

これは管理人の個人的なメモです。なんのメモかは内緒。
こっちは抄版なのでアクセス制限なし。

- メモリ領域
 - WRAM On-board 256KB, 2Wait
 - WRAM In-chip 32KB
- ポインタ類
 - Stack Pointer 初期値
 - 特別な用途に使われるポインタ
- SRAM領域
- ROM領域
 - RTCで利用される領域
 - 不明
 - ポケモンデータ
 - ハウエン図鑑 => 全国図鑑マップ
- 画像データ
 - その他・未分類
- 定数
- サウンド設定
- タイマー設定
 - タイマー0
- その他メモ

メモリ領域

ARMなのでhWORDが16ビット、WORDが32bit、x86と異なるので注意。

WRAM On-board 256KB, 2Wait

POINTER \$02000000 何かの構造体のベースアドレス。初期化@func_0x08000bac(0x02000000, 0x1c000)

```
hWORD +0x0 0x0に初期化
hWORD +0x2 0xa3a3に初期化
WORD +0x4 0x1bf f0に初期化
WORD +0x8 0x02000000に初期化
WORD +0xc 0x02000000に初期化
```

POINTER \$020205ac 構造体の配列のベースアドレス? 要素が何個有るか要調査

```
読み出し@func_0x0800dc40()
POINTER +0x0, 0x44, 0x88, 0xcc 個数調査
hWORD +0x3c 読み出し@func_0x0800dc40()
```

POINTER \$02022080 構造体の配列のベースアドレス? 要素が何個有るか要調査

```
読み出し@func_0x080334e0()
hWORD +0x000
hWORD +0x100
```

hWORD +0x200

....

hWORD [\$0202267e] シリアル通信の初期化関数func_0x0829243c(\$03004230, 0x0e64, \$030027b4, 0x1)の戻り値が保存される。0だと正常終了

POINTER \$020226a0 - \$0202272b 0x46*2 (=0x8c) bytes分 0 fill @func_0x0800e0c8()

POINTER \$0202272c 配列のベースアドレス? 用途不明

byte [\$02022798] この値が0xffの場合には func_0x0800dc40()--V-Blankで呼ばれる関数は何もせずに returnする

byte [\$020227c8] \$020205acにある構造体の配列の1要素を指す値

byte [\$020229bd] 読み出し@func_0x080334e0()

byte [\$02022d08] func_0x08033050

byte [\$02022d0c] func_0x0805cdb8 で利用されている

byte [\$02023d08] func_0x0805cdb8, func_0x08033050 で利用されている

POINTER [\$020397c8] 構造体のベースアドレス

byte +0x15 0を保存@func_0x080b9710() -- ループでボタンを押していないときに呼ばれる関数
byte +0x18 0xffの場合に func_0x080a8964が実行されない@func_0x080b9710()

byte [\$02039bca] 何かのフラグ addr addr \$080f987c

POINTER [\$0203cc28] 0x0で初期化@func_0x080004d8()

[[[\$0203cc28]]] がV-Blankでカウントアップされる。利用方法を要調査

??? [\$02023d70] func_0x0805cdb8 で利用されている。アドレスのオフセット

POINTER [\$02024140] func_0x0805cdb8 で利用されている

POINTER \$020229c4 func_0x08033fb0 で利用されているベースアドレス。

byte +0x0 func_0x0803fb0 で0x21が保存される
byte +0x1 func_0x0803fb0 で第2引数が保存される
byte +0x2 func_0x0803fb0 で第3引数の下位8ビットが保存される
byte +0x3 func_0x0803fb0 で第3引数の上位8ビットが保存される

WORD [\$020229d4] 乱数のシード

WORD [\$02022c90] [\$03002799]&0x2 == 0|| [\$02022c90] &0x013f0102!=0 の場合 V-Blankの乱数が進まない

func_0x08033050でもなぜか読み出されている
func_0x0803340() で bit 5 i.e. 0x20 が立っていないと処理が実行されない
多分、ビット毎のフラグ、意味を要調査

WORD [\$02024664] V-Blank用乱数呼び出し回数のカウンタ

POINTER \$020246f8 構造体の or 配列のベースアドレス [\$03005af0] にアドレスが保存されている

hWORD +0xe プレイ時間の時間数が入る max 999
byte +0x10 プレイ時間の分数が入る max 59
byte +0x11 プレイ時間の秒数が入る max 59
byte +0x12 プレイ時間のフレーム数が入る max 59
byte +0x13 L=Aモードの判定 2: L=Aモード
読み出し@func_0x080005e4()

POINTER \$020373b4-\$020373d3 データが\$085c0c74から0x20バイトコピーされる
@func_0x080a1200(\$085c0c74, arg2, 0x20)

POINTER \$020377b4-\$020377d3 データが\$085c0c74から0x20バイトコピーされる
@func_0x080a1200(\$085c0c74, arg2, 0x20)

POINTER \$02037bb4-\$02037c73 0xcバイト, 0x10個の構造体配列のベースアドレス

```
WORD +0x0 0x084fcf98に初期化@func_0x080a183c(offset)
WORD +0x4 0x0に初期化?, 共用体?
byte +0x8 0x0に初期化
byte +0x9 0x0に初期化
struct {WORD val1, WORD val4, byte val8, byte val9}[16]
```

WORD \$02037c74 構造体のベースアドレス、初期化@func_0x08a1898()

```
WORD +0x0 0x0に初期化
WORD +0x4 0x0に初期化? 共用体?
WORD +0x8 0x00100000に初期化? 共用体?
```

WORD [\$0203b9fc] V-Blank割り込み中に [\$03005ae0]の値がバックアップされている。何に利用されているかは不明

WRAM In-chip 32KB

```
WORD [$03000000] 0xfc0 に初期化@func_0x080003a4()
WORD [$03000004] 0x02000000に初期化@func_0x08000bac(0x02000000, 0x1c000)
WORD [$03000008] 0x1c000に初期化@func_0x08000bac(0x02000000, 0x1c000)
```

POINTER \$03000010 - \$0300080f 要素数0x80の構造体のFIFO

```
初期化は0x80回 0x10のoffsetずつ繰り返す@func_0x08000c48()
V-Blank中にDMA3転送で利用される@func_0x08000c80()
WORD +0x0 0x0に初期化
V-Blank中にDMA3 Source Addressに代入
WORD +0x4 0x0に初期化
V-Blank中に DMA3 Dest Addressに代入
hWORD +0x8 0x0に初期化
V-Blank中に DMA Countに代入。0x1000 bytes以上だと複数回に分けて転送
hWORD +0xa 初期化箇所不明
V-Blank中で分岐に利用されるフラグ
0, 下記以外: DMA3転送を行わない
1: DMA3転送を32bit, Incrementで行う
2: DMA3転送を32bit, Fixedで行う
3: DMA3転送を16bit, Incrementで行う
4: DMA3転送を16bit, Fixedで行う
WORD +0xc V-Blank DMA転送後に0に設定
```

byte [\$03000810] 0x1に初期化@func_0x08000c48()の先頭

```
0x0に変更@func_0x08000c48()の最後
$03000010 - $0300080f の操作をアトミックにするためのmutex
0: unlock 1: lock
func_0x0800c80() V-Blank中に利用
```

byte [\$03000811] 0x0に初期化@func_0x08000c48()

```
$03000010 に対する0x10単位の FIFO の位置
```

byte [\$03000818] - byte [\$03000877] 初期化関数 func_0x08001074 で 0x00 に初期化されている。

```
[$03000818 + offset] の値が func_0x080010cc(offset) を呼び出すと IO Register [$04000000
```

byte [\$03000878] - byte [\$030008d7] 初期化関数 func_0x08001074 で 0xff に初期化されている。

```
割り込みなどの要因で LCD用IOレジスタを変更することが出来ないときに変更対象を溜めておく
0xffの時にキューに値がないと判断される
```

値は 0x04000000 からのオフセット、実際の設定値は \$03000818-に存在する
キューを全てflushする関数は func_0x08001110() で V-Blankで呼ばれている

byte [\$030008d8] \$03000878 - \$030008d7 の読み書き操作をアトミックにするためのmutex

0: unlock 1: lock

byte [\$030008d9] IME (Interrupt Master Enable Register)設定を保存しておく領域
hWORD [\$030008da] IE(Interrupt Enable Register)設定を保存しておく領域

POINTER \$030008e0 何に利用されている? 下記に初期化@func_0x0800134c()

WORD +0x00 [\$0829beac] == 0x0 に初期化。デバッグ用か?
WORD +0x14 [\$0829beac] == 0x0 に初期化。デバッグ用か?
WORD +0x18 [\$0829beac] == 0x0 に初期化。デバッグ用か?
WORD +0x1c [\$0829beac] == 0x0 に初期化。デバッグ用か?
hWORD +0x10 0x0に初期化@func_0x08001308() 共用体?

hWORD [\$03000db8] 0x0に初期化@func_0x0802ee80()の先頭

func_0x0802ef60()の戻り値を保存する

hWORD [\$03000dcc] func_0x08290708()の戻り値を保存する。RTCなどの処理をしている?
hWORD [\$03000dce] IMEを保存しておく領域。func_0x0802ed30(), func_0x0802ed48()のセットで使
われる。多分一部ルーチンのみ

WORD [\$03000e08] プレイ中かどうかを判定するフラグ

0: プレイ中ではない 1: プレイ中 2: プレイ中で999時間を超えている

WORD [\$03000e14] 読み出し@func_0x08086f98()

hWORD [\$03000f48] 0x0に初期化@func_0x080a27a8()

func_0x080a3040([\$03000f48])として利用@func_0x080a26b0()

hWORD [\$03000f4a] 0x0に初期化@func_0x080a27a8()

[\$03000f4c] == 0x5 && [\$030074d0 + 0x4] == 0の場合に0が代入@func_0x080a26b0()

byte [\$03000f4c] 0x0に初期化@func_0x080a27a8()

シーケンスが入っておりこの値を元にswich @func_0x080a26b0()

[\$03000f4c] == 0x5 && [\$030074d0 + 0x4] == 0の場合に0が代入 i.e. シーケンスをクリア

byte [\$03000f4d] 0x0に初期化@func_0x080a27a8()

WORD [\$03003020] \$0829cea4 に初期化@func_0x0061a4()

POINTER [\$030031e0] 関数のポインタだが使われる場所不明

0x08009b09 と比較@func_0x08009ad4()

byte [\$03001a68] 0x2に初期化@func_0x0828fecc(0x2,\$030027cc)

初期化のSRAM読み込み時に使われるタイマーの番号

hWORD [\$03001a6a] Timer 2 Overflow のカウンタ。実行時 1 減算、0 になったときに byte
[\$030075e0]=1 となる

POINTER [\$03001a6c] タイマー2のレジスタ TM2CNT_Lが保存される

hWORD [\$03001a76] 0x0に初期化@func_082906d8()

RTC読み書きをアトミック処理するためのmutex
0: ロック解除 1: ロック中

WORD \$03001b50 - \$03002444f 0x200*0x4 = 0x800 bytes 分 \$0828d668 - \$0828de67 からコピーさ

れている。実行される関数の高速化

```
func_0x0828d5e4() i.e. V-Blankの関数 からインラインで bx r3(=$03001b51) と飛んで
この中ではThumbとARMが混在している。WRAMなので32bitアクセスだからその方が効率が良い。
$0828d674 - $0828d6c7 ARM
$0828d804 - $0828d9fc ARM
$0828da24 - $0828dcb0 ARM の関数 func_0x0828da24()
```

hWORD [\$03002350] func_0x08005bcで 0x28に初期化

```
func_0x080005e4(キーの読み出し)で読み出されている
```

byte [\$03002354] 0x0に初期化@func_0x080003a4()

```
func_0x08086f98() == 1 の時に 0x1にセット@ループ部分
func_0x080004c4()を呼び出した後 0x0 にセット
何かを行うためのmutex?
```

POINTER \$03002360-\$03002397 ステータスを保存する構造体のベースアドレス

```
POINTER +0x00 LCD-V-Blank 内 0x0800077e で実行される関数のポインタとして利用されている。
func_0x0800051c でも関数のポインタとして実行されている
0x0に初期化@func_0x080004d8()
$0808653d と比較@func_0x080856f4()
```

```
POINTER +0x04 $0816cc91 に初期化@func_0x08000540($0816cc91)@func_0x080004d8()
func_0x0800051c で関数のポインタとして実行されている@ループ
[$03005ae8]!=1の場合0x0に変更。$0816cc91は多分セーブデータの存在を見て分岐するためのループ
```

```
POINTER +0x0c 0x0に初期化@func_0x08000684()
LCD V-Blank割り込みで実行される関数のポインタ
POINTER +0x10 0x0に初期化@func_0x08000684()
LCD H-Blank割り込みで実行される関数のポインタ、引数なし、戻り値無し
POINTER +0x14 V-Counter Match割り込みで実行される関数のポインタ 引数無し、戻り値無し
POINTER +0x18 0x0に初期化@func_0x08000684()
Serial Communication割り込みで実行される関数のポインタ
hWORD +0x1c V-Counter Match割り込みで bit 2 (0x04) を立てている
LCD H-Blank割り込みで bit 1 を立てている
LCD V-Blank割り込みで bit 0 を立てている
Serial Communication割り込みでbit 7を立てる
この bit 0 が立たない限り 通常ルーチンでは 0x080008c6-0x080008ce間でループ@func_0x08000
ループが終了すると bit0 がクリアされる
```

```
WORD +0x20 0x0に初期化。@func_0x080004d8()
LCD V-Blank割り込みで1カウントアップされている、+0x0c の関数のポインタの実行*前*@func_0
WORD +0x24 0x0に初期化@func_0x080004d8()
LCD V-Blank割り込みで1カウントアップされている、+0x0c の関数のポインタの実行*後*@func_0
```

ここからキーのステータスを保存する部分。KEYINPUTレジスタから論理が反転される
0: Released, 1: Pressed または 0: キーステータスの変更なし 1: キーステータスの変更あり

```
hWORD +0x28 キーステータスを保存@func_0x080005e4()
hWORD +0x2a キーステータスの変更を保存 0: 変更なし 1: 変更あり @func_0x080005e4()
hWORD +0x2c L=Aモードの反映されたキーステータスを保存@func_0x080005e4()
hWORD +0x2e L=Aモードの反映されたキーステータスの変更を保存@func_0x080005e4()
hWORD +0x30 キーステータスの変更を保存@func_0x080005e4()
0x32にあるキーの押し続け判定のカウンタが0になると上記値が現在のキーステータスに上書きさ
hWORD +0x32 キーの状態が前と同じ場合に1減算されるカウンタ@func_0x080005e4()
押し続ける状態がどれだけ続いているか判断される
```

キーの状態が変更されると0x28 (640 ms)にセットされる
押し続けている状態が続くと更に +0x5 (80ms) される

hWORD +0x34 [\$03002360 + 0x2e] & [\$03002360 + 0x36] != 0 の場合1が保存される@func_0x080005
hWORD +0x36 読み出し@func_0x080005e4()

POINTER \$03002398 - \$03002797 構造体配列のベースアドレス, 要素0x80個

func_0x08006e68(0x0, 0x80): 初期化
WORD +0x0 [\$082bf2f8+0x0] (=0x013000a0) で初期化
WORD +0x4 [\$082bf2f8+0x4] (=0x00000c00) で初期化
上記が0x80個分用意されている
struct {WORD val0; WORD val4;}[0x80]

byte [\$03002798] 0x0に初期化@func_0x08000540(\$0816cc91)

読み出し@func_0x0816cad8() この値を元にswich-case分岐
0x0, 0x8c, 0x8d, default の値を取る

byte [\$03002799] V-Blank乱数の進み判定に利用@func_0x08000738()

hWORD [\$0300279c] func_0x08005bcで 0x05に初期化

func_0x080005e4(キーの読み出し)で読み出されている

POINTER \$030027b0 - \$030027e7 まで 0x38 bytes 分 \$0829bdbc からコピー @func_0x08000684()

POINTER [\$030027b0]== \$08000845 ;LCD V-Counter Match
POINTER [\$030027b4]== \$08000879 ;Serial Communication
POINTER [\$030027b8]== \$0800b4e5 ;Timer 3 Overflow
シリアル通信の初期化処理中に\$0829179d(Thumb)に変更@func_0x08291078(\$030027b4+0x4, 0x3)
func_0x0800e194()から復帰時に元に戻される
POINTER [\$030027bc]== \$08000815 ;LCD H-Blank
POINTER [\$030027c0]== \$08000739 ;LCD V-Blank
POINTER [\$030027c4]== \$080008a9 ;Timer 0 Overflow
POINTER [\$030027c8]== \$080008a9 ;Timer 1 Overflow
POINTER [\$030027cc]== \$080008a9 ;Timer 2 Overflow
初期化時のSRAMの読み込み中に \$0828fea5 に変更func_0x0828fecc(0x2, \$030027cc)
POINTER [\$030027d0]== \$080008a9 ;DMA 0
POINTER [\$030027d4]== \$080008a9 ;DMA 1
POINTER [\$030027d8]== \$080008a9 ;DMA 2
POINTER [\$030027dc]== \$080008a9 ;DMA 3
POINTER [\$030027e0]== \$080008a9 ;Keypad
POINTER [\$030027e4]== \$080008a9 ;Game Pak

byte [\$030027a0] 0x0に初期化@func_0x080003a4() i.e. main関数

Serial Communicationを行っているかどうかを判断するフラグか?

byte [\$030027e8] この値が0の場合には func_0x0800b474()が実行される@func_0x08000738() -
V-Blankの関数

POINTER \$030027f0 - \$03002fdf 0x800 byte 分 \$08000248 - \$08000a47 からコピーされている。

割り込みハンドラの実行ルーチン

byte [\$03002ff0] V-Blank中に byte [\$03006120 + 0x4]の値を保存

POINTER \$0300311c 構造体 or 配列のベースアドレス

byte +0x0 読み出し@func_0x0800a114()
byte +0x1 読み出し@func_0x0800a114()
byte +0x2 読み出し@func_0x0800a114()
byte +0x3 読み出し@func_0x0800a114()

POINTER \$03003130 - \$0300317f の 0x28*2(=0x50) bytes分 0 fill @func_0x0800e0c8()
WORD [\$03003180] func_0x0800aee0 でfunc_0x0800b0f4(0x030031e4, 0x030031b0, 0x03003130)
の戻り値を保存

byte [\$0300319c] func_0x0800738(V-Blank割り込み), func_0x0800aee0, func_0x08009b64 で利用

setされる場所が見つからない
0: unlock 1:lock だと思われる。でも何をlockしているのか不明
シリアル通信をしているかどうかの制御かな?

POINTER \$030031b0 - \$030031bf の 0x8*2(=0x10) bytes分 0 fill @func_0x0800e0c8()

byte [\$030031c4] func_0x080334e0() V-Blank中で呼ばれる関数で利用

POINTER \$03003210 何かの構造体 or 配列

byte	+0x0	Timer 3 Overflow の呼び出しで使われる値、 この値が0だと Timer 3 の reload設定が行われない@func_0x0800b854()
byte	+0x1	[\$03003210] == 0 の時に 0x1を代入@func_0x0800b0f4(POINTER r0, POINTER r1, POIN
byte	+0x3	読み出し@func_0x0800b0f4(POINTER r0, POINTER r1, POINTER r2)
byte	+0xc	読み出し@func_0x0800b0f4(POINTER r0, POINTER r1, POINTER r2)
byte	+0xe	値を保存@func_0x0800b0f4(POINTER r0, POINTER r1, POINTER r2)
byte	+0xf	読み出し@func_0x0800b0f4(POINTER r0, POINTER r1, POINTER r2)
byte	+0x10	読み出し@func_0x0800b0f4(POINTER r0, POINTER r1, POINTER r2)
byte	+0x11	読み出し@func_0x0800b0f4(POINTER r0, POINTER r1, POINTER r2)
byte	+0x12	読み出し@func_0x0800b0f4(POINTER r0, POINTER r1, POINTER r2)
byte	+0x13	読み出し@func_0x0800b0f4(POINTER r0, POINTER r1, POINTER r2)
byte	+0x339(=\$03003549)	[\$0300319c] == 0 の時 func_0x08086ff4()の戻り値となる

WORD \$03004150 - \$030041e5

シリアル通信の割り込みルーチンが展開される@func_0x0829243c(\$03004230, 0x0e64, \$030027b4,

POINTER \$030041e0 V-Blank中にいろいろ読み込まれる構造体

byte	+0x02	func_0x0800d0cc()で func_0x0800cdc8(0x45, 0x0) が実行されるかどうかを判定するフ 0x0で無い場合は実行される
byte	+0x04 +0x40	の値が0の場合に0が代入される@func_0x0800c008(疑似乱数) +0x07 の値が0x5の場合に +0x07の値が代入される
byte	+0x06 +0x07	の値が0x5の場合に0x10が代入される@func_0x0800c008(疑似乱数)
byte	+0x07	0で無い場合にfunc_0x0800c200([\$030041e0 + 0x07])が実行される@func_0x0800c008(疑 何かのシーケンス番号のような気がする
byte	+0x0e	[\$0300r1e0 + 0x4] !=0 の場合に 1がセットされる@func_0x0800c008(疑似乱数)
hWORD	+0x14	0x0に設定@func_0x0800cdc8(byte r0, byte r1)
hWORD	+0x16	0x0に設定@func_0x0800cdc8(byte r0, byte r1)
hWORD	+0x1a	書き込み@func_0x0800c008(疑似乱数) 0x8c以下の疑似乱数が保存される
hWORD	+0x1c	読み出し@func_0x0800c008(疑似乱数) 0x8c - [+0x1a] の値が保存される
WORD	+0x40	関数のポインタ (*POINTER func)(byte r0, byte r1)として func_0x08295a10(byte r0, この値が0の場合には func_0x0800cdc8(byte r0, byte r1)で func_0x08295a10(r0, r1, r2)

hWORD \$03004230 - \$030042e3 の 0x0b4 byteを0で初期化@func_0x08292564()

POINTER [\$03007630]に保存されている
byte +0x0 0xff に初期化@func_0x08292564()
func_0x08293128で利用されている
読み出し@func_0x0800c008(疑似乱数)

```

byte      +0x4      0x0 に初期化@func_0x0829243c($03004230, 0x0e64, $030027b4, 0x1)
           func_0x08295240 で +0x5と比較して分岐
byte      +0x5      0x0 に初期化@func_0x0829243c($03004230, 0x0e64, $030027b4, 0x1)
           func_0x08295240 で利用
byte      +0x6      0x0 に初期化@func_0x0829243c($03004230, 0x0e64, $030027b4, 0x1)
byte      +0x9      0x4 に初期化@func_0x08292564()
byte      +0xf      0x57に初期化@func_0x0829243c($03004230, 0x0e64, $030027b4, 0x1)
byte      +0x10     0x10に初期化@func_0x08292564()

```

POINTER \$03004230+0x10(=\$03004240) 何かの構造体 or 配列のベースアドレス 上記の処理の後実行

```

byte      +0x0     0x10に初期化@func_0x0829243c($03004230, 0x0e64, $030027b4, 0x1)
byte      +0x1     0x10に初期化@func_0x0829243c($03004230, 0x0e64, $030027b4, 0x1)
byte      +0x2     0x10に初期化@func_0x0829243c($03004230, 0x0e64, $030027b4, 0x1)
byte      +0x3     0x10に初期化@func_0x0829243c($03004230, 0x0e64, $030027b4, 0x1)

```

POINTER \$030042e4 何かの構造体or配列のベースアドレス, シーケンスを制御している?

```

POINTER [$03007634]に保存されている
byte      +0x2     0x0に初期化@func_0x08293a10()
byte      +0x6     0x0に初期化@func_0x08292564()
           func_0x08293128 で1カウントダウンされている(0でない場合)
byte      +0x9     0x0に初期化@func_0x08292564()
hWORD     +0x12    0x0に初期化@func_0x08292564()
hWORD     +0x14    0x0に初期化@func_0x08292564()
hWORD     +0x16    0x0に初期化@func_0x08292564()
hWORD     +0x18    0x0に初期化@func_0x08292564()

```

hWORD \$030042e8 - \$300430f の 0x28 byte を0で初期化@func_0x08292564()

```

byte      +0x6     0x0 に初期化@func_0x08292564()

```

構造体の配列4個の実体

POINTER \$030043ec +0x0 構造体のベースアドレスが \$03007620 +0x0 に保存される
 POINTER \$030043ec +0x70 構造体のベースアドレスが \$03007620 +0x4 に保存される
 POINTER \$030043ec +0x70*2 構造体のベースアドレスが \$03007620 +0x8 に保存される
 POINTER \$030043ec +0x70*3 構造体のベースアドレスが \$03007620 +0xc に保存される

```

struct {byte value0[0x68], WORD value68, WORD value6c}, 全て0に初期化

```

構造体の配列4個の実体

POINTER \$030045ac +0x0 構造体のベースアドレスが \$03007610 +0x0 に保存される
 POINTER \$030045ac +0x1c 構造体のベースアドレスが \$03007610 +0x4 に保存される
 POINTER \$030045ac +0x1c*2 構造体のベースアドレスが \$03007610 +0x8 に保存される
 POINTER \$030045ac +0x1c*3 構造体のベースアドレスが \$03007610 +0xc に保存される

```

struct {byte value0[0x14] , WORD value14, WORD value18}, 全て0に初期化

```

POINTER \$03004ab0 割り込み制御構造体のベースアドレス. 初期化@func_0x0829243c(\$03004230, 0x0e64, \$030027b4, 0x1)

```

POINTER [$03007608]に保存されている
WORD      +0x0     0x0 に初期化
           0x0aに変更@func_0x082952f4(0x8)
           下位16ビットはfunc_0x0829117c(2)の戻り値
byte      +0x4     0x0 に初期化
byte      +0x5     0x0 に初期化
byte      +0x6     func_0x0829117c(3)の戻り値
byte      +0x7     0x0 に初期化

```

```

byte      +0x8      0x0 に初期化
byte      +0x9      0x0 に初期化
byte      +0xa      初期化は不明 実施しなくても0だが...
           割り込み直後 func_0x030027f0()で利用されている
           シリアルをGeneral-Purpose Modeに変更した直後 func_0x082910b4()で利用されている。
           タイマーの番号だと思われる 2bit lsl して $0400100h からのoffsetとして利用される
WORD      +0xc      0x0 に初期化
byte      +0x10     0x0 に初期化
hWORD     +0x12     0x0 に初期化
           func_0x08291218()ではこの値を返す。
           func_0x0829117c(0)の戻り値
byte      +0x14     0x1 に初期化
           func_0x0829117c(1)の戻り値
byte      +0x15     0x0 に初期化
WORD      +0x18     0x0 に初期化
WORD      +0x1c     0x0 に初期化
           割り込み待ちを行って、この値を返す @ func_0x08292770()
POINTER   +0x20     $08295511に初期化@func_0x0829120c($0x08295511) @func_0x082952f4(0x8)
           0x0に戻す@func_0x082952f4(0x8)
WORD      +0x24     $03004068 + 0x74(=$030040dc)に初期化
WORD      +0x28     $03004608 に初期化
byte      +0x2c     0x0 に初期化
           @func_0x08291218()
           この値が1の間は永久ループ、割り込みの待ち合わせか?
           0の場合は +0x12 の値を返す

```

POINTER \$030050a0 構造体のベースアドレス

```

POINTER   +0x00     関数のポインタが入る。利用箇所は不明
           0x0800f255と比較@func_0x0800f2a4()
byte      +0x0c     0xffに初期化@func_0x0800e0c8()
byte      +0x5c + num 0を代入@func_0x0800f1e8(num), numの範囲を要調査
byte      +0x66     読み込み@func_0x080116bc() --- V-Blankの関数
struct    +0x80, +0x94, +0xa8, +0xbc, +0xd0 size 0x14の配列5個
  hWORD   +0x0     0x0に初期化@func_0x0800f7a8($030050a0 + 0x80)
  hWORD   +0x2     0x0に初期化@func_0x0800f7a8($030050a0 + 0x80)
  WORD    +0x4     0x0に初期化@func_0x0800f7a8($030050a0 + 0x80)
  WORD    +0x8     0x0に初期化@func_0x0800f7a8($030050a0 + 0x80)
  byte    +0x10     0x0に初期化@func_0x0800f7a8($030050a0 + 0x80)
  byte    +0x11     0x0に初期化@func_0x0800f7a8($030050a0 + 0x80)
  byte    +0x12     0x0に初期化@func_0x0800f7a8($030050a0 + 0x80)
byte      +0xee     0x0に初期化@func_0x08010ca0(0x0)
           func_0x0810ca0(arg0) (arg0 !=0) を呼び出すと4カウントアップ
           $030050a0 + 0xee + 0x9ac (= $03005b39)まで0x4d6*2(=0x9ac)bytes 0 fill@func_0x0800e0c8()
byte      +0xef     読み込み@func_0x080109b0()

```

この辺アドレスの重複有り。要再調査

POINTER \$0300510c 構造体のベースアドレス 構造は\$030050a0 + 0x80に有るものと同じ

```

本当に$0300510c?
struct    hWORD    +0x0  0x0に初期化@func_0x0800f7a8($0300510c)
  hWORD   +0x2     0x0に初期化@func_0x0800f7a8($0300510c)
  WORD    +0x4     0x0に初期化@func_0x0800f7a8($0300510c)
  WORD    +0x8     0x0に初期化@func_0x0800f7a8($0300510c)
  byte    +0x10     0x0に初期化@func_0x0800f7a8($0300510c)
  byte    +0x11     0x0に初期化@func_0x0800f7a8($0300510c)
  byte    +0x12     0x0に初期化@func_0x0800f7a8($0300510c)

```

POINTER \$0300510c + 0xb8 (=\$030051c4) 何かの構造体or配列のベースアドレス

```
byte +0x0 [0x14][0x46] の配列(0x578 bytes)を0 fill @func_0x0800d184($030051c4)
byte +0x578 0x0で初期化@func_0x0800d184($030051c4)
byte +0x579 0x0で初期化@func_0x0800d184($030051c4)
byte +0x57a 0x0で初期化@func_0x0800d184($030051c4)
byte +0x57b 0x0で初期化@func_0x0800d184($030051c4)
```

POINTER \$0300510c + 0x634 (=\$03005740) 何かの構造体or配列のベースアドレス

```
byte +0x0 [0x28][0xe] の配列を0 fill@func_0x0800d1e0($03005740)
```

byte [\$03005a25] 0x0を代入@func_0x080109b0()

WORD [\$03005ae0] 乱数のシード - V-Blankで進む

WORD [\$03005ae4] 乱数のシード 用途不明

POINTER [\$03005af0] \$020246f8に初期化@func_0x080004d8()

POINTER [\$03005af4] 0x020294acに初期化@func_0x080004d8()

WORD [\$03005ae8] func_0x082902e4()の結果によって 0x0 or 0x1に設定される

hWORD [\$03005b58] 読み出し@func_0x080a3040(hWORD r0)

```
読み出し@func_0x080a2a5c(hWORD r0, byte r1)
```

POINTER \$03005b60 - \$03005f5f の先に構造体(0x28 bytes)の配列 0x10個

```
ベクタ構造になっている
func_0x080a8818(): 構造体を初期化する関数@最初のループ
func_0x080a8964(byte r0): r0番目の要素を取り除く関数
WORD +0x0 0x080a8870に初期化
読み出し@func_0x080a8aac(WORD r0)
byte +0x4 0x0に初期化
現在の構造体の位置
byte +0x5 配列の要素の番号に初期化 i.e. 0x0-0xf
先頭の要素み0xfeに初期化
前の構造体の位置
byte +0x6 配列の要素の番号 + 1 に初期化 i.e. 0x1-0x10
最後の要素のみ 0xffに初期化
次の構造体の位置
byte +0x7 0xffに初期化
byte +0x8 byte 0x20個分の配列のベースアドレス
全て0x0に初期化
struct { WORD val0; byte val4; byte val5; byte val6; byte val7; byte val8[0x20]}
```

POINTER \$03005970 上の配列の2番目の要素の先頭が \$03005b88 よって +230 => 2番目の要素の +0x18

```
byte +0x230 (=$03005ba0) 0x0で初期化@func_0x0800d1e0($03005740)
byte +0x231 0x0で初期化@func_0x0800d1e0($03005740)
[$0300319c]!=0 の時 func_0x08086ff4()の戻り値となる
byte +0x232 0x0で初期化@func_0x0800d1e0($03005740)
byte +0x233 0x0で初期化@func_0x0800d1e0($03005740)
```

WORD [\$03005e18] 乱数のシード

POINTER \$03006120 多分サウンド用構造体のベースアドレス

POINTER [\$03007ff0]にアドレスが保存されている
 \$03006120 - \$030070cf 0xfb0バイト分がサウンド用のデータが保存される領域、0で初期化されている
 WORD +0x0 0x0に初期化されている。
 タイマー1の設定後に 0x68736d53が代入される。
 \$030071c0 や \$03007250 の先のデータを変更する際にアトミックな処理を行うために利用さ
 0x08736d53で無い場合はデータ変更中
 0x08736d53 + 0x978c92ad = 0x0 Vフラグ: 1
 VCOUNTER Match内で利用されている
 byte +0x4 0x0に初期化されている。
 byte +0x5 0x0に初期化。
 byte +0x6 0x8に初期化されている。要調査
 0xc5に変更@func_0x0828ec00(0x0094c500)
 byte +0x7 0xfに初期化されている。
 0x4に変更@func_0x0828ec00(0x0094c500)
 byte +0x8 0x4に初期化されている。要調査
 byte +0x0b func_0x08295e78(0x630, [\$805fd684 + 0x6])の下位8ビットで初期化 @func_0x0828eb5c
 byte +0x0c 0に初期化@func_0x0828e920(\$03007250)
 WORD +0x10 タイマー0のカウンタ設定値。[0x085fd684 + 0x6] に初期化されている @func_0x0828eb5c
 WORD +0x14 func_0x08295e78(???,???)に初期化されている @func_0x0828eb5c(0x40000)
 WORD +0x18 func_0x08295e78(???,???)に初期化されている @func_0x0828eb5c(0x40000)
 WORD +0x1c \$03007250 i.e. サウンド用マップデータの先頭アドレスに初期化@func_0x0828e920(\$03
 WORD +0x20 関数のポインタが保存されている@func_0x0828d5e4() V-Blank中の処理
 関数のポインタとして実行@func_0x0828d5e4()
 関数のポインタとして実行@func_0x0828e24c(POINTER r0, POINTER r1) - ループ中
 WORD +0x24 func_0x0828da1e(arg0, arg1, arg2, arg3) のarg0として利用されている@func_0x0828d
 WORD +0x28 0x0828fbf9 に初期化されている
 0x0828f231に変更@func_0x0828e920(\$03007250)
 関数のポインタとして実行@func_0x0828d5e4()
 WORD +0x2c 0x0828fbf9 に初期化されている
 0x0828f165に変更@func_0x0828e920(\$03007250)
 WORD +0x30 0x0828fbf9 に初期化されている
 0x0828f0bdに変更@func_0x0828e920(\$03007250)
 POINTER +0x34 \$030071c0 i.e. ROMからコピーしたデータの先頭アドレスに初期化
 WORD +0x38 0x0828e2c1に初期化されている。要調査
 WORD +0x3c 0x0828fbf9 に初期化されている
 byte +0x50 +0x40毎0x0c個 0x0に初期化@func_0x0828ec00(0x0094c500)
 多分大きさ 0x40の配列が0x0c個有る。
 byte +0x350 DMA1 サウンドA, 右チャンネル用のサウンドデータが保存する
 byte +0x980 DMA2 サウンドB, 左チャンネル用のサウンドデータを保存する
 これら先数bytes にデータを書き込むと VCOUNTER Match で自動サウンド転送されると思われ

WORD \$030071c0 - \$0300724f は初期化時に \$085fd500 - \$085fd58f からメモリ領域をコピーされ
ている。

 0x90バイト分@func_0x0828e920(\$03007250)
 多分サウンド用のマップデータだが何に使われているか要調査
 WORD +0x20 0x0828f8f1に初期化
 WORD +0x44 0x0828e52dに初期化
 WORD +0x4c 0x0828e541に初期化
 WORD +0x70 0x0828fa49に初期化
 WORD +0x74 0x0828e4c5に初期化
 WORD +0x78 0x0828ed5dに初期化
 WORD +0x7c 0x0828e24dに初期化
 WORD +0x80 0x0828ef41に初期化
 WORD +0x84 0x0828f009に初期化

WORD \$03007250 - \$0300734f (0x100バイト分) サウンドで利用されているデータ

```

                                全域を0で初期化@func_0x0828e920($03007250)
byte      +0x01  0x1で初期化
byte      +0x1c  0x11で初期化
byte      +0x41  0x02で初期化
byte      +0x5c  0x22で初期化
byte      +0x81  0x03で初期化
byte      +0x9c  0x44で初期化
byte      +0xc1  0x04で初期化
byte      +0xdc  0x88で初期化

```

hWORD \$030074d0 構造体のベースアドレス

```

hWORD  +0x04  読み出し@func_0x080a2b20()
        この値が0の場合func_0x080a2b20()が1を戻す、0でない場合1を戻す
POINTER +0x2c  何かの構造体の配列へのポインタが入っている
WORD    +0x34  この値が0x68736d53であるかチェックされる@func_0x0828f6a4($030074d0, 0xffff, 0)
        この構造体をロックするためのmutex

```

byte [\$030075e0] Timer 2 Overflow で利用されているカウンタ [\$03001a6a] が 0 になったときに 1 がセットされる。

WORD [\$030075e4] func_0x0828fff8(sp) で一時的にsp+0x1が保存される

WORD [\$030075e8] SRAMの読み込み成否によって結果が変わる @func_0x082902e4()

```

WORD [$030075ec] 同上
WORD [$030075f0] 同上
WORD [$030075f4] 同上
WORD [$030075fc] 同上
WORD [$03007600] 同上
WORD [$03007604] 同上

```

WORD [\$03007608] \$03004ab0に初期化@func_0x0829243c(\$03004230, 0x0e64, \$030027b4, 0x1)

POINTER \$03007610 構造体4個のベースアドレスを保存した配列

```

POINTER +0x0  $030045ac      に初期化@func_0x0829243c($03004230, 0x0e64, $030027b4, 0x1)
POINTER +0x4  $030045ac+0x1c に初期化@func_0x0829243c($03004230, 0x0e64, $030027b4, 0x1)
POINTER +0x8  $030045ac+0x1c*2 に初期化@func_0x0829243c($03004230, 0x0e64, $030027b4, 0x1)
POINTER +0xc  $030045ac+0x1c*3 に初期化@func_0x0829243c($03004230, 0x0e64, $030027b4, 0x1)

```

POINTER \$03007620 構造体4個のベースアドレスを保存した配列

```

POINTER +0x0  $030043ec に初期化@func_0x0829243c($03004230, 0x0e64, $030027b4, 0x1)
POINTER +0x4  $030043ec+0x70 に初期化@func_0x0829243c($03004230, 0x0e64, $030027b4, 0x1)
POINTER +0x8  $030043ec+0x70*2 に初期化@func_0x0829243c($03004230, 0x0e64, $030027b4, 0x1)
POINTER +0xc  $030043ec+0x70*3 に初期化@func_0x0829243c($03004230, 0x0e64, $030027b4, 0x1)

```

POINTER [\$03007630] \$03004230 に初期化@func_0x0829243c(\$03004230, 0x0e64, \$030027b4, 0x1)

POINTER [\$03007634] \$03004230 +0xb4(=\$030042e4) に初期化@func_0x0829243c(\$03004230, 0x0e64, \$030027b4, 0x1)

POINTER [\$03007638] \$03004230 +0xb4 +0x28 (=\$0300430c) に初期化
@func_0x0829243c(\$03004230, 0x0e64, \$030027b4, 0x1)

```

                                POINTER +0x4  $03004230 + 0xb4 +0x28 + 0x9 に初期化。
                                次のルーチンを割り込み処理でthumb実行できるように設定?
                                hWORD    +0x8 - 0x68 の 0x60 bytes分 $0829397c からコピー
                                見たところ実行ルーチン。

```

POINTER \$03007640 - \$0300764b の0x3*4(=0xc) bytes分0 fill func_0x082953a8()

		厳密な動作は追っていないが、シリアル通信のシーケンスを制御するための構造体
byte	+0x0	この値にステータスを保存
byte	+0x1	この値でシリアル処理をしていることの判別 0: シリアル処理をしていない 1: シリ
byte	+0x2	
hWORD	+0xa	

ポインタ類

Stack Pointer 初期値

POINTER \$03007e40 通常モード, システムモードでの Stack Pointer の初期値
 POINTER \$03007fa0 IRQモードでのStack Pointerの初期値

特別な用途に使われるポインタ

HW制御のための構造体を指すなど特別な用途に使われるポインタ

POINTER [\$03007ff0] 多分サウンド用構造体のベースアドレスを保存する(\$03006120)

LCD V-Counter Match内呼び出される func_0x0828df98で利用されている

byte [\$03007ff4] LCD V-Counter Match内呼び出される i.e. func_0x0828df98()

hWORD [\$03007ff8] LCD V-Counter Match 割り込み内で bit 2 を立てている (0x04との or)

LCD H-Blank割り込み内で bit 1 を立てている
 Serial Communication割り込みで bit 7を立てる

POINTER [\$03007ffc] 割り込みハンドラのアドレス \$030027f0 が設定されている

SRAM領域

0E000000-0E00FFFF Game Pak SRAM (max 64 KBytes) - 8bit Bus width @gbatek

byte [\$0e005555] 0xaaを代入 @func_0x0828fe08()

byte [\$0e002aaa] 0x55を代入した後に0x90を代入@func_0x0828fe08()

byte [\$0e002aaa] 0x55を代入 @func_0x0828fe08()

ROM領域

RTCで利用される領域

See gbatek.

hWORD [\$080000c4] 0x1が書き込まれる @func_0x08290864(WORD r0)

上記の後すぐに0x5が書き込まれる@func_0x08290864(WORD r0)

hWORD [\$080000c6] 0x7が書き込まれる @func_0x08290864(WORD r0)

RTC 3bitを Outモードにする

hWORD [\$080000c8] 0x1が書き込まれる @func_0x08290f78()

RTCをread-writeに変更

不明

POINTER \$0863c27c - ポインタとして利用@func_0x0828e6b8(arg0)

0xc bytesのデータを持つ構造体の配列
データの的には \$0863c22abまでの 0x4個の様だが...

POINTER \$0863c2ac - ポインタとして利用@func_0x0828e6b8(arg0)

8bytesのデータを持つ構造体の配列
データの的には \$0863d423 までの 0x22f (=559d) 個の様だが...
POINTER +0x0
hWORD +0x4 \$0863c27c のデータを参照するときのオフセットが入っている 0 -

ポケモンデータ

POINTER \$082f0d70- フシギダネのデータ

POINTER \$082f1dbc ミュウツウのデータ 差分が0x95 (=149) OK

POINTER \$082f2890 ルギアのデータ 差分が0xf8 (=248) OK セレビィまではOK

POINTER \$082f2ba0 キモリのデータ 差分が0x114(=276)個分 0x18分ずれている。なぜ?

POINTER \$082f3a10 ジラーチのデータ 差分が0x18, デオキシスのデータは不明

各ポケモンデータは 0x1c bytes の構造体

byte	+0x00	HP
byte	+0x01	攻撃
byte	+0x02	防御
byte	+0x03	特攻
byte	+0x04	特防
byte	+0x05	素早
byte	+0x06	?
byte	+0x07	?
byte	+0x08	被捕獲度
byte	+0x09	経験値
byte	+0x0a	?
byte	+0x0b	?
byte	+0x0c	?
byte	+0x0d	?
byte	+0x0e	?
byte	+0x0f	?
byte	+0x10	?
byte	+0x11	?
byte	+0x12	?
byte	+0x13	?
byte	+0x14	?
byte	+0x15	?
byte	+0x16	?
byte	+0x17	?
byte	+0x18	?
byte	+0x19	?
byte	+0x1a	?
byte	+0x1b	?

ハウエン図鑑 =>全国図鑑マップ

POINTER \$082ee940-\$082eead3 ハウエン図鑑から全国図鑑へマップするhWORD 202(=0xca)個の配列

hWORD +0x?? 全国図鑑の番号

画像データ

POINTER \$085c0c94 初期化直後画面の圧縮データの先頭アドレス, \$06000000に展開される

WORD +0x0 0x0004e010 i.e. LZ77, 展開後のサイズ 0x4e0バイト
byte +0x4 0x33

POINTER \$085c0ef8 初期化直後画面の圧縮データの先頭アドレス, \$06003800に展開される

POINTER \$085c0c74-\$085c0c93 読み出し@func_0x080a1200(\$085c0c74, 0x0, 0x20),最初のループ

その他・未分類

byte [\$08000878] V-Blank割り込みで呼び出される関数 func_08001110で利用されている。
??? 0x08001fd5 関数の引数として利用されているfunc_0x080a8aac(0x08011fd5)@func_0x080116bc()

POINTER \$0829397c - \$082939db の0x60 bytes 分、シリアル通信で使われる?

WORD [\$0829beac] 0x0 [\$030008e0]に代入されるが何に使われる?@func_0x0800134c()

POINTER 0x082facb4 0x0805cdfcで使われている。何かの領域のPOINTER, V-Blankなので画像を扱っている? @func_0x080334e0()

POINTER \$0890ed54 ポインタの配列として利用されている@func_0x082902e4()

POINTER +0x00 \$0890eda0
hWORD +0x2c 0x09c2
POINTER +0x04 \$0890ee3c
hWORD +0x2c 0x1362
POINTER +0x08 \$0890edd0
hWORD +0x2c 0x0000

定数

0x41c64e6d 線形疑似乱数生成器で使われる定数

0x00006073 線形疑似乱数生成器で使われる定数

0x08736d53 サウンド処理をアトミックにするための mutexの値。なぜ 0 or 1 ではなくこの値なのか不明

サウンド設定

- DMA1, DMA2 でV-COUNTER Match 割り込み中にサウンドデータの転送を行う
- SND A: 右チャンネル, DMA1
- SND B: 左チャンネル, DMA2
- サウンドは 8bit/ 65.536kHz
- サウンド用構造体のベースアドレスは \$03006120
- 初期化時の DMA1 Source Addr は \$03006470
- 初期化時の DMA2 Source Addr は \$3006aa0

タイマー設定

タイマー0

タイマー0はサウンドに利用されるようである

- TMOCNT_Hを0に初期化@func_0x0828eb5c(0x40000)
 - Prescalar Selection:0=F/1(システムクロック16.78MHz毎にカウントアップ), Count-up Timing: Normal, Timer IRQ: Disable, Timer Start/Stop: Stop
- TMOCNT_L i.e. タイマーのカウンタ設定値を func_0x08295e78(0x44940, [\$03006120+0x10]) に設定

その他メモ

- 線形疑似乱数生成器のシードは最低4個、5関数存在する
- H-Blank/V-Blank の割り込みは VCOUNT, IE, IME の3箇所で設定する?
- 多重割り込みが入るのは H-Blank, VCOUNTER, Timer ?, Timer 3, Serial, Game Pak
- V-Blankは他の割り込み中に入らない
- func_0x0800a6b8()でfunc_0x082959bc(\$0202272c+(3-r4)*0x1c, 0, 0x1c)を呼び出しているけどなんだ? bios領域に書き込んでいる?