



FACULTY  
OF INFORMATICS  
Masaryk University

# Curso de Desenvolvimento GBA

---

## 1. Introdução ao GBA

João Paulo Taylor Ienczak Zanette

# Índice I

1. Introdução
  - 1.1 Contextualização do Curso
  - 1.2 Evolução dos consoles
2. Conhecendo a plataforma
  - 2.1 O interior do GBA
3. Vídeo
  - 3.1 Funcionamento
  - 3.2 Registrador de Controle do Visor
  - 3.3 VRAM
4. Mão à obra
  - 4.1 Introdução ao Assembly ARMv4
  - 4.2 DevKitPro
5. Extras
  - 5.1 DevKitPro

# Índice II

## 5.2 VisualBoyAdvance

# Objetivos

- Ensinar programação de baixo-nível (comunicação direta com hardware/integração com assembly);
- Ensinar técnicas de programação aplicadas;
- Mostrar o funcionamento de imagens/gráficos e áudio no mundo digital;
- Relacionar as tecnologias vistas com as utilizadas atualmente.

# Programação

- Assembly ARM7-TDMI — Modo Thumb (GBA);
- OpenGL (NDS);
- C++ (GBA/NDS).

A mesma forma de programação para GBA serve também para: GB, GBC, NES, SNES, MegaDrive, SegaSaturn e PSX (PS1).

# Circuitos e Técnicas Digitais

- Leitura/escrita de registradores (em que cada bit é mapeado para uma função específica) via programação;

# Sistemas Digitais

- Compreensão a respeito de como o Assembly gerado pela compilação altera o estado/memória do circuito;
- Compreensão do sistema que gera imagens em um circuito digital (VGA, LCD, etc...);
- Funcionamento (inclusive a nível de circuito) da execução de músicas em formato de instrução MIDI;
- Técnicas de otimização através de Hardware.

# Computação Gráfica

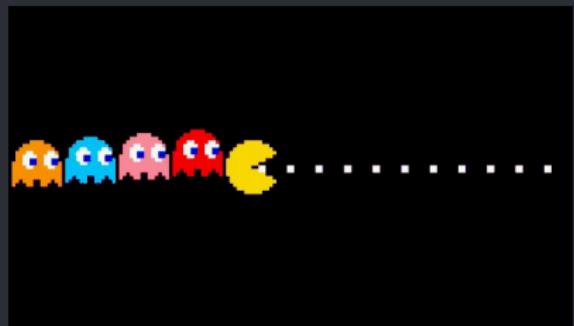
- Desenho de primitivas (linhas, triângulos, circuitos, etc...);
- Aceleração gráfica via Hardware.

# Notações

0b???	«??» está em binário ( $0b11111111 = 255$ ).
0x???	«??» está em hexadecimal ( $0xFF = 255$ ).
???b	«??» está em binário ( $11111111b = 255$ ).
???h	«??» está em hexadecimal ( $FFh = 255$ ).
????:????	«:» serve apenas para melhor visualização ( $0x04000000 = 0400:0000h$ ).
??? >> x	«??» deslocado “x” bits para direita.
??? << x	«??» deslocado “x” bits para esquerda.

# Hello, World!

Um breve prólogo das especificações técnicas e limitações dos consoles de VideoGame ao longo do tempo e uma análise de sua evolução.



# Atari 2600 (1977)

**Processador:** MOS Technology  
6507 (variante do  
6502 de 1975)

**Barramento:** 8 bits

**Clock:** 1.19MHz

**RAM:** 128 bytes

**ROM:** 16KB

**Resolução:**

- $160 \times 192$   
(NTSC)
- $160 \times 228$   
(PAL)

**Cores:** 128

**Som:** 2 canais (1 chip  
cada)



# NES (1983)

**Processador:** MOS 6502 Customizado

**Barramento:** 8 bits

**Clock (CPU):** 1.79MHz (NTSC),  
1.66MHz (PAL)

**Clock (GPU):** 5.37MHz (NTSC),  
5.33MHz (PAL)

**RAM:** 2KiB + RAM Expandida  
(do cartucho)

**ROM:** 48KB

**Resolução:** 256 × 240

**Cores:** 56 cores (paleta básica)

**Cores na tela:** 25 cores por scanline (cor

de fundo + 4 conjuntos  
de 3 cores de tiles + 4  
conjuntos de cores por  
sprite)

**OAM:** 256 bytes

**Dim. das Sprites:** 16 × 16 ou 24 × 24

**Máx. Sprites na tela:** 64

**Som:** 5 canais (2 square, 1  
triangle, 1 ruído-branco, 1  
modulação de código  
delta-pulse (DPCM) de 6  
bits)



00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D

# SNES (1990)

**Processador:** Ricoh 5A22 customizado da Nintendo

**Barramento:** 16 bits

**Clock (CPU):** 1.79MHz, 2.86MHz ou 3.58MHz

**Clock (GPU):** Mesmo da CPU

**RAM:** 128KB

**VRAM:** 64KB (512 + 32 bytes de sprite, 256×15 bits de paleta)

**RAM (Áudio):** 64KB

**Resolução:** 256 × 224/512 × 448

**Cores:** 32768 (15 bits)

**OAM:** 544 bytes

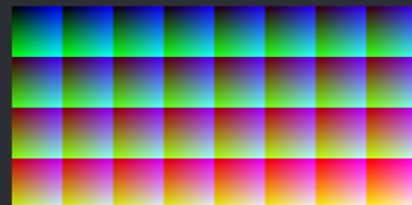
**Dim. das sprites:** 8 × 8, 16 × 16, 32 × 32 e 64 × 64

**Cores sprite:** 16

**Máx. sprites na tela:** 128 (32 na mesma linha)

**Camadas de background:** 4

**Som:** 8 canais (32KHz 16-bit stereo).



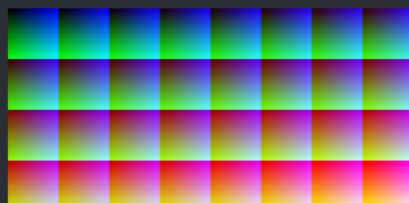
# GameBoy (1989)

<b>Processador:</b> Sharp LR35902 Customizado	<b>VRAM:</b> 8KB (interno)
<b>Barramento:</b> 8 bits	<b>Resolução:</b> 160x144
<b>Clock (CPU):</b> 4.19MHz	<b>Cores:</b> 2 bits (4 tons de «cinza»)
<b>RAM:</b> 8KB (podendo ser expandido para 32KB)	<b>OAM:</b> 160 bytes (4 bytes/sprite)
<b>ROM:</b> 256 bytes (interno), 256K/512K/1M/2M/4M/8M (cartuchos)	<b>Dim. das sprites:</b> 8x8, 8x16 <b>Cores/sprite:</b> 16 <b>Máx. sprites na tela:</b> 40 <b>Som:</b> 2 geradores de pulso de onda



# GameBoy Advance (2001)

<b>Processador:</b>	ARM7-TDMI com memória embarcada	mode), 32768 cores (bitmap mode)
<b>Barramento:</b>	16 bits	<b>OAM:</b> 1KB (128 objetos de 3x16bit, 32 transformações de objetos de 4x16bit)
<b>Co-processador:</b>	Z80 8-bit de 4/8MHz (para compatibilidade com GB/GBC)	<b>Dim. das sprites:</b> 8x8, 8x16, 8x32, 16x8, 16x16, 16x32, 32x8, 32x16, 32x32, 32x64, 64x32, 64x64
<b>Clock (CPU/GPU):</b>	16.8MHz/~5.5MHz <b>(59.73FPS)</b>	<b>Máx. sprites na tela:</b> 256
<b>SRAM/DRAM:</b>	32KB/256KB	<b>Som:</b> Dual 8-bit DAC para som Stereo (DirectSound)
<b>VRAM:</b>	92KB (interno)	
<b>Resolução:</b>	240x160 (3:2)	
<b>Cores:</b>	15-bit BGR (5 bits/canal), 512 cores (character)	



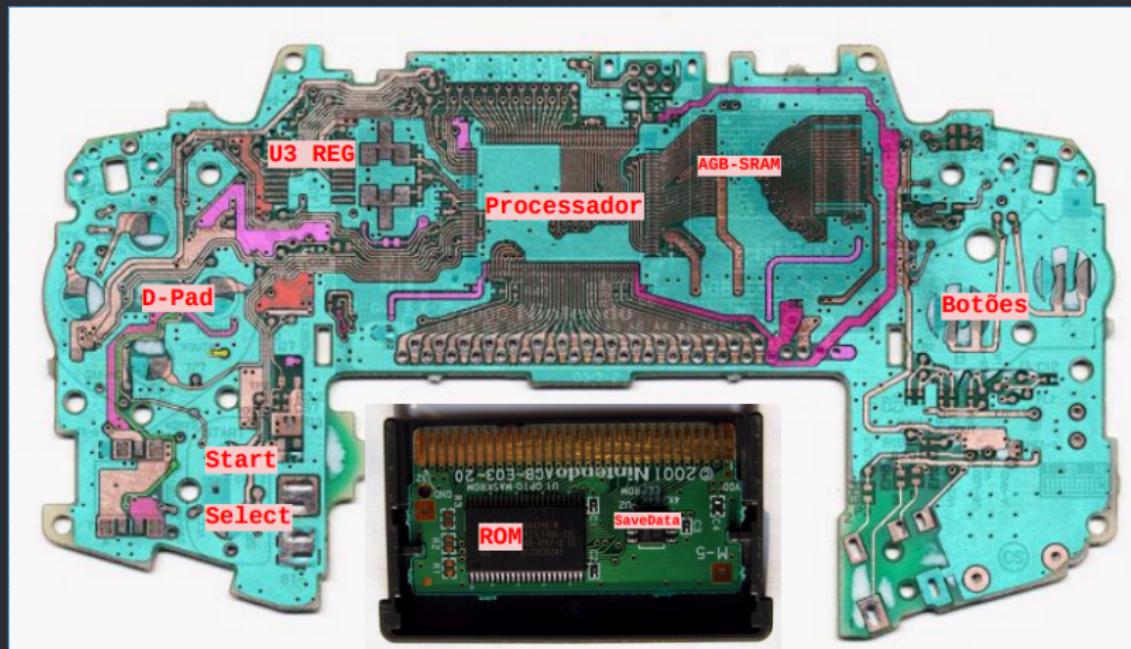
## GameBoy Advance (2001)

**ARM7-TDMI** : ARM7 + 16-bit Thumb + JTAG Debug + fast Multiplier  
+ enhanced JICE.

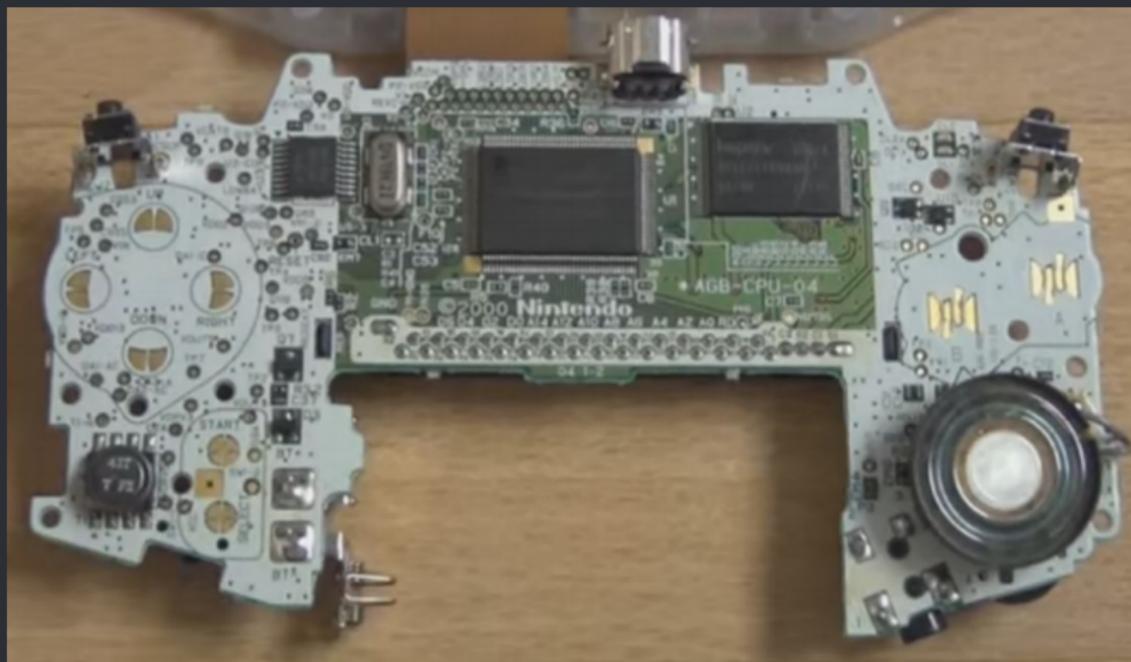
**DAC:** Digital-Analogic-Converter

# O interior do GBA

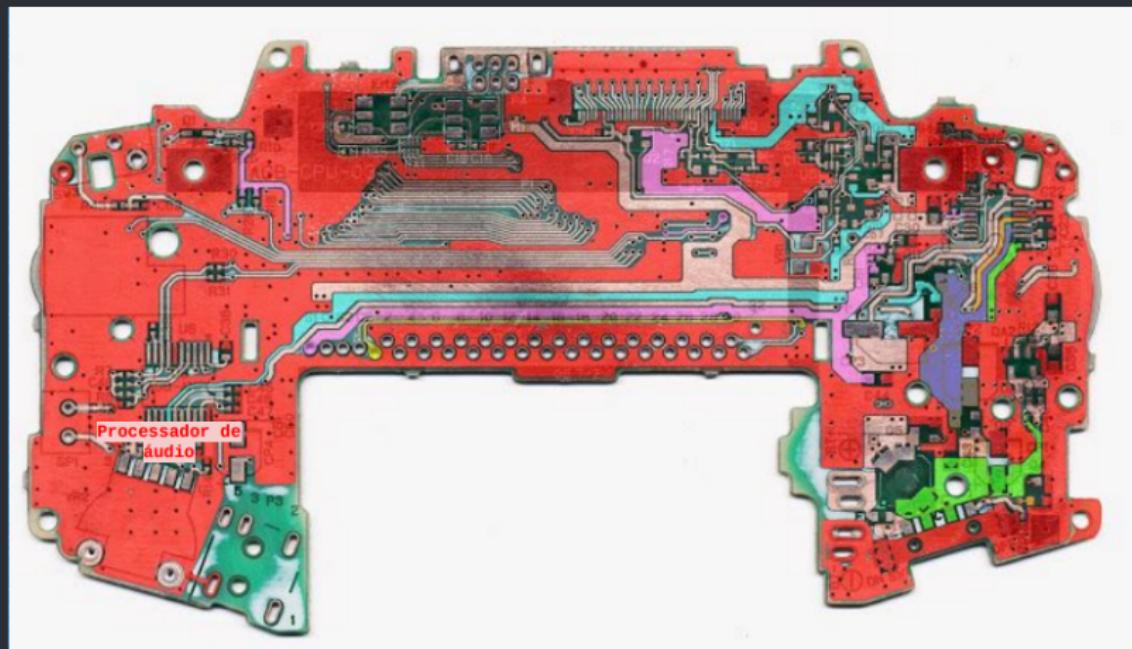
# Interior do GBA (frontal)



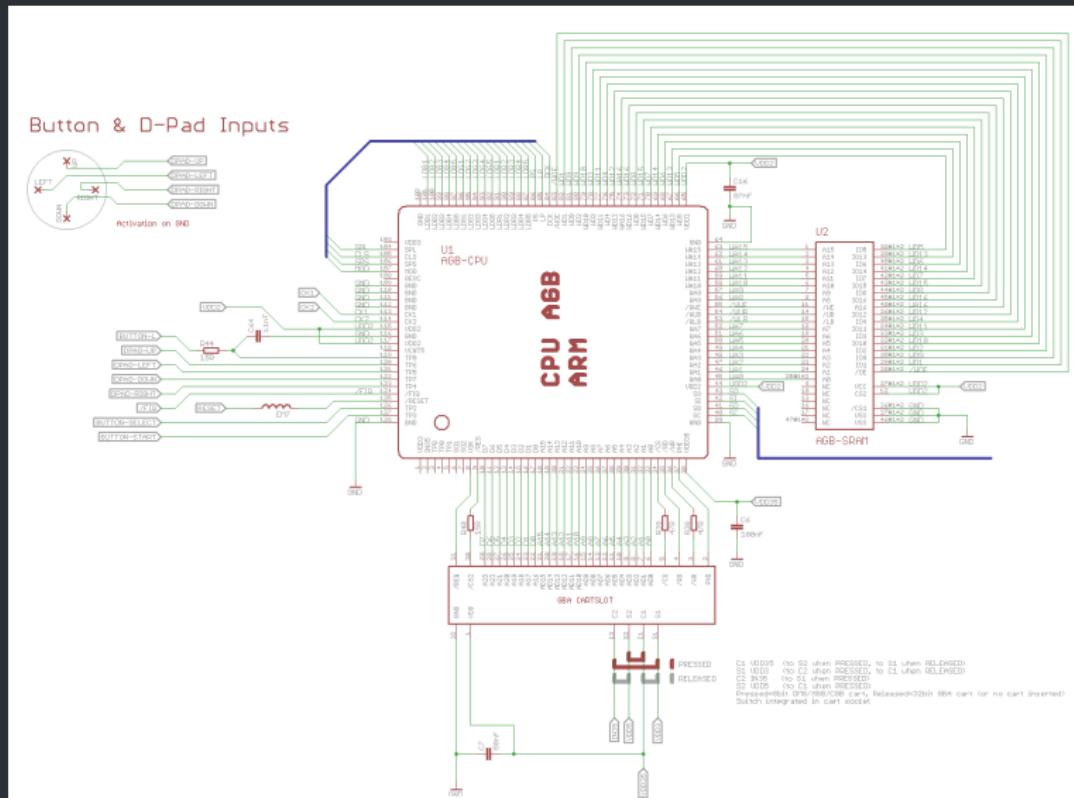
## Interior do GBA (frontal)



## Interior do GBA (trás)



# Interior do GBA: Processador



## Organização da Memória (Geral)

Descrição	Início	Tamanho
BIOS - ROM do Sistema	0x00XX:XXXX	0x0:3FFF (16KB)
WorkRAM On-Board	0x02XX:XXXX	0x3:FFFF (256KB)
WorkRAM On-Chip	0x03XX:XXXX	0x0:7FFF (32KB)
Registradores de I/O	0x04XX:XXXX	0x0:03FE (~1KB)

# Organização da Memória (Vídeo)

Descrição	Início	Tamanho
Paleta de BG/OBJ	0x05XX:XXXX	0x0:03FF (1KB)
RAM de Vídeo (VRAM)	0x06XX:XXXX	0x1:7FFF (96KB)
OAM (Obj. Attr. Mem.)	0x07XX:XXXX	0x0:03FF (1KB)

## Organização da Memória (GamePak)

Descrição	Início	Tamanho
FlashROM (ws0)	0x08XX:XXXX	0x1FF:FFFF (16KB)
FlashROM (ws1)	0x0AXX:XXXX	0x1FF:FFFF (256KB)
FlashROM (ws2)	0x0CXX:XXXX	0x1FF:FFFF (32KB)
GamePakSRAM	0x0EXX:XXXX	0xFFF:FFFF (64KB)

# Vídeo

## Vídeo: Funcionamento

Vimos que o GBA possui uma tela com resolução 240x160 pixels.

Porém, isso varia do ***modo de vídeo*** (Video Mode).

No GBA há 3 modos ***Bitmapeados*** (enumerados como 3, 4 e 5) e 3 modos ***Tilemapeados*** (enumerados como 0, 1 e 2). Por simplicidade, começaremos com modos Bitmapeados (especificamente o 3).

Modo	BGs	Resolução	bpp	RAM	Page-Flip
3	2	240x160	16	(1x) 12C00h	Não
4	2	240x160	8	(2x) 9600h	Sim
5	2	160x128	16	(2x) A000h	Sim

# Vídeo: Registrador de Controle do Visor

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
W	W	W	OB	B3	B2	B1	B0	F	0	H	P	C			Mode

Bit	Descrição
M (0-2)	Display-Mode.
C (3)	(Read-Only) Indica se o cartucho inserido é de GBC (1) ou GBA (0).
P (4)	Seleção de página.
H (5)	Habilita acesso à OAM quando em HBlank.
O (6)	Modo de mapeamento de objetos. 0 = 2D/Matricial; 1 = 1D/Sequencial.
F (7)	Força uma tela em branco.
B<X> (8-B)	Habilita Background <X>.
OB (C)	Habilita camada de objetos.
W (D-F)	Habilita o uso das janelas 0/1/de objetos, respectivamente. Janelas podem ser usadas como máscaras (como foi feito com o lampião em Zelda).

# Vídeo: Registrador de Controle do Visor

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
W	W	W	OB	B3	B2	B1	B0	F	0	H	P	C	Mode		

Bit	Descrição
M (0-2)	Display-Mode.
C (3)	(Read-Only) Indica se o cartucho inserido é de GBC (1) ou GBA (0).
P (4)	Seleção de página.
H (5)	Habilita acesso à OAM quando em HBlank.
O (6)	Modo de mapeamento de objetos. 0 = 2D/Matricial; 1 = 1D/Sequencial.
F (7)	Força uma tela em branco.
B<X> (8-B)	Habilita Background <X>.
OB (C)	Habilita camada de objetos.
W (D-F)	Habilita o uso das janelas 0/1/de objetos, respectivamente. Janelas podem ser usadas como máscaras (como foi feito com o lampião em Zelda).

## Vídeo: Registrador de Controle do Visor

O **Mode 3** usa o BG2 para renderizar, então precisamos habilitar o bit do Mode 3. É necessário também habilitar o próprio **Mode 3** em si. Para isso, o campo representado pelos bits 0, 1 e 2 precisa conter o valor "3", que em binário é representado por 0b11.

...	B	A	9	8	7	6	5	4	3	2	1	0
...	B3	B2	B1	B0	F	0	H	P	C	Mode		
...	0	1	0	0	0	0	0	0	0	0	1	1
...	$2^{11}$	$2^{10}$	$2^9$	$2^8$	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$

# Vídeo: Registrador de Controle do Visor

...	B	A	9	8	7	6	5	4	3	2	1	0
...	B3	B2	B1	B0	F	0	H	P	C	Mode		
...	0	1	0	0	0	0	0	0	0	0	1	1
...	$2^{11}$	$2^{10}$	$2^9$	$2^8$	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$

Com isso, temos:

$$2^{10} + 2^1 + 2^0 = 1024 + 2 + 1 \quad (1)$$

$$1027 = 0x403 \quad (2)$$

Representando a alteração do registrador de controle do visor em pseudo-código:

```
memory[0x4000000] = 0x403;
```

## Vídeo: VRAM

A memória de Vídeo **pode ser entendida** como uma matriz de bytes.

Seu intervalo de endereço é de 0x6000000 a 0x6017FFF.

Os bytes da VRAM são interpretados de forma diferente conforme o **Display-Mode** selecionado.

No Mode 3, por exemplo, a VRAM é organizada como um conjunto de cores de 16 bits, montando uma "matriz" de pixels 240x160 (totalizando 76800B = 75KB).

# Vídeo: VRAM

Os 16 bits de cores são organizados da forma:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
-	B				G				R						

Portanto, para representar a cor azul puro:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
-	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
-	7			C				0				0			

Assim, basta alterar o pixel desejado para o valor 0x7C00:

```
memory[0x6000000][x, y] = 0x7C00;
```

## Vídeo: VRAM

Se quisermos deixar o pixel (4, 8) em azul puro, portanto, a VRAM poderá ser vista da forma:

-	0	1	2	3	4	5	6	7	8	...	238	239
0	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
1	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
2	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
3	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
4	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
5	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
6	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
7	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
8	0000	0000	0000	0000	7C00	0000	0000	0000	0000	...	0000	0000
9	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
10	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
...	...	...	...	...	...	...	...	...	...	...	...	...
157	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
158	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
159	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000

## Vídeo: VRAM

Se quisermos deixar o pixel (4, 8) em azul puro, portanto, a VRAM poderá ser vista da forma:

x/y	0	1	2	3	4	5	6	7	8	...	238	239
0	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
1	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
2	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
3	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
4	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
5	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
6	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
7	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
8	0000	0000	0000	0000	7C00	0000	0000	0000	0000	...	0000	0000
9	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
10	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
...	...	...	...	...	...	...	...	...	...	...	...	...
157	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
158	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
159	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000

# Vídeo: VRAM

Alterando o pixel em (4, 4) para 0:00000:00000:11111b (31):

x/y	0	1	2	3	4	5	6	7	8	...	238	239
0	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
1	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
2	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
3	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
4	0000	0000	0000	0000	001F	0000	0000	0000	0000	...	0000	0000
5	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
6	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
7	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
8	0000	0000	0000	0000	7C00	0000	0000	0000	0000	...	0000	0000
9	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
10	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
...	...	...	...	...	...	...	...	...	...	...	...	...
157	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
158	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
159	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000

# Vídeo: VRAM

Alterando o pixel em (4, 6) para 0 : 00000 : 11111 : 00000b (992):

x/y	0	1	2	3	4	5	6	7	8	...	238	239
0	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
1	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
2	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
3	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
4	0000	0000	0000	0000	001F	0000	0000	0000	0000	...	0000	0000
5	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
6	0000	0000	0000	0000	03E0	0000	0000	0000	0000	...	0000	0000
7	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
8	0000	0000	0000	0000	7C00	0000	0000	0000	0000	...	0000	0000
9	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
10	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
...	...	...	...	...	...	...	...	...	...	...	...	...
157	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
158	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
159	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000

# Vídeo: VRAM

Alterando o pixel em (4, 7) para 0:11111:11111:00000b (32736):

x/y	0	1	2	3	4	5	6	7	8	...	238	239
0	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
1	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
2	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
3	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
4	0000	0000	0000	0000	001F	0000	0000	0000	0000	...	0000	0000
5	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
6	0000	0000	0000	0000	03E0	0000	0000	0000	0000	...	0000	0000
7	0000	0000	0000	0000	7FE0	0000	0000	0000	0000	...	0000	0000
8	0000	0000	0000	0000	7C00	0000	0000	0000	0000	...	0000	0000
9	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
10	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
...	...	...	...	...	...	...	...	...	...	...	...	...
157	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
158	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
159	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000

# Vídeo: VRAM

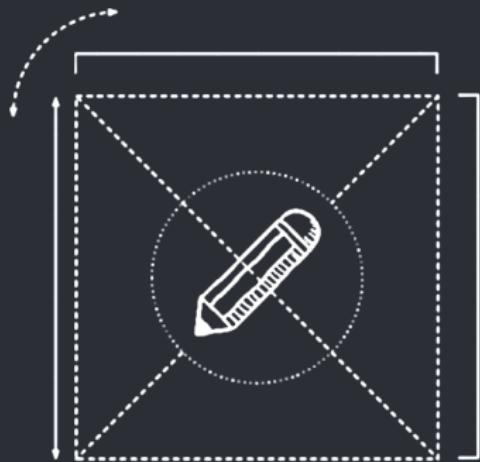
Alterando o pixel em (4, 5) para 0:00000:11111:11111b (1023):

x/y	0	1	2	3	4	5	6	7	8	...	238	239
0	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
1	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
2	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
3	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
4	0000	0000	0000	0000	001F	0000	0000	0000	0000	...	0000	0000
5	0000	0000	0000	0000	03FF	0000	0000	0000	0000	...	0000	0000
6	0000	0000	0000	0000	03E0	0000	0000	0000	0000	...	0000	0000
7	0000	0000	0000	0000	7FE0	0000	0000	0000	0000	...	0000	0000
8	0000	0000	0000	0000	7C00	0000	0000	0000	0000	...	0000	0000
9	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
10	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
...	...	...	...	...	...	...	...	...	...	...	...	...
157	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
158	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
159	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000

# Vídeo: VRAM

Alterando o pixel em (2, 6) para 0:11111:11111:11111b (32767):

x/y	0	1	2	3	4	5	6	7	8	...	238	239
0	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
1	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
2	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
3	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
4	0000	0000	0000	0000	001F	0000	0000	0000	0000	...	0000	0000
5	0000	0000	0000	0000	03FF	0000	0000	0000	0000	...	0000	0000
6	0000	0000	7FFF	0000	03E0	0000	0000	0000	0000	...	0000	0000
7	0000	0000	0000	0000	7FE0	0000	0000	0000	0000	...	0000	0000
8	0000	0000	0000	0000	7C00	0000	0000	0000	0000	...	0000	0000
9	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
10	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
...	...	...	...	...	...	...	...	...	...	...	...	...
157	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
158	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000
159	0000	0000	0000	0000	0000	0000	0000	0000	0000	...	0000	0000



# Mãos à obra

Introdução ao Assembly ARMv4

## Mãos à obra: Introdução ao Assembly ARMv4

Na arquitetura ARM anterior à ARMv8, toda instrução é também uma comparação. Há vários modos de comparação nas instruções que indicam quando ela será executada ou não, por simplicidade manteremos AL (*Always*) como condição para as instruções executarem ou não.

**OBS:** ARM7TDMI é a versão do processador. ARMv4 é a versão do conjunto de instruções (*ISA: Instruction Set Architecture*).

# Mãos à obra: Introdução ao Assembly ARMv4

Os processadores ARM7 podem rodar em 2 modos:

- ARM (instruções de 32 bits);
- Thumb-2 (instruções de 16 bits).

Utilizaremos majoritariamente o modo Thumb-2 por duas razões:

1. Instruções Thumb-2 ocupam metade do espaço no cartucho em relação às instruções ARM, e como precisaremos de várias instruções, não queremos que o cartucho acabe ficando muito pesado;
2. O barramento do GBA é de 16 bits, portanto rodar instruções de 32 bits acabam ficando mais lento.

## Mãos à obra: Registradores - Convenções de uso

Registrador	Função
r0-12	Propósito geral
r13	<i>Stack Pointer</i> (sp)
r14	<i>Link Register</i> * (lr)
r15	<i>Program Counter</i> (pc)

Função	Registradores
Valor de retorno de função	r0-1
Parâmetros de função	r0-3
Parâmetros adicionais	sp
Registradores de rascunho	r0-3, r12
Registradores preservados	r4-13

\*Utilizado para o endereço de retorno de uma função.

# Mãos à obra: Registradores

Processador	
r0	0x00000000
r1	0xFFFFFFFF
r2	0x00000000
r3	0x00000000
r4	0x00000000
r5	0x00000000
...	...
r15	0x08000004

Memória	
0x00000000	0x0000
0x00000002	0x0000
...	...
0x02000000	0x1001
0x02000002	0x0001
0x02000004	0x0018
0x02000006	0xAAA0
...	...
0x03000000	0xA011
0x03000002	0x00FF
0x03000004	0x09B0
...	...
0x04000000	0x0000
...	...
0x01FFFFFF	0x0000

# Mãos à obra: Instruções ARMv4

## **MOV** (Move):

Copia um dado para o registrador de destino.

```
MOV reg.destino #const.8bits
```

```
MOV reg.destino reg.origem
```

## **STRH** (*Store Register Halfword*):

Armazena um dado em um registrador de meia-palavra\*.

```
STRH reg.origem [reg.destino, reg.offset]
```

\*Palavra ("Word"): Tamanho da instrução. ARMv4 toma por base que cada instrução tenha 32 bits. Halfword portanto significa 16 bits neste contexto.

# Mãos à obra: Instruções ARMv4

**ADD** (Somar):

Soma uma constante e o valor de um registrador.

`ADD reg.destino reg.entrada #const.8bits`

**LDR** (*Load Register* - Pseudo-instrução):

Armazena uma constante a um registrador

(diferenças da instrução MOV serão explicadas adiante).

`LDR reg.destino #const.32bits`

## Mãos à obra: Exemplo de execução de instrução

```
mov r0, #0x04000000 ; r0 ← 0x40000000  
ldr r1, =#0x403      ; r1 ← 0x403  
strh r1, [r0]         ; memory[r0] ← r1
```

Processador		Memória	
r0	0x00000000	0x00000000	0x0000
r1	0x00000000	0x00000002	0x0000
r2	0x00000000	...	...
r3	0x00000000	0x03000000	0xA011
r4	0x00000000	0x03000002	0x00FF
r5	0x00000000	0x03000004	0x09B0
...	...	...	...
r15	0x08000004	0x04000000	0x0000
		...	...
		0x01FFFFFF	0x0000

## Mãos à obra: Exemplo de execução de instrução

```
mov r0, #0x04000000 ; r0 ← 0x40000000
```

```
ldr r1, =#0x403      ; r1 ← 0x403
```

```
strh r1, [r0]        ; memory[r0] ← r1
```

Processador		Memória	
r0	0x04000000	0x00000000	0x0000
r1	0x00000000	0x00000002	0x0000
r2	0x00000000	...	...
r3	0x00000000	0x03000000	0xA011
r4	0x00000000	0x03000002	0x00FF
r5	0x00000000	0x03000004	0x09B0
...	...	...	...
r15	0x08000004	0x04000000	0x0000
		...	...
		0x01FFFFFF	0x0000

## Mãos à obra: Exemplo de execução de instrução

```
mov r0, #0x04000000 ; r0 ← 0x40000000
```

```
ldr r1, =#0x403      ; r1 ← 0x403
```

```
strh r1, [r0]         ; memory[r0] ← r1
```

Processador		Memória	
r0	0x04000000	0x00000000	0x0000
r1	0x000000403	0x00000002	0x0000
r2	0x000000000	...	...
r3	0x000000000	0x03000000	0xA011
r4	0x000000000	0x03000002	0x00FF
r5	0x000000000	0x03000004	0x09B0
...	...	...	...
r15	0x08000006	0x04000000	0x0000
		...	...
		0x01FFFFFF	0x0000

## Mãos à obra: Exemplo de execução de instrução

```
mov r0, #0x04000000 ; r0 ← 0x40000000  
ldr r1, =#0x403      ; r1 ← 0x403  
strh r1, [r0]         ; memory[r0] ← r1
```

Processador		Memória	
r0	0x04000000	0x00000000	0x0000
r1	0x000000403	0x00000002	0x0000
r2	0x000000000	...	...
r3	0x000000000	0x03000000	0xA011
r4	0x000000000	0x03000002	0x00FF
r5	0x000000000	0x03000004	0x09B0
...	...	...	...
r15	0x08000008	0x04000000	0x0403
		...	...
		0x01FFFFFF	0x0000

## Exemplo de código

O código abaixo troca o *Display-Mode* para 3 e pinta o pixel nas coordenadas (0, 0) de Amarelo-escuro (BGR 0, 15, 15):

```
main:  
    ldr r0, =0x04000000  
    ldr r1, =0x403  
    strh r1, [r0]  
    ldr r0, =0x60000000  
    ldr r1, =0x1EF  
    strh r1, [r0]
```

Iremos supor que o código está salvo em um arquivo chamado `asm_example.s`.

## **DevKitPro**

Como estamos em uma arquitetura diferente, precisamos de uma ferramenta (mais especificamente, uma **Toolchain**) para o compilador que saiba o esquema de memória adequado. Para GBA, a ferramenta que utilizaremos é **DevKitPro** e seu submódulo **DevKitARM**.

# DevKitPro- Gerar o arquivo do jogo

## 1. Compilar e gerar um arquivo-objeto:

```
arm-none-eabi-as -mthumb -mthumb-interwork -c asm_example.s
```

## 2. Gerar um arquivo .elf (Executable and Linkable File):

```
arm-none-eabi-as -mthumb -mthumb-interwork -specs=gba.specs asm_example.o -o asm_example.elf
```

## 3. Tradução/Limpeza para um executável puro:

```
arm-none-eabi-objcopy -O binary asm_example.elf asm_example.gba
```

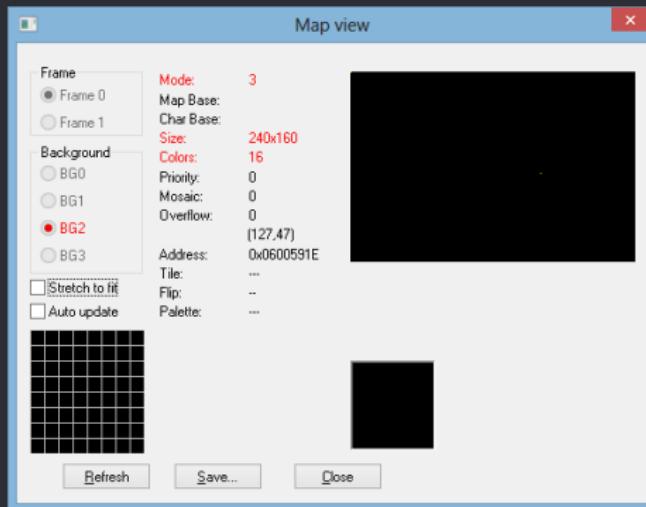
OBS: a etapa acima é necessária apenas para limpar os símbolos de debug que vêm no .elf.

## 4. Consertar Header:

```
gbafix asm_example.gba
```

Todo jogo de GBA possui um Header que checka se o binário é um GBA válido. O gbafix trata de ajeitar o header para que seja válido.

# Testando o jogo



# Extras

# DevKitPro- Instalação

## Windows:

1. Baixar a versão mais atualizada do DevKitProUpdater:  
<http://sourceforge.net/projects/devkitpro/files/Automated%20Installer/>
2. Executar o instalador. É possível customizar a instalar somente o DevKitARM e ainda retirar os exemplos, se necessário.

## Linux:

Seguir o guia:

[https://docs.google.com/document/d/1ieTi\\_zARtRsm93u6QDUKRLEDU7JGLYOKGSNuII0d02s/pub](https://docs.google.com/document/d/1ieTi_zARtRsm93u6QDUKRLEDU7JGLYOKGSNuII0d02s/pub)

# VisualBoyAdvance



# VisualBoyAdvance: Recursos

- Emulação rápida;
- Leve e de fácil configuração;
- Ferramentas para desenvolvedor (não disponível na versão VBA-M):
  - Disassemble;
  - Logging;
  - IO Viewer;
  - Map Viewer;
  - Palette Viewer;
  - Memory Viewer;
  - OAM Viewer;
  - Tile Viewer;
  - Habilitar/Desabilitar Layers.

# VisualBoyAdvance: Disassemble

The screenshot shows the VisualBoyAdvance disassembly window titled "Disassemble". The assembly code pane displays the following instructions:

```
0300128c e3ccc8FF bic r12, r12, #0xFF0000
03001290 e08c7467 add r7, r12, r7, ror #0x08
03001294 e0899004 add r9, r9, r4
03001298 e1b0eba9 mous lr, r9, lsr #0x17
0300129c 0a000007 beq $030012c0
030012a0 e3c995fe bic r9, r9, #0x3F800000
030012a4 e052200e subs r2, r2, lr
030012a8 daffffcF ble $030001ec
030012ac e25ee001 subs lr, lr, #0x1
030012b0 00800001 addeq r0, r0, r1
030012b4 11b300d4 ldrnesb r0, [r3, lr]!
030012b8 e1f319d1 ldrsb r1, [r3, #0x1]!
030012bc e0411000 sub r1, r1, r0
030012c0 e2955101 adds r5, r5, #0x40000000
030012c4 3affffea bcc $03001274
030012c8 e5857630 str r7, [r5, #0x630]
030012cc e4856004 str r6, [r5], #0x4
```

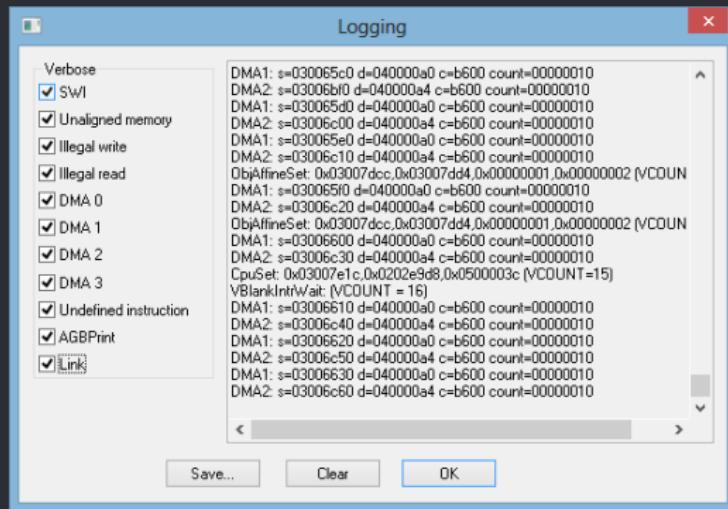
The "Registers" pane on the right shows the current state of the CPU registers:

R0:	00000000
R1:	FFFFFFFFFF
R2:	00000092
R3:	0848c40F
R4:	0072083d
R5:	43006744
R6:	F8F6fcfb
R7:	FaF800Fe
R8:	0000003c
R9:	000F0d6e
R10:	00040000
R11:	00010000
R12:	FF000000
R13:	03007dd4
R14:	00000001
R15:	030012a0
R16:	0000001F

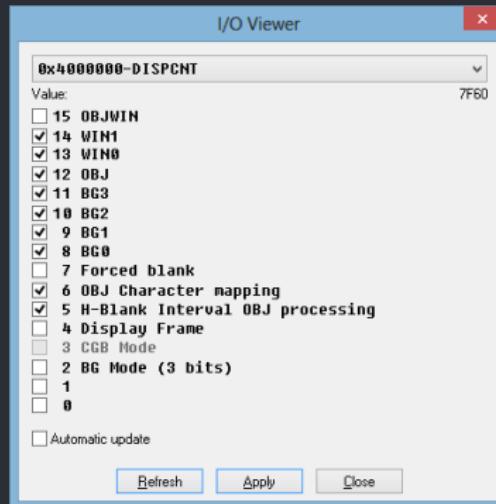
Below the registers are checkboxes for N, Z, C, T, and V, and a "Mode" field set to "1F".

At the bottom of the window are buttons for "Automatic update", "Goto R15", "Refresh", "Next", and "Close".

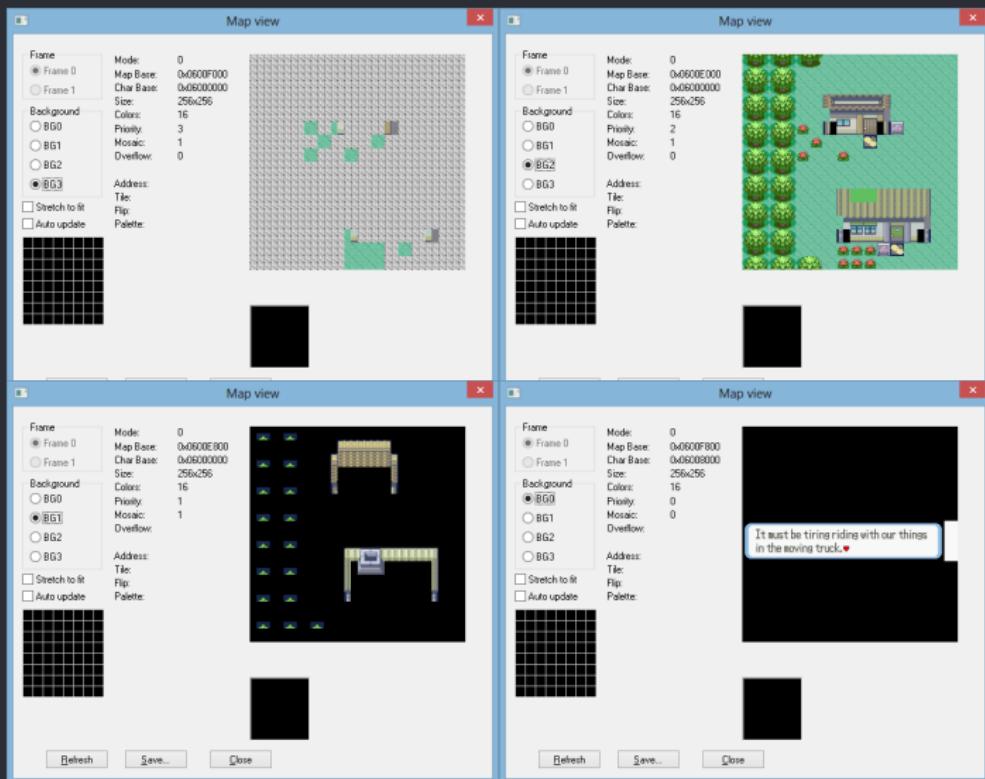
# VisualBoyAdvance: Logging



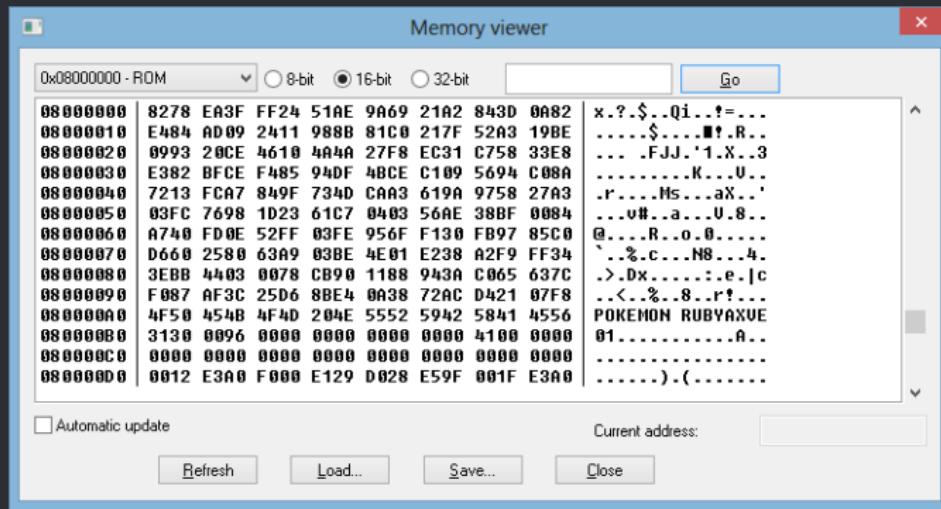
# VisualBoyAdvance: IO Viewer



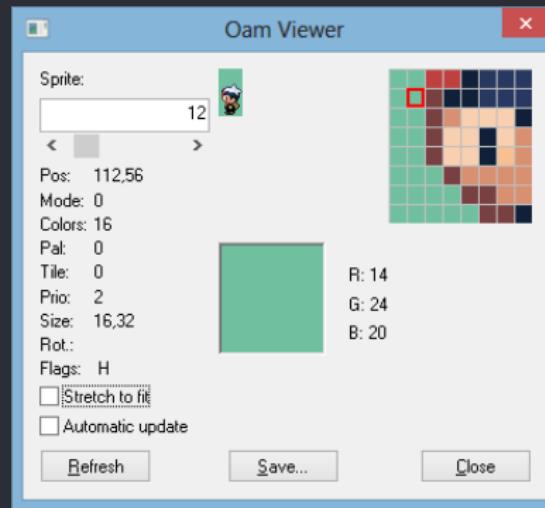
# VisualBoyAdvance: Map Viewer



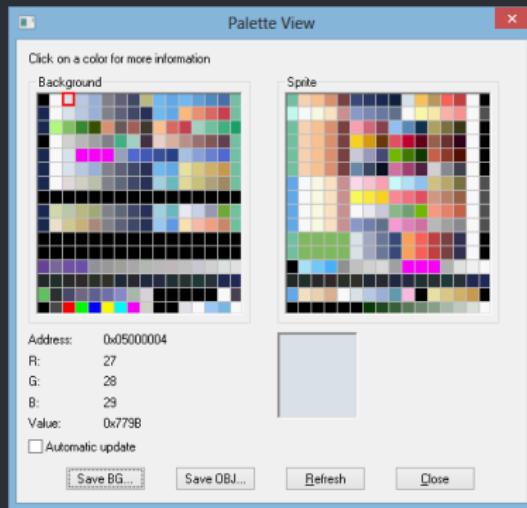
# VisualBoyAdvance: Memory Viewer



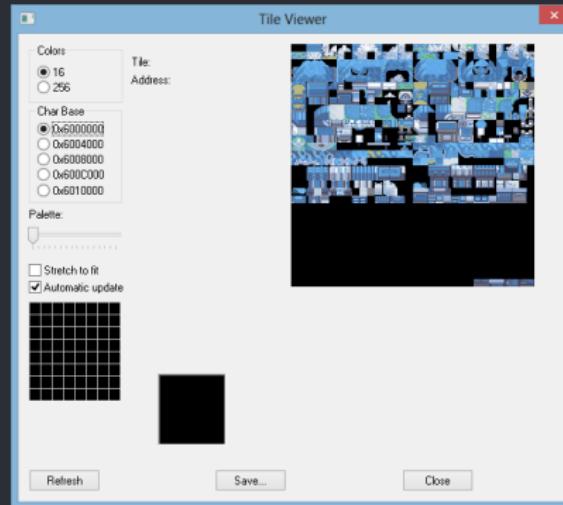
# VisualBoyAdvance: OAM Viewer



# VisualBoyAdvance: Palette Viewer



# VisualBoyAdvance: Tile Viewer



## Referências/Bibliografia I

- [1] “ARM Overview - Registers - OSDev.”  
[http://wiki.osdev.org/ARM\\_Overview#Registers](http://wiki.osdev.org/ARM_Overview#Registers).
- [2] “GBADev/Docs.”  
<http://www.gbadev.org/docs.php>.
- [3] “TONC.”  
<http://www.coranac.com/tonc/>.
- [4] “Wikipédia (Lista de Paletas de Videogames).”  
[https://en.wikipedia.org/wiki/List\\_of\\_video\\_game\\_console\\_palettes](https://en.wikipedia.org/wiki/List_of_video_game_console_palettes).
- [5] “Wikipédia (GBA).”  
[https://en.wikipedia.org/wiki/Game\\_Boy\\_Advance](https://en.wikipedia.org/wiki/Game_Boy_Advance).

## Referências/Bibliografia II

- [6] "Alex Schütz (Sketch do circuito interno do GBA)." <http://assemblergames.com/l/threads/game-boy-advance-schematic-cadsoft-eagle-files.54814/>.
- [7] "ProblemKaputt." <http://problemkaputt.de/gbatek.htm>.
- [8] "SimpleMachines (ARM)." [http://simplemachines.it/doc/arm\\_inst.pdf](http://simplemachines.it/doc/arm_inst.pdf).
- [9] "InfoCenter.ARM." <http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0497a/BABJGHFJ.html>.

## Referências/Bibliografia III

- [10] “uTexas.EDU.”  
[http://users.ece.utexas.edu/~valvano/EE345M/Arm\\_EE382N\\_4.pdf](http://users.ece.utexas.edu/~valvano/EE345M/Arm_EE382N_4.pdf).

Obrigado!

Perguntas?