

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。

## カートリッジ

アセンブラをポチポチ書き写しながらやっていたけど効率が悪すぎるので効率化することを考える。

データの中身をバイナリエディタで眺めると0x0829bd9cまではプログラムコード、実質はポインタがあるので0x0829bda3まで、以降がデータのようなのである。

```
dd if=DUMP.GBA of=code.gba bs=1 count=2735524
```

でコード部分を取り出す

```
~/bin/disarm/disarm.exe -t -s8000000 code.gba |sed -e 's/^10/08/' > code-thu.asm  
~/bin/disarm/disarm.exe -s8000000 code.gba |sed -e 's/^10/08/' > code-arm.asm
```

でthuとarmのコードを取り出す。

一部のアドレス(\$08xxxxxx)となっている部分を数値に変換する

```
#!/bin/perl  
  
open(IN1, "code-thu.asm");  
open(IN2, "../code.gba",);  
  
@codes = <IN1>;  
  
foreach $line (@codes){  
    chop($line);  
    if ($line =~ /\ (\ \ $([0-9a-f]{8}) \ )$/){  
        $addr = $1;  
        seek IN2,hex($addr)-0x08000000, 0;  
        read (IN2, $mem, 4) or die "error $addr ",hex($addr);  
        my ($num) = unpack('L',$mem);  
        printf "%s (= \ $%08x) \ n", $line, $num;  
    } else {  
        print $line," \ n";  
    }  
}
```

valueの部分を取り除くためにvalueのアドレスを取り出す。

```
#!/bin/perl  
  
open(IN1, "code-thu.asm");  
  
@codes = <IN1>;  
  
foreach $line (@codes){  
    chop($line);  
    if ($line =~ /\ (\ \ $([0-9a-f]{8}) \ )$/){  
        $addr = $1;  
        printf ("%08x \ n%08x \ n", hex($addr), hex($addr)+2);  
    }  
}
```

bl \$addr となっている部分をサーチして関数の先頭アドレスを取り出す。

```
egrep bl[^a-z] code-thu.asm |cut -d \$ -f 2 |sort |uniq |egrep -v "^07" > func-addr-thu
```

関数の先頭が分かるようにコードを加工する。

```
#!/usr/bin/perl

open(IN1, "func-addr-thu");
open(IN2, "code-thu3.asm");
@vals = <IN1>;
@codes = <IN2>;

$i=0; $j=0;
chop $vals[$i];
foreach $line (@codes) {
    if (length($line)<8){ next;}
    $addr = substr($line,0,8);

    if ( $addr gt $vals[$i] ) {
        $i++;
        chop $vals[$i];
        next;
    } elsif ($addr eq $vals[$i]){
        print " \n";
        print " \n";
        print ";;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;";
        print ";;; ??? func_0x$vals[$i](???)", " \n";
        print ";;;", " \n";
        print $line;
        $i++;
        chop $vals[$i];
        next;
    } else {
        print $line;
        next;
    }
}
```

関数のポインタとして実行される部分、メモリに展開されてARMで実行される関数はリストされない  
ので解析しながら随時付け足してゆく。

あと、GBAのデバッガが欲しいな。なんか良い方法が無いかな。エミュレータにgdbでアタッチできる  
か要調査。

## ARM読み方メモ

- r0, r1, r2, r3 が引数のレジスタとして使われるっぽい
- 引数が5個以上の場合はspに引数を積む。関数の先頭でspを読み出す。関数の最初にレジスタをpushする処理があるので、それによりずれるアドレスを考える必要がある。
- r0 が戻り値のレジスタとして使われるっぽい
- bx r0 で戻っているのは void の関数 bx r1, で戻っているのは戻り値がある関数
- r4-r7 が変数用レジスタとして使われるっぽい
- r15がプログラムカウンタ pc
- r14 がリンクレジスタ lr
- r13 がスタックポインタ sp
- r12 が ip レジスタ
- r11 が fp レジスタ
- r10 が sl レジスタ
- THUMBではr8-r12はほとんど使われない ... まあ使うのが面倒だからね。
- 割り込みなどの非常に多用するルーチンは0x3000000からの WRAM - In-chip に展開する。32bitバスなのでARM命令

- V-Blankの一部関数も WRAM - In-chip に展開される。ThumbとARMが切り替わりながら実行されるので読みにくい
- カートリッジから直接実行するルーチンは16bitバスなのでThumb命令