

逆アセンブラを用いたコードサーチ

～はじめに～

ARMのCPU仕様はちゃんとリファレンスマニュアル読んだ方がいいです。
(変則的なビットパターンが多いので、偏に説明はできんとです)。
"ddi0100e_arm_arm.pdf" で [ググって](#) 出てきたPDFを読むとよかですな。

ゲーム改造でよく使うオペコード一覧と用法とか。

opcode	書式	効果	用途とか
add	add rd,rn,op2	rd = rn + op2	加算。お金増やしたりEXP増やしたり
sub	sub rd,rn,op2	rd = rn - op2	減算。体力減らしたりお金減らしたり
mov	mov rn,op2	rn=op2	代入。直接値を叩き込むときに使う
cmp	cmp rn,op2	if(rn==op2)	比較命令。条件分岐フラグの値を変更する
ldr	ldr rd, addr_mode1	rd=*((DWORD)addr_mode1)	指定アドレスのDWORD値をrdレジスタにロード
ldrh	ldrh rd, addr_mode1	rd=*((WORD)addr_mode1)	指定アドレスのWORD値をrdレジスタにロード
ldrb	ldrb rd, addr_mode1	rd=*((BYTE)addr_mode1)	指定アドレスのBYTE値をrdレジスタにロード
str	str rd, addr_mode1	*((DWORD)addr_mode1)=rd	指定アドレスにrdレジスタにDWORD値でストア
strh	strh rd, addr_mode1	*((WORD)addr_mode1)=rd	指定アドレスにrdレジスタにWORD値でストア
strb	strb rd, addr_mode1	*((BYTE)addr_mode1)=rd	指定アドレスにrdレジスタにBYTE値でストア
b	b addr	r15(pc)=addr	アドレスaddrにジャンプする。cmpと組み合わせて分岐したり
bx	bx reg	r15=reg	bx r0でr0レジスタの値のアドレスに飛ぶ。blと組み合わせたりする
bl	bl addr	r14(lr)=r15(pc)+4 , r15=addr	r14(lr)に戻り先アドレスを入れてから飛ぶ。bx r14で戻れる

Irregular child [dip_007.htm](#)より一部抜粋・改変
ARMはほぼ全ての命令に条件文を付けることができる。
cmp r0,#0x00 / beq 020241FC ... r0と0を比較 / beq(b:ジャンプ eq=等しい:Equal) 020241FC
= r0が0のとき、アドレス020241FCにジャンプ。
こんな感じ。

以下、スレ6/171-172・218より抜粋および修正。
文中はhasteDSだけemuhaste使っても同じなので好みで。

～逆アセできない場合の対処～

普通は [ndsdis2](#) [ndsromname.nds](#) > [disasmlist.txt](#) だけでOKだけど、
ニュースーパーマリオ(以後NewMario)とかはプログラムコードが
圧縮(スクランブル?)されてるっぽいので生データを抜くために

エミュで一度走らせるっす。
NewMarioをNo\$gbaで立ち上げたらhasteDSでSnapして、
[メモリダンプ(DUMP)]でヘキサダンプを保存(mario_dumped.bin)、
NDSDIS2に以下の引数を付けて実行すれば出力可能。
[ndsdis2 -NH9 2000000 mario_dumped.bin > mario_dumped.txt](#)
ただし逆アセリリストが超巨大(40メガバイトオーバー)なので、
バイナリエディタで必要なコードだけ切り出して実行したほうが
スマートだと思うのさ。

～1UP処理の書き換え～

hasteDSでコイン枚数をサーチするとアドレス0208A994にヒットするけど、
逆アセリリストで"0208a994"を検索するとこんな感じのがヒット。
:0201FE34 E59F103C ldr r1,[r15, #+0x3c];r15+0x3c=(0201fe78)=#34122132(0x0208a994)
:0201FE38 E1A02104 mov r2,r4,lsl #0x2;r2=0(0x0)
:0201FE3C E7910104 ldr r0,[r1,+r4, lsl #0x2]
:0201FE40 E3500063 cmp r0,#0x63
:0201FE44 17910002 ldrne r0,[r1, +r2]
:0201FE48 12800001 addne r0,r0,#0x1
:0201FE4C 17810002 strne r0,[r1, +r2]
:0201FE50 1A000005 bne 0201FE6C
コイン99枚でさらにコインを取得して～っぽい雰囲気を感じ取ってくれたらOK、
ちなみにアドレス0201FE50の1A000005をE1A00000(nop)に書き換えると
コイン1枚取ると同時にコインが0枚になり1UPする。
また、0201FE40のcmp r0,#0x63のコレを#0x04(E3500004)にすると
コイン5枚ごとに1UPする(なんというデフレ)。
あと、0201FE48のaddne r0,r0,#0x1が「コイン1枚増やす」なので
この値を増やせば1枚取得ごとに猛烈にコインが増えるけど、
「99 100以上」というステップが無い限り1UPイベントが起こらないため、
E2800021 (add r0,r0,#0x21)で33枚ずつ増える程度にしたほうが良い、
～というわけで今回のコード～
コイン1枚で1UP
0201FE50 E1A00000
コイン5枚で1UP
0201FE40 E3500004
コイン取得時に33枚増える
0201FE48 E2800021
とりあえずhasteDSでパラメータのアドレスが判明したら、
逆アセンブルリストからそれを検索してみると色々面白いさ。
キャラクターステータスとかはテーブルでまとめて処理してるから
見つからないと思うけど、所持金とか「倒した敵の数」あたりは
あっさり見つかるのでそこを掘り起こしていくと結構面白い。
「倒した敵の数」処理特定 戦闘ルーチン特定 戦闘ルーチンをコールしている処理を特定
そこを書き換える&キーコード組み合わせると「Lボタン押してると敵が出ない」とか。

～スコア上昇倍率の変更～

NewMarioのスコア格納アドレスは0208A99C。
コレを逆アセリリストから検索すると結構色々ヒットするが、以下のコードに注目。
:0201FD78 E59FC01C ldr r12,[r15, #+0x1c];r15+0x1c=(0201fd9c)=#34122140(0x0208a99c)
:0201FD7C E59F201C ldr r2,[r15, #+0x1c];r15+0x1c=(0201fda0)=#34111996(0x020881fc)
:0201FD80 E79C3100 ldr r3,[r12,+r0, lsl #0x2]
:0201FD84 E0833001 add r3,r3,r1 << スコア加算処理
:0201FD88 E78C3100 str r3,[r12,+r0, lsl #0x2]
:0201FD8C E5920034 ldr r0,[r2, #+0x34];r2+0x34=(02088230)=#0(0x00000000)
:0201FD90 E0800001 add r0,r0,r1
:0201FD94 E5820034 str r0,[r2, #+0x34];r2+0x34=(02088230)=#0(0x00000000)
:0201FD98 E12FFF1E bx r14 (Jump to addr_00000000?)
r12レジスタにスコア格納アドレスをロードした後、r3レジスタにスコア値をロード、
add r3,r3,r1でスコアにr1レジスタの内容を加算してからstrで書き戻しているが、
これのスコア上昇倍率をいじってみるっぺ。
ADDのビットパターンは以下の通りで、
cond(4) + 00(2) + l(1) + 0100(4) + S(1) + Rn(4) + Rd(4) + shifter_operand(12)
カッコ内の値はビット数。ARMモードなので合計32ビット。
add r3,r3,r1はE0833001なので以下のようなになる。
1110 + 00 + 0 + 0100 + 0 + 0011 + 0011 + 000000000001
最後のshifter_operandをいじればシフト命令を埋め込めるのだが、
shifter_operandについてはARMリファレンスのA5-2以降を参照してくれい。
今回の場合shifter_operandの最後4ビットが演算結果保存用レジスタなので、

r1つまり000000000001という値だが、add r3,r3,r10の場合は 000000001010 となる。
倍率を変えるには000000000001とシフト値を合わせたものと最後の12ビットを
差し替える必要がある。

(左1ビットシフトごとに2倍、右1ビットシフトで半分になる)。

シフトをいじると以下のようなコードになるので参考に。

```
E0833001 add r3,r3,r1 (元コード)
```

```
E0833081 add r3,r3,r1,lsr #0x1
```

```
E0833101 add r3,r3,r1,lsr #0x2
```

```
E0833181 add r3,r3,r1,lsr #0x3
```

```
E0833201 add r3,r3,r1,lsr #0x4
```

LSL(左シフト)を1ビット(2倍)、2ビット(4倍)、3ビット(8倍)、4ビット(16倍)

行うものだが、これをスコア加算部分と差し替えると

スコア取得量16倍

```
0201FD84 E0833201
```

こんな感じになる。

~あの命令のコードがわからないよっ！~

上の "add r3,r3,r1" が E0833001 で "add r3,r3,r1,lsr #0x4" が E0833201、

じゃあ "add r1,r1,r10,lsr #0x6" はどんなコード？

という場合、普通はARMリファレンスの4.1.3(A4-6)のadd命令のビットパターンに
値を当てはめてから16進数に変換して求めるんだけど、そんなんするくらいなら

逆アセリストから、書き換え後と同じ内容のものを検索すればいい。

ヒットしない場合は諦めてハンドアセンブルで求めるしかないけど、

うまくヒットすれば超カンタンにコードを発見できる。

[ASM to ARDS](#)

arm命令をコードに変換してくれる便利ツール