

[逆アセンブラを用いたコードサーチ](#)で満足できなかった変人さんいらっしゃーい。
出てきた言葉が理解できない場合はまだまだチャレンジするには早いので、
別の場所で修行をしてくるがよろし。

そもそも自作ルーチン埋込って何ぞな？

R4やらPARで使っている「改造コード」は、別にハード的にDSを改造してるわけでもなく、
単純に空きメモリにチートプログラムを展開しているだけだったり。

んで、ゲームによっては結構空きメモリがあるので、改造コードを利用して、
空きメモリに自作プログラムを書き込み、ゲームプログラムの一部を書き換えて
自作プログラムにジャンプさせ、再び元の場所に戻る……。
これで、元々あったプログラムに自分の作ったプログラムを埋め込むことができる。
原理的にはコンピュータウイルスと似たような感じ。

準備

[ココ](#)を参考にしてDevkitAdv環境を作る。
SOURCEFORGE.NETから6つのzipファイルを落とすのが超大変だけどガンガレ。
C:\devkitadv-r5-beta-3\ に全部展開できたら、C:\devkitadv-r5-beta-3\binにパスを通す。
わからなかったらその都度コマンドプロンプトで PATH=%PATH%;C:\devkitadv-r5-beta-3\bin\でおk。

C:\devkitadv-r5-beta-3\test フォルダを作って中に main.c を作成。
中身はこんなん

```
void test(){
asm("nop \n");
}
int main(){
test();
return 0;
}
```

さらに以下のコマンドを実行して、test.binが生成されたらOK。

```
cd C:\devkitadv-r5-beta-3\test
gcc -o test.elf main.c
objcopy -O binary test.elf test.bin
```

これで生のARMコードを得ることができるようになった。
本来 DevkitAdv はGBAソフトの開発環境だが、改造コード作成に利用することもできるのだ。
あと、gccとobjcopyは頻繁に使うことになるので上2行はバッチでまとめたほうがいい。

NDSの知識を身につけておこう

[NDSTech Wiki](#)は最低限読めるだけ全部目を通しておく。
どのアドレスにキー情報やタッチパネル情報が格納されていて、どうやってアクセスするかも知っておく。
ARM9/ARM7の役割やらも適当に覚えといたほうがいい。

Newスーパーマリオをハック

それでは、通常の改造コードでは絶対不可能な

コインの取得枚数によってストックアイテムが変化するコード

を作ってみる。

コインはアドレス0208A994、ストックアイテムはアドレス0208A944に格納されており、

「00無し 01キノコ 02フラワー 03コウラ 04マメキノコ 05巨大キノコ」となっている。

これより範囲外の値は入れてはダメ。

よって、コイン枚数から余計な情報を省いて00(2進数000)~05(2進数101)だけを取り出し、

これをストックアイテムのアドレスに渡してやる必要がある。

ルーチンは まあこんなもんで。

「コイン(0208A994)の値をロード 8ビットのうち上5ビットを除去(7でANDマスク) ストックアイテムに渡す」

(除算で余剰を求めてそれを入れてもいいけど、面倒じゃね？ ネット改造でそこまで本気にならなくてよくな？)

ルーチンをコーディング

さっきのmain.cにasm(" ~ ~");って感じに打ってやることでインラインアセンブラを利用できる。

これで地道にねちねちとコーディングしてやる。

```
void test(){
// nop nop nopを目印に
asm("nop \ nnop \ nnop \ n");
asm(
" stmb r13!,{r0-r5} \ n" /* r0-r5をスタック待避 */
" ldr r0,_CoAddr \ n" /* r0にコインのアドレスをロード */
" ldrb r1,[r0] \ n" /* r1にコインの枚数をロード */
" and r1,r1,#7 \ n" /* andマスクで上5ビットを除去 */
" cmp r1,#6 \ n" /* もし6以上(0-5範囲外)なら */
" bge _E \ n" /* 終了ラベル_Eに飛ぶ */
" ldr r0,_StAddr \ n" /* r0にストックアイテムのアドレスをロード */
" str r1,[r0] \ n" /* r1をストックアイテムに上書き */
"_E: ldmia r13!,{r0-r5} \ n" /* r0-r5をスタックから戻す */
" bx r14 \ n" /* コール元に戻る */
"_CoAddr: .long 0x0208A994 \ n" /* コイン格納アドレス */
"_StAddr: .long 0x0208A944 \ n" /* ストックアイテム格納アドレス */
);
}
int main(){
test();
return 0;
}
```

なんてひどいレイアウト.....誰か修正してくれえ。

念のためtest.c test.elf test.bin test.bat をアップ。参考に。

[Download test.zip](#)

これをさっきのコマンドで test.bin を生成してやれば自作ルーチン作成は終わり。

ちなみに末端を "bx r14" でシメているが、これは後々で説明する。

test.binから必要コードを取り出す

ndsdis2のNHモードでムリヤリ逆アセ。

```
ndsdisk2 -NH9 0 test.bin > test.txt
```

これで出てきた test.txt をテキストエディタで開いて "nop" で検索。

```
:00000380 E1A00000nop(mov r0,r0)mov r0,r0 ;r0=134219196(0x80005bc)
:00000384 E1A00000 nop (mov r0,r0)mov r0,r0 ;r0=134219196(0x80005bc)
:00000388 E1A00000 nop (mov r0,r0)mov r0,r0 ;r0=134219196(0x80005bc)
:0000038C E92D003F stmdb r13!,{r0,r1,r2,r3,r4,r5}
:00000390 E59F001C ldr r0,[r15, #+0x1c] ;r15+0x1c=*(000003b4)=#34122132(0x0208a994)
:00000394 E5D01000 ldrb r1,[r0, #+0x0] ;r0+0x0=*(0208a994)=#0(0x00000000)
```

言うまでもなくコレが自作ルーチン。ちゃんとコードも出てる出てる。

頭のnop x 3を除いた「アドレス 0000038C ~ 000003B8」の部分が必要なコードなのでこれをしっかりと控える。

```
E92D003F
E59F001C
E5D01000
E2011007
E3510006
AA000001
E59F000C
E5801000
E8BD003F
E12FFF1E
0208A994
0208A944
```

アドレスは常駐先によって変動するので不要。

どこに常駐するのか？を考える。

とにかく emuhaste で「空きメモリっぽい場所」を探す。
他のゲームではありえないが、独自のプログラム展開ルーチンを持つ New マリオの場合、
アドレス 02000B90 ~ 02002FFF もの巨大な領域がガラ空きになっているのでココを使う。
(ちょっと間をとってアドレス 02002000 から使ってみる)。

ふつーのゲームなら 02370000 ~ 0237FFFF の間くらいで探すべし。

どこから飛ぶのか？を考える。

さっきの自作ルーチンのシメが "bx r14" だったが、これはつまり
「どっかで使われている bx r14 を乗っ取って自作ルーチンに処理を飛ばして、
最後に bx r14 を代わりに自作ルーチン内で実行して戻る」
ということだ。

んじゃ [逆アセンブラを用いたコードサーチ](#) で説明した方法で New マリオの生コードを逆アセンしてエディタで開く。
4000130 と検索してヒットし、なおかつ **直後に bx r14 のあるところ** を探す。
つーわけで

```
:02043CF4 E59F1044 ldr r1,[r15, #+0x44] ;r15+0x44=*(02043d40)=#67109168(0x04000130)
中略
:02043D38 E12FFF1Ebx r14(Jump to addr_02043AA4?)
```

アドレス 02043D38 がアタリ。

これはゲーム内に存在するキー入力ルーチンの末端であり、bx r14 で元の処理に戻る瞬間である。
ここを "b 02002000" に書き換えることで、自作ルーチン常駐アドレスにムリヤリ飛ばすことができる。
このコードは以下の式で求めることができる。

1. 「常駐先アドレス < ジャンプ元のアドレス」の場合

```
EA000000 + (常駐先アドレス - ジャンプ元のアドレス - 8) / 4  
EA000000に足す値は下6桁のみ
```

2. 「常駐先アドレス > ジャンプ元のアドレス」の場合
EA000000 + (常駐先アドレス - ジャンプ元のアドレス) / 4 - 2

今回の場合は「02002000 < 02043D38」なので1の式を用いることになり、

```
EA000000 + (02002000 - 02043D38 - 8) / 4  
EA000000 + FEF8B0 (下6桁のみ)  
EAFEF8B0
```

よって、コード**02043D38 EAFEF8B0**を実行することで、
:02043D38 E12FFF1E bx r14 EAFEF8B0 b 02002000
と変化する。

[Kodinator](#)

上記の事を自動でやってくれるツール。ダウンロードには要ログイン。

cheats.gbatemp.net/forum/general-hacking-discussion/nds-hacking-tools/
ログイン不要でダウンロード出来る改造ツールの詰め合わせの中にも入ってる。

自作ルーチンをアドレス02002000以降に書き込む

これは簡単。

要は、さっきの羅列に02002000、02002004、02002008...とアドレスをつけるだけ。
bx r14書き換えコードと組み合わせると以下の通り。

コインの下3ビット情報をストックアイテムに反映する

```
02002000 E92D003F  
02002004 E59F001C  
02002008 E5D01000  
0200200C E2011007  
02002010 E3510006  
02002014 AA000001  
02002018 E59F000C  
0200201C E5801000  
02002020 E8BD003F  
02002024 E12FFF1E  
02002028 0208A994  
0200202C 0208A944  
02043D38 EAFEF8B0
```

コイン取るたびにストックアイテムが変化したら成功。

6(110)と7(111)は無視され、8以上になったら下3ビットだけが用いられるため、
8(1000)=0(000/無し)、99(1100011)=3(011/コウラ)という扱いになる。

実機だとダメかもねえ

ぶっちゃけPARでチェックしてねーです。

もしかすると 02043D38 EAFEF8B0 でクラッシュするかもしれないので、その場合は
キーコードと組み合わせて「L + R同時押下したら02043D38 EAFEF8B0を実行」みたいにしたほうがいいのかも。

ふー、つかれた。

以下FAQ

Q.自作ルーチン冒頭の stmdb r13!, { r0-r5 } と最後の ldmia r13!, { r0-r5 } って何？

A.86系CPU風に言うとPUSHとPOP。

レジスタ内容をぶっ壊すとキー入力ルーチンだけでなくプログラム全体に影響が出るかもしれんから、r0～r5をstmdbでスタックに逃がしておいてから自作ルーチンでレジスタを使って、使い終わったらまたr0～r5をスタックから戻して元の値を入れるわけやね。

Q.DevkitAdvの代わりにdevkitProのインラインアセンブラは使えない？

A.stmdbとかldmiaだけでなくldrのラベルすらまともに使えなかった。

つーか、よくわからんかったので俺はお手上げ。

Q.キー判定以外のbx r14は利用できんの？

A.ふつーに使える。んで、ゲームによってはキー判定のbx r14を利用できんものもある。

そういうときはDSエミュのデバグ回して使えそうなアドレスを探す。

bx r14以外にも色々使える場所はあるので根性で探すべし。

Q.アセンブラやったことないけど自作ルーチン組める？

A.根性があればどうにでもなるんじゃない？ でも、せめてCASLくらいはかじっておかないと、

アセンブリの概念が全く理解できんと思うのだけど。

Q.インラインアセンブラ使うときわざわざtest()関数作ってるのは何故？

A.main()に直接コーディングすると、たまによくわからんエラーを吐くことがあるから。

理由は知らないけど、別関数作ってやると回避できるんよね。