1.前言

这个文档是为了帮助你在 Game Boy™ Classic，Game Boy™ Pocket，Super Game Boy™ 和 Game Boy™ Color（注：GBC 有其特有的 CPU 指令，附在附录之中） 上编程而撰写的。这是一本可以用来立刻开始为这种硬件编程的完全手册。由三个主要部分组成。

第一部分是‘GBSpec.txt’(GB 硬件规格)，（这里是作者名字），这将在第一段出现。

第二部分是几篇文档的混编，它包括了汇编指令的操作码及其执行时间和影响的标志位。

第三部分是为了在 GBC 上编程的其特性和指令的摘要。

关于如何使你的模拟器支持运行真正的 gameboy 游戏的部分可以在附录中找到，并且在 classic GB 总线上进行一般性的读写操作的时序图也附在其中

最后一页则包括了用于快速查找汇编指令的页码索引。

Have gun !

动态规划先生

被 REALDaGong 翻译，顺便赞美款爷，他可好了。

2 硬件规格

2.1. 前言：

这里只是免责声明，还有一个巨长的复合句，不翻译了

2.2. 发展时期：

GB=Original GameBoy（Gameboy Classic）

GBP=GameBoy Pocket/GameBoy Light

GBC=GameBoy Color

SGB=Super GameBoy

2.3.GameBoy 硬件规格

CPU：8 位，16 位总线，与 Z80 处理器相似
主存：8K Byte
显存：8K Byte
屏幕尺寸：2.6 英寸
分辨率：160x144（20x18 个区块）
最大精灵数:40(sprites 翻译成精灵？我可去您的妈吧)
最大精灵/每扫描线 数：10
最大精灵尺寸：8x16
最小精灵尺寸：8x8
CPU 时钟速度：4.194304 MHz
4.295454 SGB，4.194/8.388 GBC
水平刷新：9198KHz
垂直刷新：59.73Hz
声音：四声道立体声(!?)
电源：朱宏明

任天堂在其文档中描述机器指令时的时间单位是机器周期，而这篇文档则用时钟周期来描述。
1 机器周期=4 时钟周期

2.4.处理器

GameBoy 使用一块与 Intel 8080 相似的处理器芯片，它包括除了交换指令之外的所有 8080 指令。在许多方面上这个处理器更像 Zilog Z80 处理器，与 Z80 相比，这里新增了一些指令并且移除了一些指令。
新增的:
自己看，懒得打，第七页，不过也没用

2.5. 存储器映射

2.5.1. 通用存储器映射

开启中断寄存器

------------------FFFF

内部高速 RAM

------------------FF80

不可用区域

------------------FF4C

I/O 区域内存

------------------FF00

空但是不可用作 I/O

------------------FEA0

精灵属性存储器(OAM)

------------------FE00

8kB 内部高速 RAM 镜像(原文 Echo of 8kb Internal RAM)

------------------E000

8kB 内部高速 RAM

------------------C000

8kB 可切换 RAM 页面

------------------A000

8kB 显存

------------------8000--

16kB 可切换 ROM 页面 |

------------------4000 |=32kB 卡带

16kB ROM 页面 #0 |

------------------0000--

译注：建议了解有关虚拟内存的知识，可以去翻深入了解计算机系统。 [斜眼笑]

2.5.2.8kB 内部高速 RAM 镜像

E000-FE00 区域(译注：我推测不含 FE00？)地址似乎与 C000-DE00 访问同一块高速 RAM。（例：如果你向 E000 写入一个字节它将会出现在 C000 和 E000，反之同理。）

2.5.3. 用户 I/O

在存储器映射区中没有实现用户数据输入接口(指加载游戏存档)的空余区域，除了可切换 RAM 页面区域(在 Super Smart Card 上不可行，因为它的 RAM 区域总是处于启用状态)。一个只能输出的接口可能会被映射在 A000-FDFF 之间的任何位置。如果被映射在了一个 RAM 区域则需要注意确保使用一块不被用作它用的区域。FE00 和更高的区域不能被使用，因为 CPU 并不能在这些区域上向外部存储器输出。(译注：这里可能是在解释为什么要将外部存储映射安排到这一个位置上，可能是工艺问题？)

如果你有一个卡带，它有 MBC1，一个 1MB 或更小的 ROM 和一个 8Kbyte 或更小的 RAM，这样你就可以使用 MBC1 的引脚 6 和 7 作为两个输出引脚，为了使用它们你必须首先向 6000 写入 01 来将 MBC1 置于 4MbitROM/32KbyteRAM 模式，向 4000 中写入至少两个有效位来作为这些引脚的输出。

2.5.4.预留的内存区域

0000 , 0008，0010，0018，0020，0028，0030，0038
Restart 00 Address RST 调用这些地址
0040 垂直空白中断的起始地址
0048 LCDC 状态中断的起始地址
0050 Timer 溢出中断的起始地址
0058
。。。。

2.6.卡带类型

下面详细定义了卡带 0147 位置字节的意义
（自己参照把）

2.7.特殊程式

2.7.1.开机序列

当 Gameboy 电源打开时，一个从存储器 0 号位置开始的 256 字节的程序将被执行，这个程序位于 gameboy 内部的一个 ROM 里。程序做的第一件事是读取卡带区域的 104 到 133,并将这里存储的任天堂 logo 图像放在屏幕顶部。这个图片之后一直滚动到屏幕中央，然后内置扬声器会播放两个音符。之后卡带区域的 104 到 133 将再次被读取，但是这一次将用来与内置 ROM 中的一个表进行比对，如果有任何字节比对失败，gameboy 就停止比对并且停止所有操作。

GB

下一步，Gameboy 开始将卡带中从 134 到 14d 的所有字节加起来，再在总和上加上十进制的 25 。如果结果的最后一个标志字节不是 0，Gameboy 将会停止所有操作。（译注：理论上，但是这一事件事实上不会发生。）

SuperGB

。。。

如果上文的检查通过则内部 ROM 将被停用，卡带的程序将于位置 100 开始执行，同时寄存器具有以下的值。。。。。。

假设上方所提及的值将一直存在不是个好主意，更新版本的 gameboy 可能在重启之后有与它们不同的值。应当总把这些寄存器都清空，而不是假设它们总与上面的值一致。

请注意 gmaeboy 的内部 RAM 在开机时含有随机的数据，所有的模拟器都倾向于在进入时把所有的 RAM 设为 00。

卡带 RAM 在实体 gameboy 上被第一次读取时包含随机

数据。它将只在 Gameboy 代码将它初始化之后才包含有意义的数据。

2.7.2.停止模式

STOP 命令会中止 Gameboy 处理器和屏幕直到任何一个键被按下。GB 和 GBP 的屏幕会变白，并有一条水平暗线，GBC 的屏幕会变黑。

2.7.3.低电量模式

推荐在任何可能能够减少电量消耗和延长电池寿命的时候使用 HALT 指令。这条指令将停止系统时钟，减少 CPU 和 ROM 的电量消耗。

CPU 将会保持挂起状态直到一个已启用中断在其相应的生效点触发(即使 CPU 的中断处理处于关闭状态)。然后紧接着 HALT 的指令将被执行。如果中断被禁用（DI）那么 halt 将不会进行挂起操作，但是它的确会导致 GB，GBP 和 SGB 上的程序计数器停止一个指令的计数，就像下面所提到的。

取决于一个游戏需要多少的 CPU 时间,HALT 指令可以延长电池 5-50%甚至更多的寿命。

警告：紧接着 HALT 的指令在 GB，GBP 和 SGB 上中断被禁用时将被“跳过(skipped)”。因此，应当总在 HALT 后写一条 NOP。跳过指令的情况将不会在中断开启的时候（EI）发生。

这个“跳过(skipping)”似乎在 GBC 上不会触发，即使在普通 GB 模式下。（$143=00）

例 从记录了这个问题的马 K 方先生：

（假设在所有的例子中中断都被禁用）

1 这段代码使 a 寄存器被自增**两次**

```
76   halt
3C   inc a
```

2 下一个例子有点难了，下面的代码

```
76   halt
FA 34 12       ld a,(1234)
```

(注：(立即数)代表 立即数 号内存，同时你应该注意到了 GB 的 CPU 是小端的。)

在效果上相当于执行

```
76   halt
FA FA 34       ld a,(34FA)
12   ld (de),a
```

3 最后是一个有趣的副作用

```
76   halt
76   halt
```

这个组合会把 CPU 当场吊死。

第一个 HALT 会导致第二个 HALT 重复，因此造成下一条（他自己）被重复--一次又一次。在两条之间放一条 NOP 会使得 NOP 被重复一次，第二个 HALT 就不会锁定 CPU。

下面是 GB 程序用于等到垂直空白中断的推荐代码

```
;游戏主循环
Main:
     halt      ;停止系统时钟，在中断后从 halt 返
     nop       ;回（参照上方的警告）

     ld a,(VblnkFlag)
     or a      ;有垂直空白中断?
     Jr z,Main ;不，一些其他中断

     xor a
     ld (VblnkFlag),a ;清除垂直空白标志

     call Controls ;键盘输入
     call Game     ;game operation

     jr Main
;垂直空白中断例程
Vblnk:
     push af
     push bc
     push de
     push hl
```

```
call SpriteDma     ;精灵更新

ld a,1
ld (VblnkFlag),a

pop hl
pop de
pop bc
pop af
reti
```

2.8.视频

2.8.1.区块

Gameboy 的主显示缓冲区（背景）由 256X256 像素，或是 32X32 区块组成(每个是 8X8 像素)。只有 160X144 像素可以显示在屏幕上。SCROLLX 和 SCROLLY 寄存器保存着背景中正在被显示在屏幕左上角的坐标。背景“环绕”着屏幕(也就是说当背景的一块离开了屏幕，它将出现在对侧。）

一块 VRAM 区域，称作背景区块映射区，含有所有显示出来的区块的标号。它被组织为 32 列，每列有 32 个字节，每个字节都包括一个被显示的区块的标号，区块的图案从位于 8000-8FFF 或 8800-97FF 的区块数据表中取得。在第一种情况下，图案被从 0 到 255 的无符号整数编号（例 图案 0#位于地址 8000）。在第二种情况下，图案被从-128 到+127 的有符号整数编号。（例 图案 0#位于地址 9000）背景的区块数据表的地址可以通过 LCDC 寄存器来选择。

除了背景，有着另外一个“窗口”位于背景的上方，窗口是不可滚动的，也就是说它将总是从它的左上角被显示。窗口的位置可以通过 WNDPOSX 和 WNDPOSY 寄存器进行调节，窗口左上角的屏幕坐标是 WNDPOSX-7，WNDPOSY。窗口的区块标号将被储存在区块数据表中，其中所有的颜色都不会是透明的(指其下的背景会被完全覆盖)。背景和窗口共享同一个区块数据表。

背景和窗口都可以通过设置 LCDC 寄存器中的位来分

别启用或禁用。

如果窗口处于使用状态并且一个扫描线中断把它禁用了（或者被写入 LCDC 来禁用，或者 WX 被设置大于 166）之后不久的一个扫描线中断又将它启用，这之后窗口将会继续显示在消失在屏幕上之前的同一位置。This way, even if there are only

16 lines of useful graphics in the window, you could

display the first 8 lines at the top of the screen and

the next 8 lines at the bottom if you wanted to do so. p23

WX 可能在扫描线中断的过程中被改变(将会导致图像扭曲效果或者禁用窗口（WX>166））但是对 WY 的改变不是动态的，并且在下一次屏幕重绘之前不会被注意到。

区块图片储存在区块数据表里（原文中是 Tile Pattern Tables，上文却一直在用 Tile Data Tables，这两个是同一个东西）。每个 8x8 的图片占用 16 字节，每 2 个字节代表一行：

```
Tile：                                  Image:
.33333..                    .33333..  01111100->7C
22...22.<-数码代表                    01111100->7C
11...11.   颜色码           22...22.  00000000->00
2222222.                              11000110->C6
33...33.                    11...11.  11000110->C6
22...22.                              00000000->00
11...11.                    2222222.  .....
........
```

就像之前所说，有两个区块数据表，位于 8000-8FFF 和 8800-97FF。第一个可以用作精灵，背景和窗口的显示，它的区块被 0 到 255 标号，第二个表只可以被用作背景和窗口显示，它的区块被-128 到 127 标号。

2.8.2.精灵

gameboy 的视频控制器(Video controller)可以以

8x8 或 8x16 像素的尺寸显示最多 40 个精灵，由于硬件的限制，每条扫描线上最多只能显示 10 个精灵，精灵的图案有着与区块图案一样的格式，但是它们是从位于 8000-8FFF 的精灵数据表中取得，以无符号整数标记。精灵的状态驻留在精灵状态表中（OAM-Object Attribute Memory），位于 FE00-FE9F。OAM 被分为 40 个 4 字节的块，每一块都对应一个精灵。在 8x16 精灵的模式下，精灵图案的最低一个有效位将被忽视，总是当做 0。（译注：也就是说，**每个** 8x16 精灵的所使用的图像位置编号只能是 0,2,4,6..... 假如是 0，你该从上到下显示显存中的 0 号块和 1 号块。）

当有不同 x 坐标的精灵重叠时，有更小的 x 坐标的（最靠近左边的）将会拥有优先权，会显示在其他精灵的上方。

当有相同 x 坐标的精灵重叠时，他们的优先级遵循精灵状态表的顺序。FE00 是最高的。

请注意 SpriteX=0 SpriteY=0 的精灵会被隐藏，为了显示一个精灵，注意以下的公式：

屏幕上的 X=SpriteX-8 Y=SpriteY-16 （左上角为原点）

举个例子，让精灵显示在左上角，就设置 Sprite X=8，Y=16.

在任何扫描线上只能显示十个精灵，当超出限制时，最低优先级的精灵将不会被显示出来。为了避免无用的精灵影响屏幕上的精灵可以将它们的 Y 设置为 0 或 144+16，如果只设置 X=0 或 160+8 将只会隐藏它，但是它仍然会影响同一条扫描线上的精灵。

OAM 块的格式

Byte0 Y 位置

Byte1 X 位置

Byte2 图像编号 0-255

Byte3 标志位

bit7 优先级

如果这一位被设置为 0，精灵将会显示在背景和窗口的顶部，如果被设置为 1，精灵会隐藏在编号为

1，2,3 的颜色后面，只显示在 0 号颜色的顶部。

（译注：为了防止你的理解错误，在此先提醒你一下，并不是“精灵只会透过白色”，gameboy 只有四种颜色，默认状况下，从 0-3 依次加深，但是这是可以被设置的，详情见调色板的有关解释，在设置后，0 号颜色也可以是黑色的，但你的精灵仍然可以“透过”它。同理，白色也可以不能被“透过”）

bit6 垂直翻转

设置为 1 时图像垂直翻转

bit5 水平翻转

同上。

bit4 调色板编号

设为 1 时精灵颜色会从 OBJ1PAL 中获取，设为 0 时从 OBJ0PAL 获取。

2.8.3.精灵 RAM 的 BUG

gameboy 的硬件缺陷使得如果运行下面使用 FE00-FEFF 范围内的数据的指令，OAM RAM 会被写入垃圾数据。

inc xx （xx=bc，de or hl）

dec xx

ldi a,(hl)

ldd a,(hl)

ldi (hl),a

ldd (hl),a

只有精灵 1 和 2（FE00，FE04）不会被这些指令影响。

2.9.声音

有两个声道与输出端 S01 和 S02 相连，也有一个输入端 Vin 链接着卡带，它可以连接到两个输出端的任何一个。gameboy 的周期任务(circuitry?我猜相当于“主循环”)可允许四种产生声音的方式：

带有扫描和包络函数的方波模式

只有包络函数的方波模式

来自波形 RAM 的自发波模式

带有包络函数的白噪音
（当场去世。）
这四种声音可以被独立控制并为每个输出端分别混合。
声音寄存器在生成声音时需要（原文是 may_be）一直被设置为 1
当设置了包络初值（再次去世）并重置了长度计数器时，要设置初始化标志为 1 并且初始化数据。
在接下来的情况下 Sound ON 标志将被重置并且声音输出停止：
1 在声音输出被长度计数器停止时。
2 在扫描函数在 sound1 下运行时触发了附加模式下的溢出。
当 sound3 的 Sound OFF 标志（NR30 的 bit7）被设为 0 时，OFF 模式的取消必须通过将 sound OFF 标志设置为 1 来完成。它将通过初始化 sound 3 来开始执行它的功能（？？？）
当 All Sound OFF 标志（NR52 的 bit7）被设为 0，sound1,2,3，4 的模式寄存器会重置并且声音输出会停止。（注：每一个声音的模式寄存器的设置应当在 All Sound OFF 模式取消后进行。在 All Sound OFF 模式下，每一个声音的模式寄存器无法被设置。）
注：All Sound OFF 模式下可省电了。
GB 频率寄存器和赫兹的对应关系：
gb=2048-（131072/Hz）
。。。。。
真 tm 难，有兴趣就了解一下，上 gbdevwiki.

2.10. Timer

(实现了这个功能就可以续了)
在需要进行定期更新或者精确更新的周期任务中定时产生中断的计时器会有很大用场。gameboy 中的计时器有 4096,16384,65536,262144 赫兹的可选频率，TIMA 会以选中频率自增，当它溢出时，将会产生一个中断。然后(确切的说，“然后”是再过 4 个时钟周期,比如一条 nop 之后)加载计时器模块（Timer

Modulo,TMA）中的内容。原文下面是例子：

定时器每秒中断 4096 次

```
ld a,-1
ld (FF06),a ;Set TMA to divide clock by 1
ld a,4
ld (FF07),a ;Set clock to 4096 Hertz
```

65536 次

```
ld a,-4
ld (FF06),a ;Set TMA to divide clock by 4
ld a,5
ld (FF07),a ;Set clock to 262144 Hertz
```

2.11.串行 I/0

gameboy 上的串行 io 端口的设置十分简单和粗糙，甚至没有启动和停止位，程序员在使用它们时可以脑洞大开。

在一次传输中，会同时移入和移出一个字节。移动的速度由当前使用的时钟源决定。如果是内部时钟，会以 8192hz（122 微秒）一 bit 的速度移动，最高有效位会被优先移入和移出。

当外部时钟(原文貌似笔误了)被选中时，它将会驱动一个在游戏链接端口(game link port,p31)上的时钟引脚。这个引脚在不使用的时候会保持高电平，在传输的时候会将会变成低电平 8 次来记录/读出每一位。

激活 FF02 的第七 bit 可以初始化串行传输，这一位在传输的过程结束后会被初始化为 0。如果串行中断被启用，在这一位设置之后，收到 8 个 bit 后会触发一次中断。

（在使用内部时钟时是 8 个时钟周期后，外部时钟时是 8 次低电平之后）

以外部时钟初始化串行输入时，如果不存在外部时钟，它将会永远等待下去。这允许一个特定数量的数量的串行端口保持同步。

被移出的最后一个位决定了输出线的状态(是否继续)。

如果使用内部时钟的串行传输被启用，并且外部没有gameboy与它链接，传输将会收到一个FF。
下面的代码使得75被移出串行端口并且向FF01移入一个字节。

```
ld a,75
ld FF01,a
ld a,81
ld FF02,a
```

2.12.中断

2.12.1.中断过程

IME（interrupt master enable）标志可以被DI指令重置，这样将会禁用所有终端。这个标志可以被EI设置，就可以得知所有IE寄存器设置的中断。
1.当中断产生时，IF标志会被设置。
2.如果IME和相应的IE标志被设置，下面的3步将会执行。
3.重置IME以禁用所有中断。
4.PC寄存器入栈
5.跳转到中断例程的起始地址
重置造成中断的IF，这一步由硬件自动完成。
(共消耗20clks)
在中断时，将正被使用的寄存器入栈的工作由中断例程执行。
一旦中断开始执行，所有的中断将被禁止，然而，如果IME和IE被人为控制，就可以通过内建的指令实行一系列的中断。
从中断例程中返回可以使用RETI或RET指令。
RETI指令返回时将开启中断。
RET则会延续中断的禁用状态，除非在之后执行了一次EI。中断将会在每次取指阶段前被获知。

2.12.2.中断描述

接下来的中断都只会在IE和IME设置时发生。
1 V-BLANK 垂直空白中断
垂直空白中断每秒触发约59.7次（fps=59.7），在

垂直空白的起始时被触发，在这一阶段中视频硬件不会使用显存，它可以被自由使用，这一阶段持续约1.1ms

2 LCDC Status LCDC 状态中断

这个中断发生的原因有很多，它们可被 FF40 STAT 寄存器描述。经常出现的原因是向用户表明视频硬件将要重绘一条 LCD 线。这个提醒在动态控制 SCX/SCY(FF43/FF42)寄存器来实现特殊视频效果时很有用。

3 Timer Overflow 计时器溢出

在 TIMA（FF05）从 FF 变到 00 时发生，（译注：准确的说，是在从 TIMA 从 TMA 中加载数据的同时发生）。

4 Serial Transfer Completion 串行传输完成

这个中断在一次游戏链接端口上（game link port）的串行传输完成时发生。

5 High-to-Low of P10-P13 P10-P13 高电平-低电平

在任何一条键盘输入线从高电平变到低电平时发生。因为实际上的键盘的“弹起”现象总会存在，软件应该知道这个中断在每个键按下和松开时可能会触发多次。

*弹起是任何按键在产生和断开连接都会有的副作用，在这些周期中，经常会有高低电平的波动发生。

2.13.特殊寄存器

2.13.1.IO 寄存器

1.FF00 (P1)

读入手柄信息并且决定系统种类（RW）

bit 7 空闲

bit 6 空闲

bit 5 P15 输出端（0=选中，意为我马上要检查的是方向键的按下和弹起状态）

bit 4 P14 输出端 (0=选中)

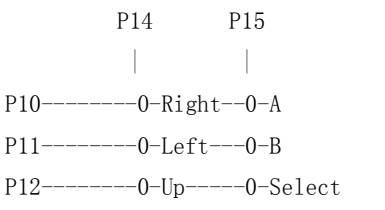
bit 3 P13 输入端

bit 2 P12 输入端

bit 1 P11 输入端

bit 0 P10 输入端

这是FF00的模型平面示意图



示例代码

游戏：吃豆人

地址 3b1

ld A,20	bit5 = 20
ld (FF00),A	设P14为**低电平**来**选中**它
ld A,(FF00)	
ld A,(FF00)	等待一小会
CPL	翻转A
AND 0F	取头4位
SWAP A	交换
LD B，A	暂存A
LD A,10	
LD (FF00),A	设P15为**低电平**来**选中**它
LD A,(FF00)	
LD A,(FF00)	
LD A,(FF00)	
LD A,(FF00)	
LD A,(FF00)	
LD A,(FF00)	多读，由于“弹起”效应
CPL	
AND 0F	取4位
OR B	把AB合取
LD B,A	把A存进B
LD A,（FF8B）	从ram中读入旧的手柄信息
XOR B	切换等待/激活(w/current)的键位
AND B	取回激活键位
LD A,B	把原值放回A

```
LD （FF8B）,A       将它作为旧手柄信息存储
LD A,30             取消选中 P14，P15
LD （FF00）,A       重置手柄
RET                 从子循环中返回
```

使用上方方法最后得到的键值:

```
80-start            8-down
40-select           4-up
20-B                2-left
10-A                1-right
```

如果我们按住 A,Start 和 Up,累加 A 的返回值是 94 (原文：the value returned in accumulator A would be 94.)

(译注：如果你不能理解，看我的解释：

整个输入控制芯片只有 6 个引脚，但你却有 8 个键，因此，选用 2 个引脚（最高的两个位）用来标注“激活”一列键，将来你从 FF00 读取的结果就是高四位+被“激活”的低四位。

0 代表了激活/按下，没有激活/按下的保持 1，未使用的最高两位永远是 1.

要注意的是，低四位和最高两位是不可写的。

假如你想得到方向键的状态，就写入 20，这时高四位应该是 D,这时你正在按着**右**，低四位就是 E,这时你读到的就是 DE。如果你按着**左**和**右**，低四位就是 C，你应该读到 DC，如果你在按着**左**的同时还按着 **A**，你会在低四位得到 D，但是“另一个低四位”会是 E，只有你写入 10 时，你才能够读到另一个低四位的数据。

在不需要读取时，会写入一个 30，即将两个列的选择线都设为 1，正常的 gameboy 游戏不会有写入 00(同时选择两个列)的情况的，如果有这种情况，是那个程序员/编译器脑残，不是你的锅。

你所要保证的就是你的模拟器完全实现了这一套规则，如果你没有完全实现/实现不正确，那么会出现一部分游戏能够响应按键，另一部分却不能响应的情况。)

2.FF01(SB)
串行输入的内容(RW)
3.FF02(SC)

SIO 控制(RW)
bit 7 传输开始标志
0：无传输
1：有传输
bit 0 移位时钟源
0：外部时钟（最大 500khz）
1:内部时钟（8192hz）
设置传输开始标志可以初始化传输，这一位会在传输结尾时被自动读取和设为 0。
串行数据的接收和传输同时结束，接受的数据会被自动存入 SB
4.FF04 （DIV）

分隔寄存器（RW）
这个寄存器每秒自增 16384 次，写入任何值都将其设为 00.
5.FF05（TIMA）

时间计数器（RW）
计时器以 TAC 寄存器指定的时钟频率自增，在溢出时产生中断。
6.FF06（TMA）

计时器模块（RW）
TIMA 溢出时，这里的数据会被加载。
7.FF07（TAC）

计时器控制（RW）
bit 2 计时器停止
0：停
1：开
bit 1+0 时钟选择

00 4096hz
01 262144
10 65536
11 16384

8.FF0F （IF）
中断标志（RW）
bit 4 HightoLow
bit 3 IO complete
bit 2 timer overflow
bit 1 LCDC（见 STAT）
bit 0 V-blank
优先级和跳转地址见下表：

中断	优先级	起始地址
V-Blank	1	0040
LCDC Status	2	0048-模式 0,1,2 LYC=LY coincide 都对应这里。(可能每条扫描线上只能触发一次，但是我不确定。)
Timer Overflow	3	0050
Serial Transfer	4	0058
Hi	5	0060

多个中断同时发生时只有最高优先级的可以被获知，
当中断进行时，在 IE 设置之前 IF 应该为 0

9.FF10 （NR10）
声音模式 1 寄存器（RW）
bit 6-4 扫频时间
bit 3 扫频增加/减少(sweep increase/decrease)
0:增加 频率上升
1:减少 频率下降
bit 2-0 扫频移位数（0-7）
扫频时间：
000 关闭 没有频率变化
001 7.8ms 1/128hz
010 15.6ms 2/128hz

011 23.4ms 3/128hz

100 31.3ms 4/128hz

101 39.1ms 5/128hz

110 46.9ms 6/128hz

111 54.7ms 7/128hz

（炸了。）

。。。。。

31.FF40（LCDC）

开机时为91.

LCD控制

bit 7 LCD控制*

0：完全停止（无图像显示）

1：开始控制

bit 6 窗口区块映射表的选择

0 9800-9BFF

1 9C00-9FFF

bit 5 窗口显示

0:关

1:开

bit 4 背景和窗口的区块数据集选择

0:8800-97FF

1:8000-8FFF 同OBJ一个区域

bit 3 背景区块显示选择

0:9800-9BFF

1:9C00-9FFF

bit 2 OBJ(精灵)尺寸

0:8x8

1:8x16（宽X高）

bit 1 OBJ(精灵)显示

0 off

1 on

bit 0 背景和窗口显示

0 off

1 on

*停止LCD控制必须在垂直空白中断执行的时候进行。

垂直空白中断可以通过LY的值大于等于144来确定。

32.FF41（STAT）

(这个寄存器的写入规则貌似有些特殊)

LCDC 状态

bit 6-3 LCDC 状态的中断选择

bit 6 LYC=LY coincidence （可选）

bit 5 mode 10

bit 4 mode 01

bit 3 mode 00

0：未选

1：已选

bit 2 coincidence 标志(在每次 LY 变动的同时，检查 LYC 是否等于 LY)

0：LYC 不等于 LCDC LY

1：等于

bit 1-0 模式标志(不可写入)

00 处于水平空白中断

01 处于垂直空白中断

10 正在搜索 OAM

11 正在向 LCD 设备传输信息

STAT 展示了 LCD 控制器现在的状态

mode 00 CPU 可以访问显存(8000-9FFF)

mode 01 CPU 可以访问显存(8000-9FFF)

mode 10 OAM 正在使用，CPU 不能访问 OAM

mode 11 OAM 和显存都在使用，CPU 不能访问这两者。

显示开始的典型过程：

000233000233000233000233000233000233000111111111111123

mode 标志的变化是 0，2，3 的一个 109 微秒的循环，0 大概存在 48.6 微秒，2 大概 19 微秒，3 大概 41 微秒，每 16.6ms 被垂直空白中断一次。mode 标志的 1 持续 1.08ms。(0 是 201-207 时钟周期，2 是 77-83,3 是 169-175，一个完整周期是 456，垂直空白期持续 4560，一次完整屏幕刷新每 70224 发生一次）

33.FF42(SCY)

00-FF,在 Y 方向上调整背景在屏幕上的位置。

34.FF43(SCX)

基本同上

35.FF44(LY)

LY 指示了哪一条水平线上的数据被传送到 LCD 设备上。LY 的取值范围在[0,153]，[144,153]代表了垂直空白中断过程，写入任何数据将重置这个寄存器。

36.FF45(LYC)

LYC 将自己和 LY 比较如果值相等则导致 STAT 设置 coincident 标志。

37.FF46(DMA)

DMA 传输和起始地址（W）

从内部 ROM 或 RAM（0000-F19F）到 OAM 的 DMA 传输在这里执行(40*28bits)。这一过程耗费 160 微秒。

40*28bits=8C，像你看到的一样，它只传输 8C 个字节，OAM 却长 A0 个字节，从 0 到 9F。

但是你如果观察 OAM 的数据你会发现有 4bit 没有启用。

这样就从 40*32 变为了 40*28(译注：全 TM 是废话，你复制 A0 个字节过去就完事了，这段话还误导了我好一阵子。)

它将全部的 OAM 数据送到 OAM RAM 里。

DMA 传输的起始地址可以在 0000-F100 间以 100 字节为间隔定位，就是 0000,0100,0200.。。。

正如从 FF41 中所知的那样，精灵 RAM 并不总是可用。一会会展示一个许多游戏都会采用的写入数据的方法，因为它能在适当的时候将数据移入精灵 RAM 中，主程序得以避免承担这个责任。

除了高地址(FF80-FFFE)的所有 RAM 在 DMA 传输时都是不可访问的。因此这个程序必须在高地址中拷贝和执行。它经常在垂直空白中断中被调用。

程序：

```
    org 40
    jp VBlank
    org ff80
VBlank:
```

```
    push af   保存 A rag 和标志
    ld a,BASE_ADRS 从 BASE_ADRS 传输数据
    ld (ff46),a 将 A 放入 DMA 寄存器
    ld a,28h 循环次数
Wait:         等待 160ms
    dec a     4 周期
    jr nz,Wait 12 周期,非 0 就继续
    pop af    恢复 A 寄存器和标志
    reti      从中断返回
```

38.FF47(BGP)
背景和窗口的调色版数据(RW)
bit 7-6 3 号颜色的深浅，由 0-3 越来越深(默认情况下是最暗的)
bit 5-4 2 号
bit 3-2 1 号
bit 1-0 0 号(默认情况下是最亮的)
39.FF48(OBP0)
对象调色板 0 的数据
精灵调色板 0 的颜色数据，与 BGP 相似，另外，0 号颜色是透明的，无论你设成啥。
40.FF49(OBP1)
见上
41.FF4A(WY)
窗口 Y 坐标
0<=WY<=143
否则不可见.
42.FF4B(WX)
0<=WX<=166 否则不可见.
WX 在真正的屏幕上有 7 的偏移量，WX=7，WY=0 时窗口的左上角将会显示在屏幕的 0,0.
原文 p59 有个例图，不过也就是这个意思了。
精灵仍然可以进入这个窗口(我猜是精灵显示在最顶端的意思？是的)，窗口的所有颜色都不会是透明的，

因此窗口后面的背景会被隐藏。
（译注：如果你不理解一个“反常”现象：坦克大战的坦克，在视觉效果上似乎是精灵被窗口遮盖了（坦克被状态栏覆盖），这其实是 LYC 中断的作用。）

43.FFFF(IE)
中断开启(WR)
bit 4.Hi to lo
bit 3 SIO
bit 2 timer overflow
bit 1 LCDL
bit 0 V-Blank
0:DISABLE
1:ENABLE

CPU 指令部分
1.BIT 系列指令有误，别忘了上网查一查。
2.条件跳转，对于否真正跳转耗时是不同的。

最后的译注：
卡带部分数字比较多，就没有译，应该很容易就懂 XD
它的 Doesn’t care 是真的写啥都没关系，不会导致结果不同，但是大家往往都是守规矩的，一般都选择用 0 来填充。
上 gbdevwiki 有更详细的解释。

CPU 同上，但是别忘了：
1.Flag 寄存器就是 F，AF 中后边的那个 F。
2.是小端的。
3.如果你有疑问，假如我要向 A000 号地址一次性放入两个字节的数据（利用 PUSH），会被放在 A001 和 A000 还是 A000 和 9FFF 呢？
我忘了:p
自己找个调试器仔细看看吧。
4.PUSH 是先减少 SP,再写入数据。POP 是先读取数据，再增加 SP。

音频有点复杂，最后也没写出来。