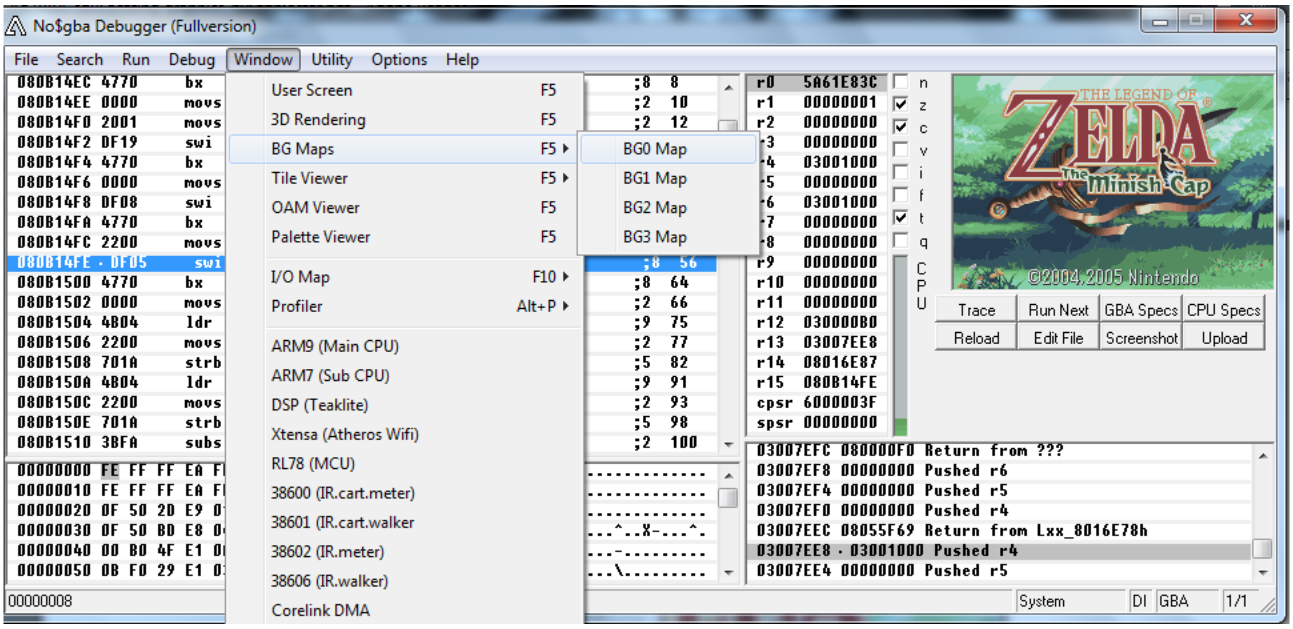
After reading the great tutorial by Labmaster from 24/03/06, which can be found at: https://www.romhacking.net/documents/361/ . I wanted to port it over to no$gba to use along other great documents like the one to find gba pointers using no$gba by Phonymike. While also adding a few quality life additions like screenshots of the full process to help visualize it.

To follow this tutorial, you will need the following:
no$gba: https://problemkaputt.de/gba.htm
yy-chr : https://www.romhacking.net/utilities/119/
Target ROM [The Legend of Zelda, The Minish Cap]. Checksum, crc32: ABCEBBB1

Since this is just a port over and my intention is to use this as a learning experience to do it later with mGBA, many of the explanations will be short like in the original (many will indeed be copy pasted), however i will try my best to add links and brief notes on some technical parts where (for me) Labmasters was too scarce. If you need to expand on, it’s encouraged to read the gba documentation at https://rust-console.github.io/gbatek-gbaonly/ . If you still have any question about the process you may ask them in the rhdn forum https://www.romhacking.net/forum/index.php or in the discord server, using the proper channels for that, but I won’t answer any private questions about your projects.

Our Target for this guide is the title screen logo from: The Legend of Zelda, The Minish Cap.

First step is to find where the graphics data is in VRAM, and the first place we look is on the various backgrounds. Open the BG Maps, and see if it's one of the Backgrounds.



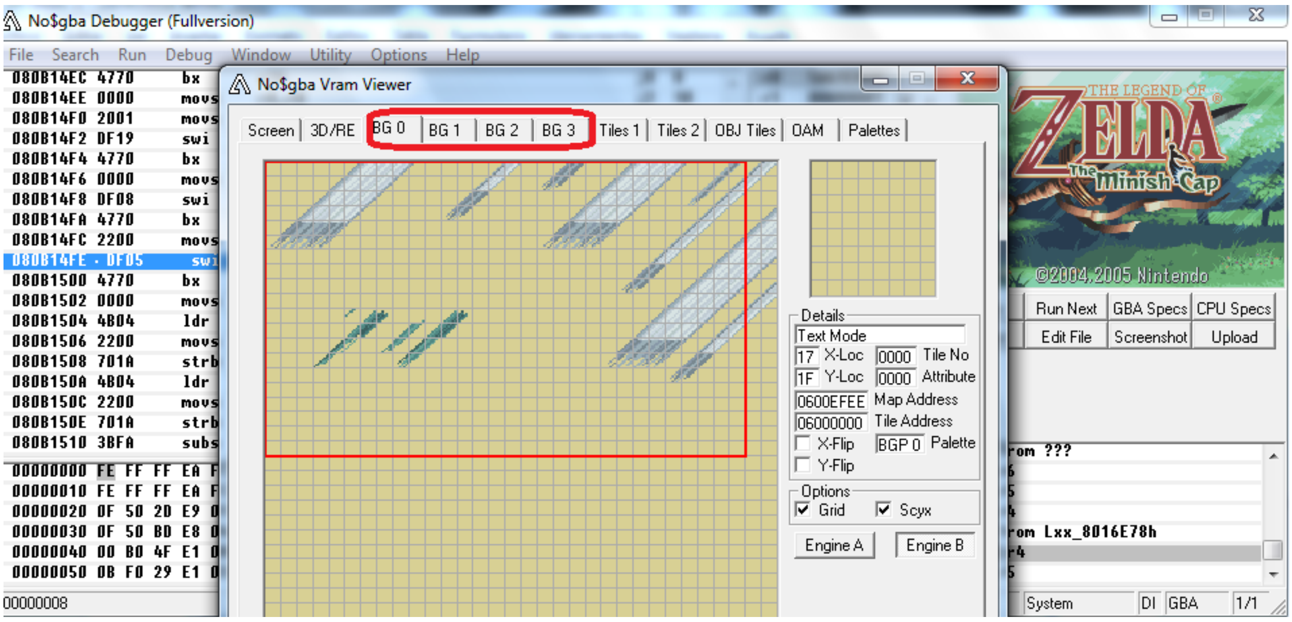
```
080B14EC 4770      bx
080B14EE 0000      movs
080B14F0 2001      movs
080B14F2 DF19      swi
080B14F4 4770      bx
080B14F6 0000      movs
080B14F8 DF08      swi
080B14FA 4770      bx
080B14FC 2200      movs
080B14FE · DF05     swi
080B1500 4770      bx
080B1502 0000      movs
080B1504 4B04      ldr
080B1506 2200      movs
080B1508 701A      strb
080B150A 4B04      ldr
080B150C 2200      movs
080B150E 701A      strb
080B1510 3BFA      subs
```

```
;8   8
;2   10
;2   12
;8   56
;8   64
;2   66
;9   75
;2   77
;5   82
;9   91
;2   93
;5   98
;2   100
```

```
r0   5A61E83C
r1   00000001
r2   00000000
r3   00000000
r4   03001000
r5   00000000
r6   03001000
r7   00000000
r8   00000000
r9   00000000
r10  00000000
r11  00000000
r12  030000B0
r13  03007EE8
r14  08016E87
r15  080B14FE
cpsr 6000003F
spsr 00000000
```

```
00000000 FE FF FF EA F   .............
00000010 FE FF FF EA F   .............
00000020 0F 50 2D E9 0   .............
00000030 0F 50 BD E8 0   ...^..X-...^.
00000040 00 B0 4F E1 0   ...-.........
00000050 0B F0 29 E1 0   ...\.........
00000008
```

```
03007EFC 080000F0 Return from ???
03007EF8 00000000 Pushed r6
03007EF4 00000000 Pushed r5
03007EF0 00000000 Pushed r4
03007EEC 08055F69 Return from Lxx_8016E78h
03007EE8 · 03001000 Pushed r4
03007EE4 00000000 Pushed r5
```



No$gba Debugger (Fullversion)
File Search Run Debug Window Utility Options Help
No$gba Vram Viewer
Screen 3D/RE BG 0 BG 1 BG 2 BG 3 Tiles 1 Tiles 2 OBJ Tiles OAM Palettes
Run Next GBA Specs CPU Specs
Edit File Screenshot Upload
Engine A Engine B

```
080B14EC 4770    bx
080B14EE 0000    movs
080B14F0 2001    movs
080B14F2 DF19    swi
080B14F4 4770    bx
080B14F6 0000    movs
080B14F8 DF08    swi
080B14FA 4770    bx
080B14FC 2200    movs
080B14FE · DF05   swi
080B1500 4770    bx
080B1502 0000    movs
080B1504 4B04    ldr
080B1506 2200    movs
080B1508 701A    strb
080B150A 4B04    ldr
080B150C 2200    movs
080B150E 701A    strb
080B1510 3BFA    subs
```

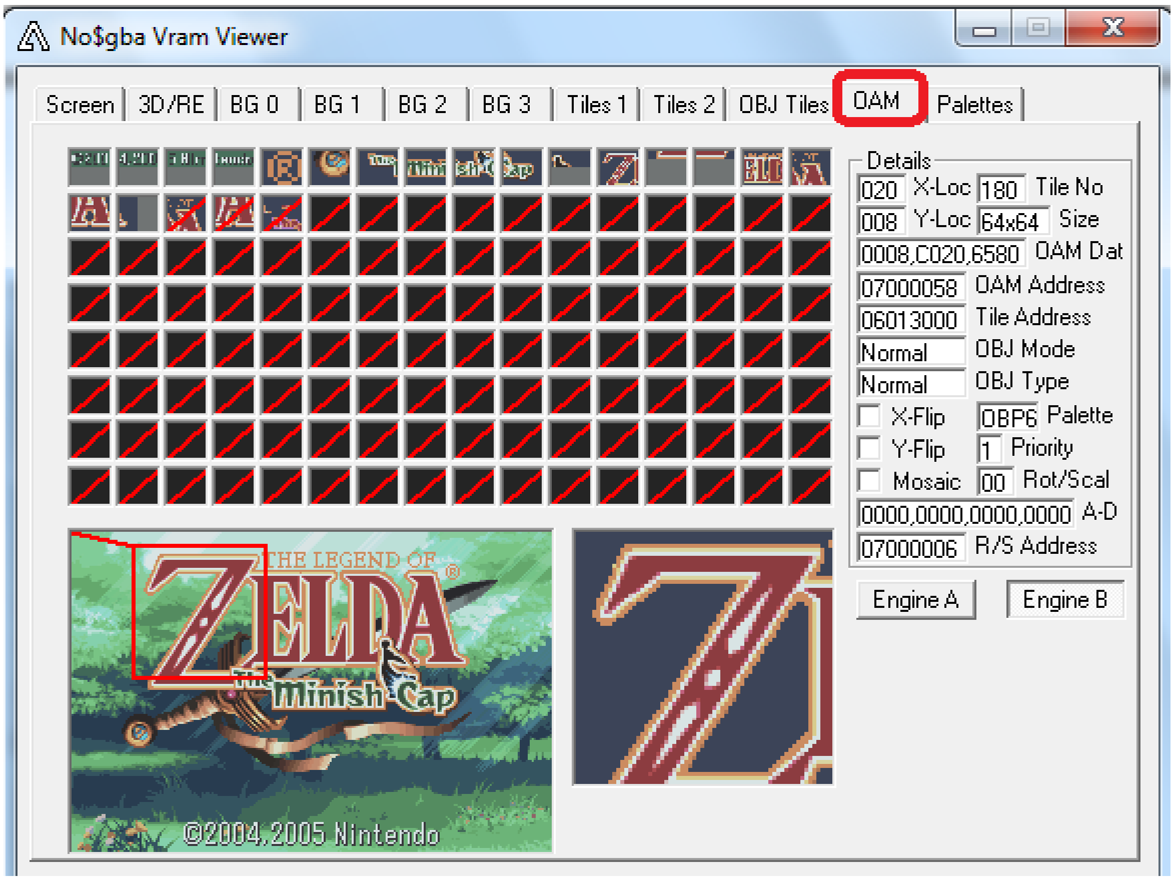
```
00000000 FE FF FF EA F
00000010 FE FF FF EA F
00000020 0F 50 2D E9 0
00000030 0F 50 BD E8 0
00000040 00 B0 4F E1 0
00000050 0B F0 29 E1 0
00000008
```

Details
Text Mode
17 X-Loc 0000 Tile No
1F Y-Loc 0000 Attribute
0600EFEE Map Address
06000000 Tile Address
X-Flip BGP 0 Palette
Y-Flip
Options
[✓] Grid [✓] Scyx

```
om ???
6
5
4
om Lxx_8016E78h
4
5
```

System DI GBA 1/1

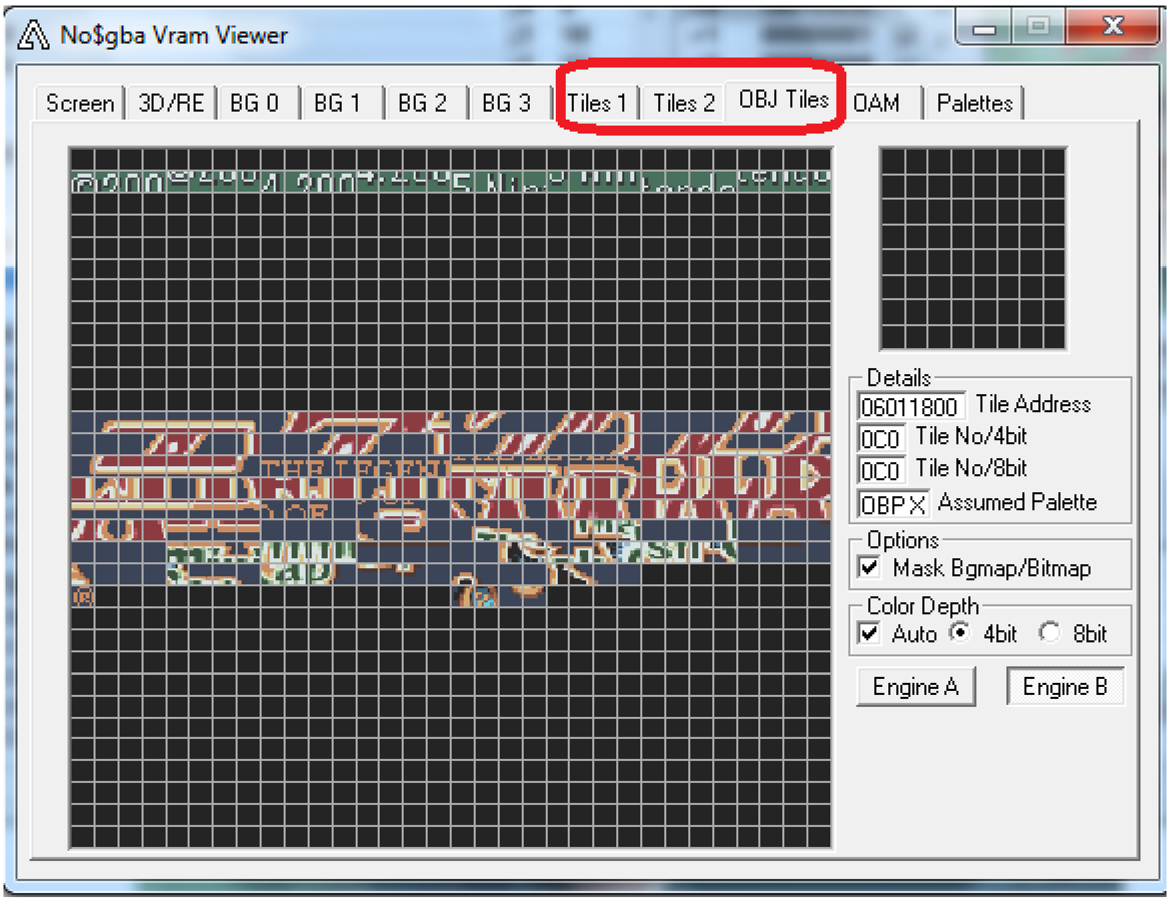
In this In this case, it's not, so it must be a sprite. To confirm this, open the OAM viewer and scroll across until you find the parts of the logo.



No$gba Vram Viewer
Screen 3D/RE BG 0 BG 1 BG 2 BG 3 Tiles 1 Tiles 2 OBJ Tiles OAM Palettes
Engine A Engine B

Details
020 X-Loc 180 Tile No
008 Y-Loc 64x64 Size
0008,C020,6580 OAM Dat
07000058 OAM Address
06013000 Tile Address
Normal OBJ Mode
Normal OBJ Type
X-Flip OBP6 Palette
Y-Flip 1 Priority
Mosaic 00 Rot/Scal
0000,0000,0000,0000 A-D
07000006 R/S Address

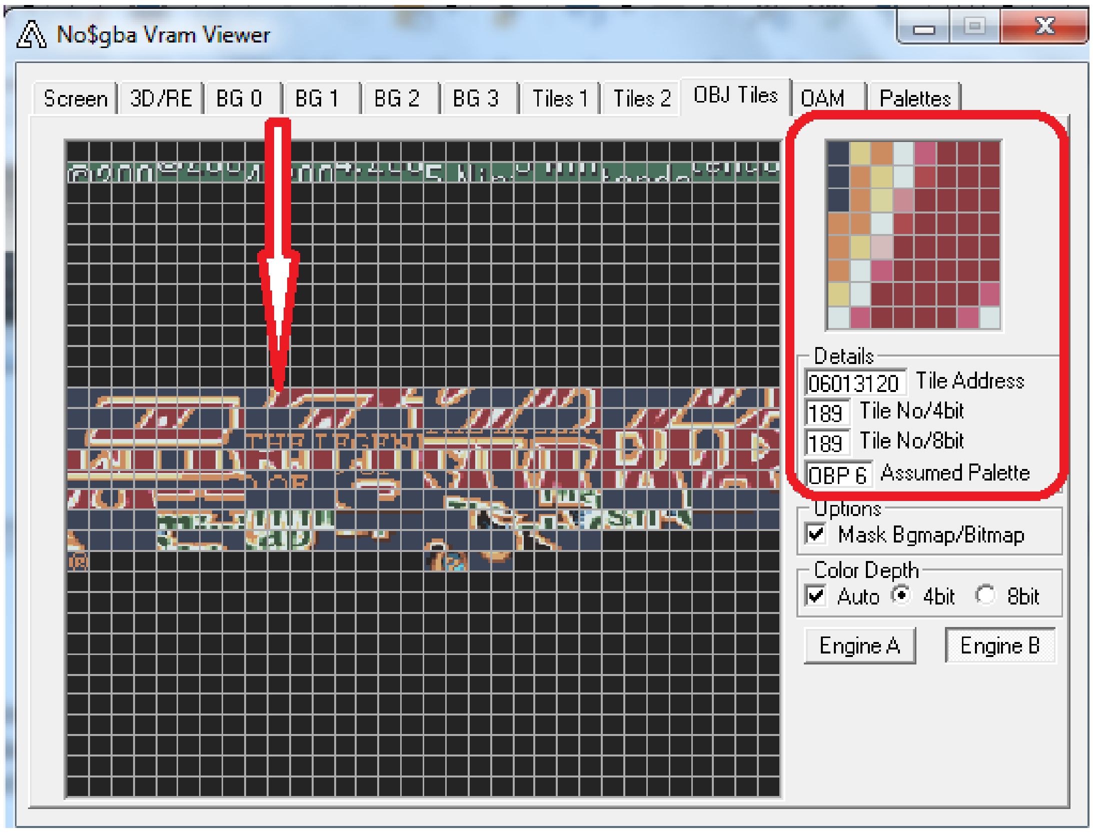
A quick background on GBA sprites. The actual graphics are stored at 0x06010000, (OAM data, which controls position, scaling etc..., is at 0x07000000), that’s why now, you have to open the tile viewer and select the 0x601000 char base.



Details
06011800 Tile Address
0C0 Tile No/4bit
0C0 Tile No/8bit
OBP X Assumed Palette
Options
✓ Mask Bgmap/Bitmap
Color Depth
✓ Auto [✓ 4bit] 8bit

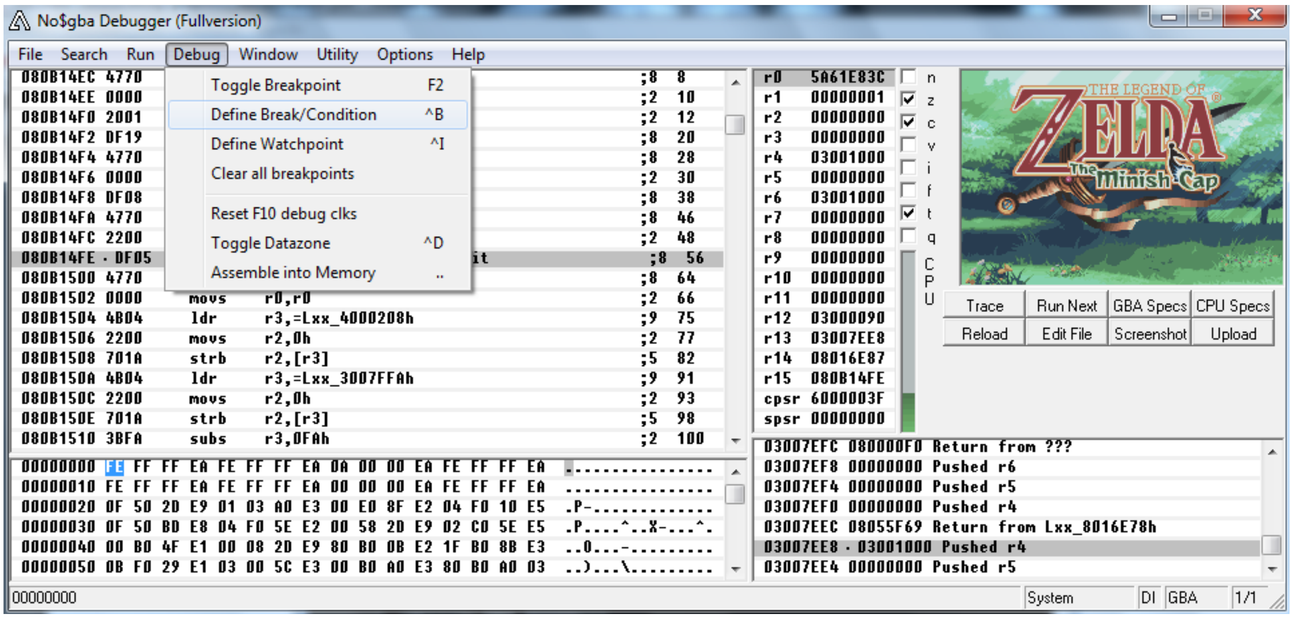
You should see the tiles that make up the logo (note the three different palettes used for the red logo, the green 'Minish Cap' text, and the sword.

So, now we need to figure out where in the ROM this data has come from. Click on one of the tiles that make up the logo (for this example, we'll use the top-right of the 'Z').



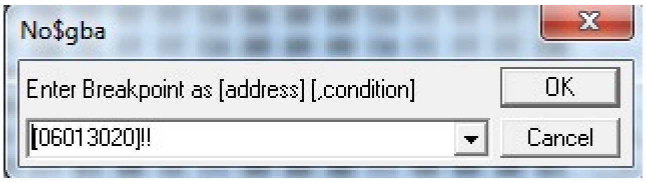
Details
06013120 Tile Address
189 Tile No/4bit
189 Tile No/8bit
OBP 6 Assumed Palette
Options
✓ Mask Bgmap/Bitmap
Color Depth
✓ Auto [✓ 4bit] 8bit

The Tile Viewer will give you the address for this tile – 06013020. Now, we will be using this information to do our reverse engineering. In addition, we will use the write breakpoints from no$gba. Which you can do by pressing ctrl+B or following the image below:


No$gba Debugger (Fullversion)
File Search Run Debug Window Utility Options Help
Toggle Breakpoint F2
Define Break/Condition ^B
Define Watchpoint ^I
Clear all breakpoints
Reset F10 debug clks
Toggle Datazone ^D
Assemble into Memory ..
Trace Run Next GBA Specs CPU Specs
Reload Edit File Screenshot Upload
System DI GBA 1/1

```
080B14EC 4770                              ;8  8
080B14EE 0000                              ;2  10
080B14F0 2001                              ;2  12
080B14F2 DF19                              ;8  20
080B14F4 4770                              ;8  28
080B14F6 0000                              ;2  30
080B14F8 DF08                              ;8  38
080B14FA 4770                              ;8  46
080B14FC 2200                              ;2  48
080B14FE · DF05                       it   ;8  56
080B1500 4770                              ;8  64
080B1502 0000      movs    r0,r0           ;2  66
080B1504 4B04      ldr     r3,=Lxx_4000208h ;9 75
080B1506 2200      movs    r2,0h           ;2  77
080B1508 701A      strb    r2,[r3]         ;5  82
080B150A 4B04      ldr     r3,=Lxx_3007FFAh ;9 91
080B150C 2200      movs    r2,0h           ;2  93
080B150E 701A      strb    r2,[r3]         ;5  98
080B1510 3BFA      subs    r3,0FAh         ;2  100

00000000 FE FF FF EA FE FF FF EA 0A 00 00 EA FE FF FF EA  ................
00000010 FE FF FF EA FE FF FF EA 00 00 00 EA FE FF FF EA  ................
00000020 0F 50 2D E9 01 03 A0 E3 00 E0 8F E2 04 F0 10 E5  .P-.............
00000030 0F 50 BD E8 04 F0 5E E2 00 58 2D E9 02 C0 5E E5  .P....^..X-...^.
00000040 00 B0 4F E1 00 08 2D E9 80 B0 0B E2 1F B0 8B E3  ..O...-.........
00000050 0B F0 29 E1 03 00 5C E3 00 B0 A0 E3 80 B0 A0 03  ..)...\.........

r0   5A61E83C   n
r1   00000001   ✓ z
r2   00000000   ✓ c
r3   00000000   v
r4   03001000   i
r5   00000000   f
r6   03001000   ✓ t
r7   00000000   q
r8   00000000
r9   00000000
r10  00000000
r11  00000000
r12  03000090
r13  03007EE8
r14  08016E87
r15  080B14FE
cpsr 6000003F
spsr 00000000

03007EFC 080000F0 Return from ???
03007EF8 00000000 Pushed r6
03007EF4 00000000 Pushed r5
03007EF0 00000000 Pushed r4
03007EEC 08055F69 Return from Lxx_8016E78h
03007EE8 · 03001000 Pushed r4
03007EE4 00000000 Pushed r5

00000000
```

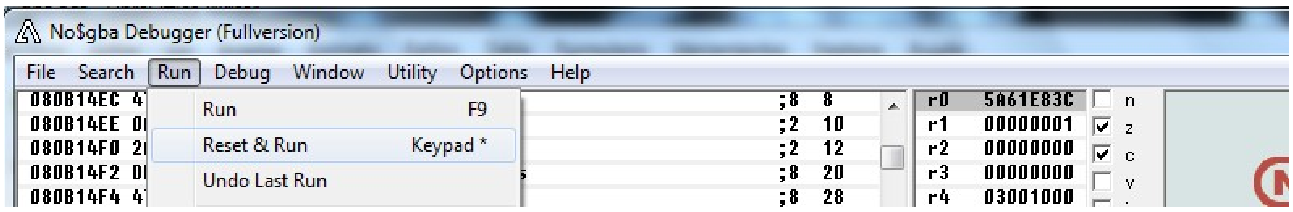
After that, a prompt will pop up. There, you Place a write breakpoint by typing in: **[06013020]!!**



Enter Breakpoint as [address] [,condition]

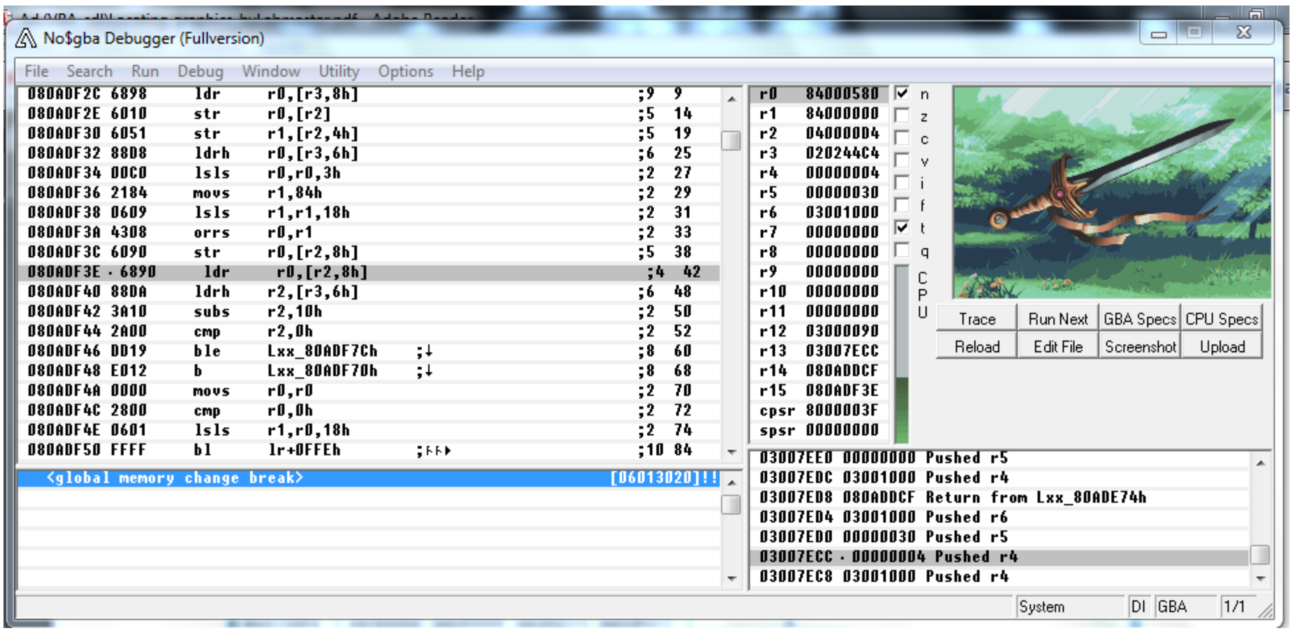
[06013020]!!

This puts a write breakpoint in 06013020, now reset & run the emulation.


No$gba Debugger (Fullversion)
File Search Run Debug Window Utility Options Help
Run F9
Reset & Run Keypad *
Undo Last Run

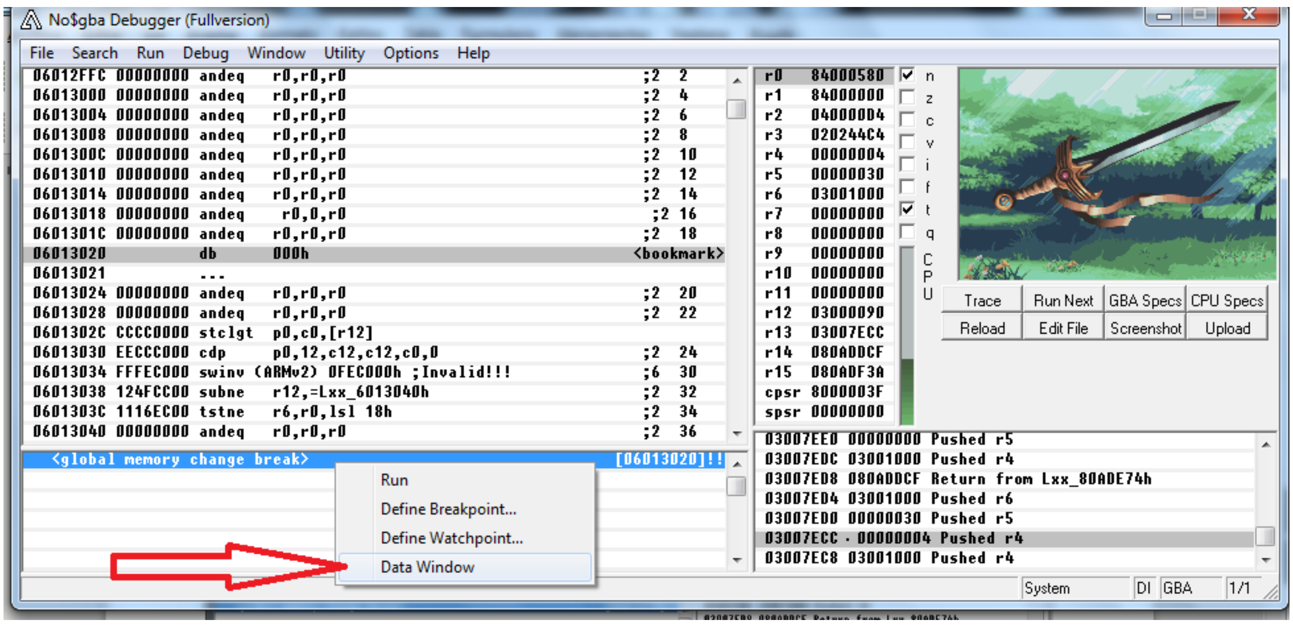
```
080B14EC 4      ;8  8      r0  5A61E83C  n
080B14EE 0      ;2  10     r1  00000001  ✓ z
080B14F0 2      ;2  12     r2  00000000  ✓ c
080B14F2 D      ;8  20     r3  00000000  v
080B14F4 4      ;8  28     r4  03001000
```

It should break thus:


No$gba Debugger (Fullversion)
File Search Run Debug Window Utility Options Help
Trace Run Next GBA Specs CPU Specs
Reload Edit File Screenshot Upload
System DI GBA 1/1

```
080ADF2C 6898      ldr     r0,[r3,8h]             ;9  9
080ADF2E 6010      str     r0,[r2]                ;5  14
080ADF30 6051      str     r1,[r2,4h]             ;5  19
080ADF32 88D8      ldrh    r0,[r3,6h]             ;6  25
080ADF34 00C0      lsls    r0,r0,3h               ;2  27
080ADF36 2184      movs    r1,84h                 ;2  29
080ADF38 0609      lsls    r1,r1,18h              ;2  31
080ADF3A 4308      orrs    r0,r1                  ;2  33
080ADF3C 6090      str     r0,[r2,8h]             ;5  38
080ADF3E · 6890     ldr     r0,[r2,8h]             ;4  42
080ADF40 88DA      ldrh    r2,[r3,6h]             ;6  48
080ADF42 3A10      subs    r2,10h                 ;2  50
080ADF44 2A00      cmp     r2,0h                  ;2  52
080ADF46 DD19      ble     Lxx_80ADF7Ch    ;↓     ;8  60
080ADF48 E012      b       Lxx_80ADF70h    ;↓     ;8  68
080ADF4A 0000      movs    r0,r0                  ;2  70
080ADF4C 2800      cmp     r0,0h                  ;2  72
080ADF4E 0601      lsls    r1,r0,18h              ;2  74
080ADF50 FFFF      bl      lr+0FFEh        ;▸▸▸   ;10 84

  <global memory change break>                    [06013020]!!

r0   84000580   ✓ n
r1   84000000   z
r2   040000D4   c
r3   020244C4   v
r4   00000004   i
r5   00000030   f
r6   03001000   ✓ t
r7   00000000   q
r8   00000000
r9   00000000
r10  00000000
r11  00000000
r12  03000090
r13  03007ECC
r14  080ADDCF
r15  080ADF3E
cpsr 8000003F
spsr 00000000

03007EE0 00000000 Pushed r5
03007EDC 03001000 Pushed r4
03007ED8 080ADDCF Return from Lxx_80ADE74h
03007ED4 03001000 Pushed r6
03007ED0 00000030 Pushed r5
03007ECC · 00000004 Pushed r4
03007EC8 03001000 Pushed r4
```

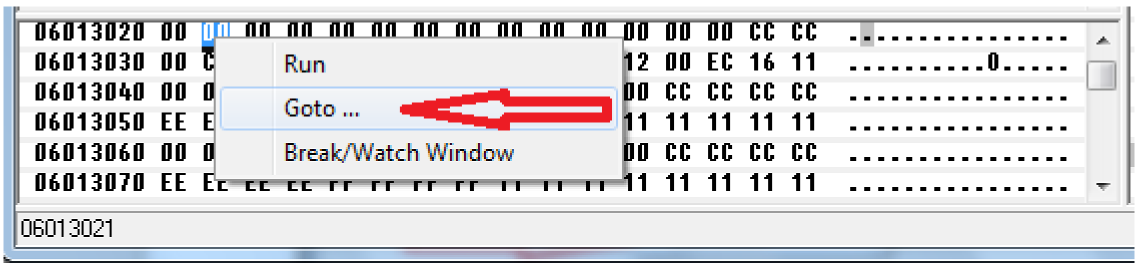
How do we know that this is the right break? If you open the memory viewer and see what 06013020 is whilst on the menu, you'll know that these are the correct values.



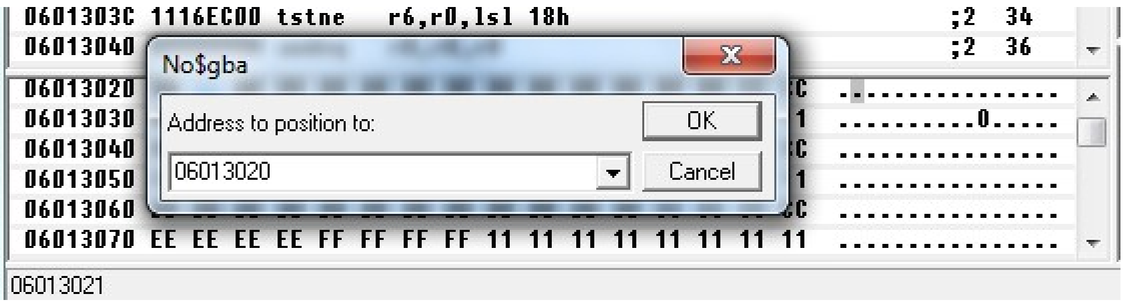
```
06012FFC 00000000 andeq   r0,r0,r0                  ;2  2
06013000 00000000 andeq   r0,r0,r0                  ;2  4
06013004 00000000 andeq   r0,r0,r0                  ;2  6
06013008 00000000 andeq   r0,r0,r0                  ;2  8
0601300C 00000000 andeq   r0,r0,r0                  ;2  10
06013010 00000000 andeq   r0,r0,r0                  ;2  12
06013014 00000000 andeq   r0,r0,r0                  ;2  14
06013018 00000000 andeq    r0,0,r0                   ;2 16
0601301C 00000000 andeq   r0,r0,r0                  ;2  18
06013020          db      000h                    <bookmark>
06013021          ...
06013024 00000000 andeq   r0,r0,r0                  ;2  20
06013028 00000000 andeq   r0,r0,r0                  ;2  22
0601302C CCCC0000 stclgt  p0,c0,[r12]
06013030 EECCC000 cdp     p0,12,c12,c12,c0,0        ;2  24
06013034 FFFEC000 swinv (ARMv2) 0FEC000h ;Invalid!!! ;6  30
06013038 124FCC00 subne   r12,=Lxx_6013040h         ;2  32
0601303C 1116EC00 tstne   r6,r0,lsl 18h             ;2  34
06013040 00000000 andeq   r0,r0,r0                  ;2  36
  <global memory change break>                [06013020]!!
```

```
r0   84000580   [✓] n
r1   84000000   [ ] z
r2   040000D4   [ ] c
r3   020244C4   [ ] v
r4   00000004   [ ] i
r5   00000030   [ ] f
r6   03001000   [✓] t
r7   00000000   [ ] q
r8   00000000
r9   00000000
r10  00000000
r11  00000000
r12  03000090
r13  03007ECC
r14  080ADDCF
r15  080ADF3A
cpsr 8000003F
spsr 00000000
```

```
03007EE0 00000000 Pushed r5
03007EDC 03001000 Pushed r4
03007ED8 080ADDCF Return from Lxx_80ADE74h
03007ED4 03001000 Pushed r6
03007ED0 00000030 Pushed r5
03007ECC · 00000004 Pushed r4
03007EC8 03001000 Pushed r4
```

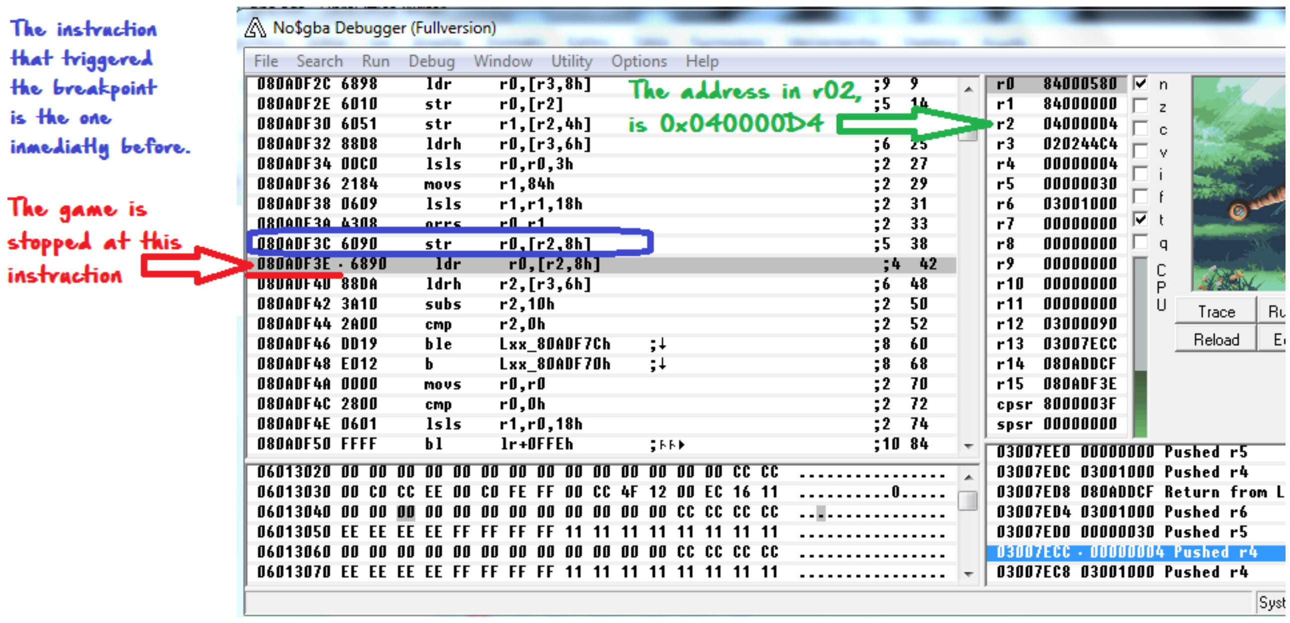


```
06013020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 CC CC  ................
06013030 00 C                              12 00 EC 16 11  ..........0.....
06013040 00 0                              00 CC CC CC CC  ................
06013050 EE E                              11 11 11 11 11  ................
06013060 00 0                              00 CC CC CC CC  ................
06013070 EE E                              11 11 11 11 11  ................
06013021
```



```
0601303C 1116EC00 tstne   r6,r0,lsl 18h                ;2  34
06013040                                               ;2  36
06013020                                      C  ................
06013030                                      1  ..........0.....
06013040                                      C  ................
06013050                                      1  ................
06013060                                      C  ................
06013070 EE EE EE EE FF FF FF FF 11 11 11 11 11 11 11 11  ................
06013021
```

So now we need to see what opcode triggered this breakpoint. In the diassembly viewer, go to 080adf3e and scroll up a bit. The instruction that triggered the breakpoint is the one immediately before, as you can see in the image below:

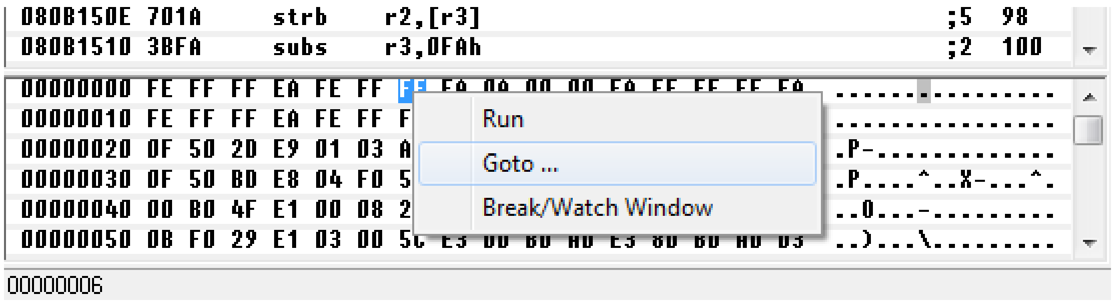

The instruction that triggered the breakpoint is the one inmediatly before.
The game is stopped at this instruction
The address in r02, is 0x040000D4
No$gba Debugger (Fullversion)
File Search Run Debug Window Utility Options Help
Trace
Reload

```
080ADF2C 6898     ldr    r0,[r3,8h]                      ;9  9
080ADF2E 6010     str    r0,[r2]                         ;5  14
080ADF30 6051     str    r1,[r2,4h]
080ADF32 88D8     ldrh   r0,[r3,6h]                      ;6  25
080ADF34 00C0     lsls   r0,r0,3h                        ;2  27
080ADF36 2184     movs   r1,84h                          ;2  29
080ADF38 0609     lsls   r1,r1,18h                       ;2  31
080ADF3A 4308     orrs   r0,r1                           ;2  33
080ADF3C 6090     str    r0,[r2,8h]                      ;5  38
080ADF3E · 6890    ldr    r0,[r2,8h]                     ;4  42
080ADF40 88DA     ldrh   r2,[r3,6h]                      ;6  48
080ADF42 3A10     subs   r2,10h                          ;2  50
080ADF44 2A00     cmp    r2,0h                           ;2  52
080ADF46 DD19     ble    Lxx_80ADF7Ch      ;↓            ;8  60
080ADF48 E012     b      Lxx_80ADF70h      ;↓            ;8  68
080ADF4A 0000     movs   r0,r0                           ;2  70
080ADF4C 2800     cmp    r0,0h                           ;2  72
080ADF4E 0601     lsls   r1,r0,18h                       ;2  74
080ADF50 FFFF     bl     lr+0FFEh          ;⊧⊧▸          ;10 84

06013020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 CC CC  ................
06013030 00 C0 CC EE 00 C0 FE FF 00 CC 4F 12 00 EC 16 11  ..........O.....
06013040 00 00 00 00 00 00 00 00 00 00 00 00 CC CC CC CC  ................
06013050 EE EE EE EE FF FF FF FF 11 11 11 11 11 11 11 11  ................
06013060 00 00 00 00 00 00 00 00 00 00 00 00 CC CC CC CC  ................
06013070 EE EE EE EE FF FF FF FF 11 11 11 11 11 11 11 11  ................

r0   84000580   ☑ n
r1   84000000   ☐ z
r2   040000D4   ☐ c
r3   020244C4   ☐ v
r4   00000004   ☐ i
r5   00000030   ☐ f
r6   03001000   ☑ t
r7   00000000   ☐ q
r8   00000000
r9   00000000
r10  00000000
r11  00000000
r12  03000090
r13  03007ECC
r14  080ADDCF
r15  080ADF3E
cpsr 8000003F
spsr 00000000

03007EE0 00000000 Pushed r5
03007EDC 03001000 Pushed r4
03007ED8 080ADDCF Return from L
03007ED4 03001000 Pushed r6
03007ED0 00000030 Pushed r5
03007ECC · 00000004 Pushed r4
03007EC8 03001000 Pushed r4
```

080adf3c 6090 str r0, [r2, #0x8]
This writes the value in r0 to the address in r02 + 0x8.

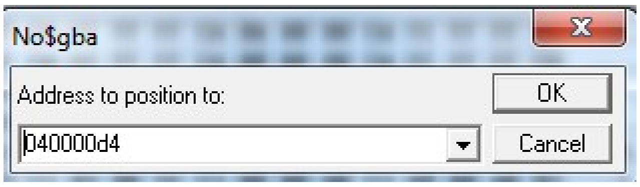
Now this is where it's handy to know your GBA IO registers. The address in r02, 0x040000d4, is one of those that control DMA transfers (https://problemkaputt.de/gbatek.htm#gbadmatransfers). 0X040000D4 + 0x8 is 040000DC, and we're writing a word to it - this actually corresponds to both the word count register and control register for DMA channel 3. (you can read the reason in the gbatek link, gbatek is one of the most useful document a GBA hacker could have).

The source register for DMA channel 3 is at 0x040000D4, so take a peek at the value there by using right clicking as shown in the image below:

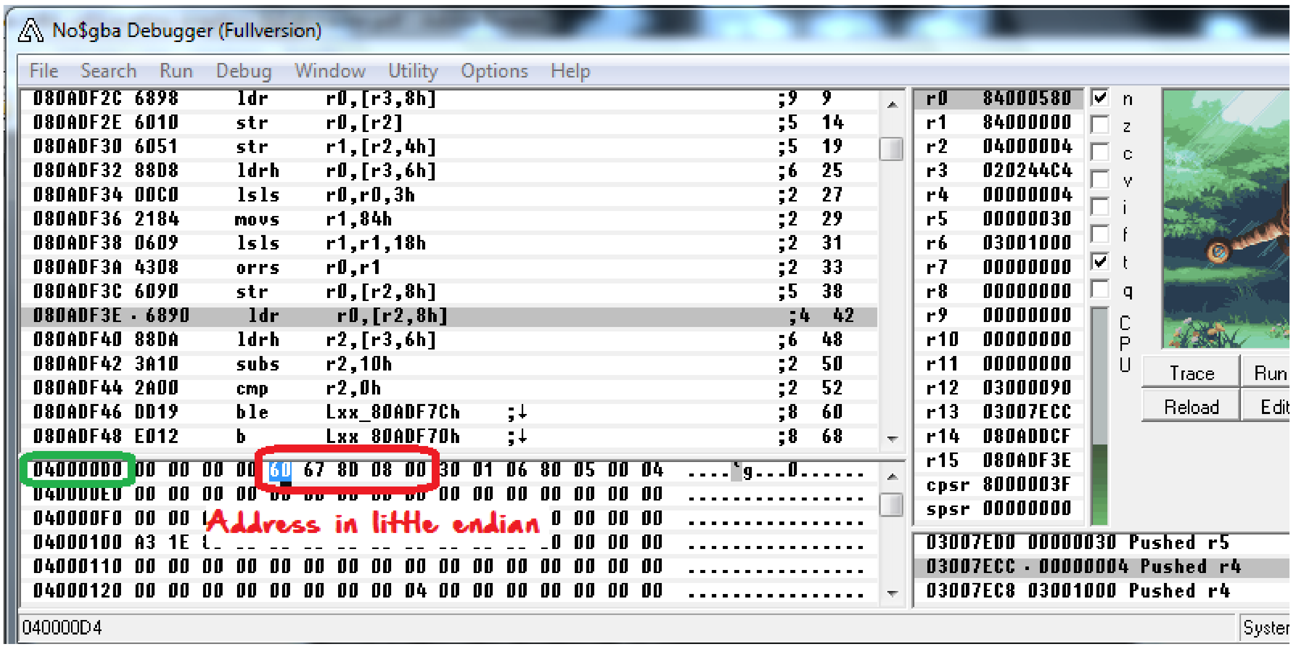


```
080B150E 701A     strb   r2,[r3]                 ;5  98
080B1510 3BFA     subs   r3,0FAh                 ;2  100

00000000 FE FF FF EA FE FF FE EA 0A 00 00 EA FE FF FF EA  ................
00000010 FE FF FF EA FE FF F                              ................
00000020 0F 50 2D E9 01 03 A                              .P-.............
00000030 0F 50 BD E8 04 F0 5                              .P....^..X-...^.
00000040 00 B0 4F E1 00 08 2                              ..O...-.........
00000050 0B F0 29 E1 03 00 5C E3 DD BD AD E3 8D BD AD D3  ..)...\.........
00000006
```

And goto: **040000d4**



You can see the result here:

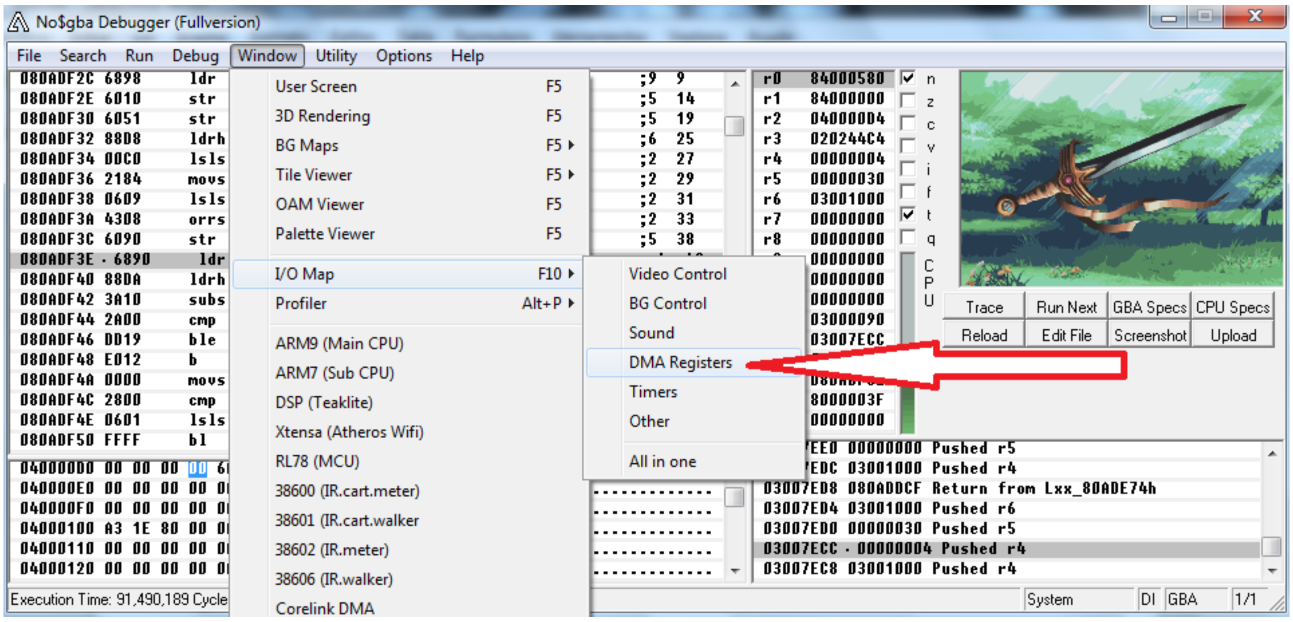

No$gba Debugger (Fullversion)
File Search Run Debug Window Utility Options Help
Address in little endian
Trace
Run
Reload
Edit
C P U
n z c v i f t q
040000D4
System

```
080ADF2C 6898     ldr     r0,[r3,8h]                 ;9  9
080ADF2E 6010     str     r0,[r2]                    ;5  14
080ADF30 6051     str     r1,[r2,4h]                 ;5  19
080ADF32 88D8     ldrh    r0,[r3,6h]                 ;6  25
080ADF34 00C0     lsls    r0,r0,3h                   ;2  27
080ADF36 2184     movs    r1,84h                     ;2  29
080ADF38 0609     lsls    r1,r1,18h                  ;2  31
080ADF3A 4308     orrs    r0,r1                      ;2  33
080ADF3C 6090     str     r0,[r2,8h]                 ;5  38
080ADF3E · 6890    ldr     r0,[r2,8h]                ;4  42
080ADF40 88DA     ldrh    r2,[r3,6h]                 ;6  48
080ADF42 3A10     subs    r2,10h                     ;2  50
080ADF44 2A00     cmp     r2,0h                      ;2  52
080ADF46 DD19     ble     Lxx_80ADF7Ch     ;↓        ;8  60
080ADF48 E012     b       Lxx_80ADF70h     ;↓        ;8  68

040000D0 00 00 00 00 60 67 8D 08 00 30 01 06 80 05 00 04  ....`g...0......
040000E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
040000F0 00 00 ..                          .. 0 00 00 00  ................
04000100 A3 1E ..                          .. 0 00 00 00  ................
04000110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
04000120 00 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00  ................

r0   84000580
r1   84000000
r2   040000D4
r3   020244C4
r4   00000004
r5   00000030
r6   03001000
r7   00000000
r8   00000000
r9   00000000
r10  00000000
r11  00000000
r12  03000090
r13  03007ECC
r14  080ADDCF
r15  080ADF3E
cpsr 8000003F
spsr 00000000

03007ED0   00000030 Pushed r5
03007ECC · 00000004 Pushed r4
03007EC8   03001000 Pushed r4
```

The address value you want is: 08 8d 67 60

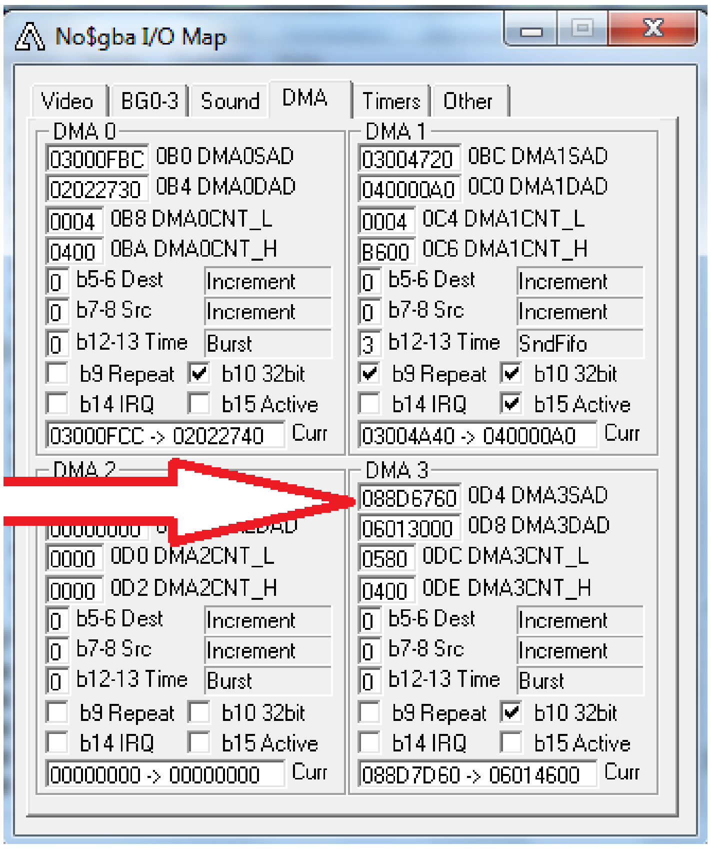
Little Endian > denoting or relating to a system of ordering data in which the least significant units are put first.

You could also do it checking the DMA Registers with:



```
080ADF2C 6898     ldr
080ADF2E 6010     str
080ADF30 6051     str
080ADF32 88D8     ldrh
080ADF34 00C0     lsls
080ADF36 2184     movs
080ADF38 0609     lsls
080ADF3A 4308     orrs
080ADF3C 6090     str
080ADF3E · 6890    ldr
080ADF40 88DA     ldrh
080ADF42 3A10     subs
080ADF44 2A00     cmp
080ADF46 DD19     ble
080ADF48 E012     b
080ADF4A 0000     movs
080ADF4C 2800     cmp
080ADF4E 0601     lsls
080ADF50 FFFF     bl

040000D0 00 00 00 00 6
040000E0 00 00 00 00 0
040000F0 00 00 00 00 0
04000100 A3 1E 80 00 0
04000110 00 00 00 00 0
04000120 00 00 00 00 0

          EED0 00000000 Pushed r5
          EDC 03001000 Pushed r4
03007ED8   080ADDCF Return from Lxx_80ADE74h
03007ED4   03001000 Pushed r6
03007ED0   00000030 Pushed r5
03007ECC · 00000004 Pushed r4
03007EC8   03001000 Pushed r4
```

And see the value here:

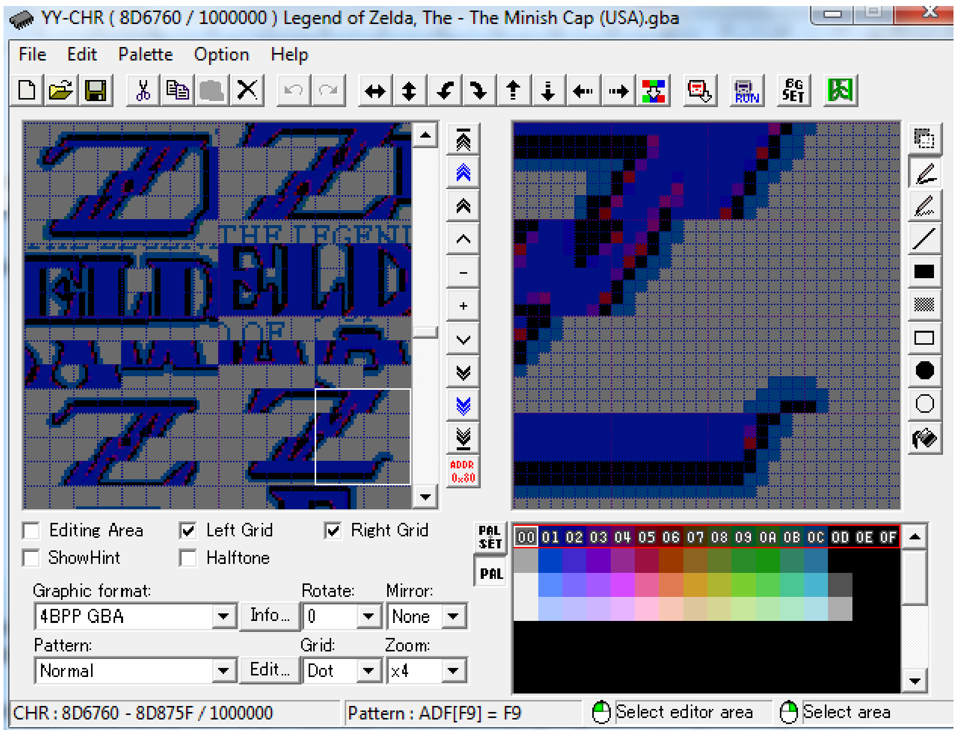

No$gba I/O Map

Video | BG0-3 | Sound | DMA | Timers | Other

```
DMA 0                                   DMA 1
03000FBC  0B0 DMA0SAD                   03004720  0BC DMA1SAD
02022730  0B4 DMA0DAD                   040000A0  0C0 DMA1DAD
0004  0B8 DMA0CNT_L                     0004  0C4 DMA1CNT_L
0400  0BA DMA0CNT_H                     B600  0C6 DMA1CNT_H
0  b5-6 Dest      Increment             0  b5-6 Dest      Increment
0  b7-8 Src       Increment             0  b7-8 Src       Increment
0  b12-13 Time    Burst                 3  b12-13 Time    SndFifo
[ ] b9 Repeat  [✓] b10 32bit            [✓] b9 Repeat  [✓] b10 32bit
[ ] b14 IRQ    [ ] b15 Active           [ ] b14 IRQ    [✓] b15 Active
03000FCC -> 02022740  Curr              03004A40 -> 040000A0  Curr

DMA 2                                   DMA 3
                                        088D6760  0D4 DMA3SAD
00000000  0    DAD                      06013000  0D8 DMA3DAD
0000  0D0 DMA2CNT_L                     0580  0DC DMA3CNT_L
0000  0D2 DMA2CNT_H                     0400  0DE DMA3CNT_H
0  b5-6 Dest      Increment             0  b5-6 Dest      Increment
0  b7-8 Src       Increment             0  b7-8 Src       Increment
0  b12-13 Time    Burst                 0  b12-13 Time    Burst
[ ] b9 Repeat  [ ] b10 32bit            [ ] b9 Repeat  [✓] b10 32bit
[ ] b14 IRQ    [ ] b15 Active           [ ] b14 IRQ    [ ] b15 Active
00000000 -> 00000000  Curr              088D7D60 -> 06014600  Curr
```

This means that, the data has been copied from 088D6760. Note that the next word in memory is 06013000, sounds familiar? It's the destination address (040000d8 is the destination register for channel 3).

Now, we convert that location into a gba rom memory address by substracting hexidecimal 8000000: **0x088D6760 - 0x8000000 = 0x8D6760** and go to that address in our graphics editor of preference, for this tutorial we will show the results in yy-chr as an example, below:


YY-CHR (8D6760 / 1000000) Legend of Zelda, The - The Minish Cap (USA).gba

File Edit Palette Option Help

```
[ ] Editing Area   [✓] Left Grid   [✓] Right Grid
[ ] ShowHint       [ ] Halftone

Graphic format: 4BPP GBA   Info...   Rotate: 0   Mirror: None
Pattern: Normal            Edit...   Grid: Dot   Zoom: x4

CHR : 8D6760 - 8D875F / 1000000    Pattern : ADF[F9] = F9    Select editor area    Select area
```

Very Important Caveat: Although the steps up to setting the breakpoint will be the same for most games (though the graphics might be in one of the other Character bases in the tile viewer), the way the game gets that data into VRAM will differ. Some may use a Software Interrupt (SWI) to copy or decompress the data from a ROM address - others will have their own functions to decompress graphics (sometimes directly to VRAM, others into WRAM, then copied into VRAM using DMA or a SWI, or even with stmia/ldmia opcodes), so it's up to you to figure out what's going on past this point.

That concludes this brief tutorial, happy hacking and reverse engineering.

- Curiosity leads to knowledge, be curious!