夏のプログラミング･シンポジウム「プログラム詠み会」2015.9.4-6

Raspberry Piのハッカソン

和田 英一[1,a)]

概要：Raspberry Pi は ARM を CPU とするシングルボードコンピュータで, Linux が動く. ARM のアーキテクチャに興味があったので, ハッカソンの機会にアセンブリ言語で `cal` と除算サブルーチンのプログラムを書いてみた. 多くの命令が条件付きで実行する/しないようにでき, また実行後に実行結果フラグの設定する/しないができ, プログラムがハックできる楽しさがあった.

キーワード：Raspberry Pi, アセンブリ言語, `cal`, 除算サブルーチン

Raspberry Pi

2015 年夏のプログラミング・シンポジウムは 1 月の企画委員会でハッカソンをやろうということになった. 幹事会は Raspberry Pi のハックと対戦型 2048 とその他という題を出した.

私としては, 最近 Raspberry Pi に興味を持っていたので, ARM のアセンブリ言語でプログラムを書くのはどうかと考えていた.

ARM については 1990 年ころ, ケンブリッジ大学を訪れたとき, Andy Hopper さんから ARM という CPU があるから使ってみてほしいと言われたのが最初かと思う. しかしその後すっかり忘れていた.

昨年だったか, インターネットで, Martin Richards 君の「Young Persons Guide to BCPL Programming on the Raspberry Pi」[0] を見つけて驚いた. そしてツィッターに書いた.

> 50 年ほど前に BCPL を開発した Martin Richards が「青少年のための Raspberry Pi 上の BCPL プログラミング入門」を書いている (`http://goo.gl/rXOKy2`). 1 つの言語に半世紀も情熱を持ち続けるとは見上げたもの. そのうち読んでみたい.

「青少年のための... 入門」と訳したのは, 元の題が「Young Persons Guide to BCPL Programming on the Raspberry Pi」であり, Benjamin Britten の曲「The Young Person's Guide to the Orchestra」が日本では「青少年のための管弦楽入門」といわれているからだ.

BCPL は Martin Richards が 1967 年頃に設計した言語である. その元はケンブリッジ大学とロンドン大学がコンパイラ記述用に開発した CPL で, Combined Programming Language の略である. その Basic 版が BCPL ということになっている. また BCPL から C 言語が生まれたこともよく知られている.

BCPL をケンブリッジで開発した後, Martin Richards は MIT の Project MAC に滞在したので, MIT でも BCPL は使えるようになった. 私が BCPL に出逢ったのは 1973 年から 1 年 MIT にいた時であった. 私の持っている BCPL の最初のマニュアルは Project MAC のドキュメントである.

[1] IIJ 技術研究所
[a)] eiiti.wada@nifty.com

ところで先頃, IIJ の研究所で Raspberry Pi を何台か購入したので, その 1 台を使わせてもらうことにした.
とにかく Mathematica が動くのがすばらしい. しかし私としては ARM のアーキテクチャに関心があったので, 早い段階から ARM のアセンブリ言語でプログラムを書いてみたくなっていた.

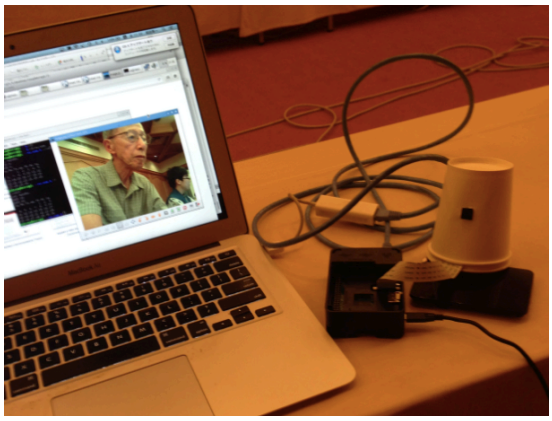
Raspberry Pi にカメラをつける

9 月 4 日から下呂温泉の水明館でシンポジウムが始まった. 参加者がそれぞれ持参したもので店を広げ始めた. 原田君が Raspberry Pi 用のカメラがあるというので, とりあえずはそのカメラを 1 台借り, 自分の Pi に取り付けてみた. なんとも簡単であった.
そうして撮影した 1 枚が下の写真である.

存外よく撮れていた. 次の写真は Pi で撮影した写真を MacBook で眺めているところで, iPhone で撮ったものである.
Pi のカメラの固定には手こずったが, 結局はこの写真のように, 紙コップにカメラのサイズ (9 ミリ × 9 ミリ) の穴を開け, そこからカメラを覗かせ, 内側からもう 1 個の紙コップを押し込んで, カメラを 2 個の紙コップで挟み込むのである.

ARM のアーキテクチャ

ARM は Risc だが, 命令幅が 32 ビットで, 水平型マイクロプログラムの様相を持つ. 16 ビット命令のモードもある.



32 ビットのレジスタを 16 個持つ. 16 個のうち最後の 2,3 個はプログラムカウンターなど用途が決まっている. アドレスはバイト単位なので, 32 ビットの語を取り出すには 4 の倍数のアドレスを使う. 記憶装置とレジスタの間のデータ転送は ldr, str だけ.
cmp(比較命令) が CPSR(Current Program Status Register, NZCV ビット) をセットするほか, 加減乗算命令でも, adds のように s を付けて CPSR をセットできる.
命令には, eq, ne, pl, mi など条件を付け, 実行するしないが指定できる. これにより数命令先へのジャンプはほとんどいらなくなった.
ARM には除算命令がない. そういえば Sparc や MIPS が現れた頃, UtiLisp を移植するのに必要ということで, Sparc や MIPS に除算のサブルーチンを書いた記憶があるので, それらの RISC マシンにも除算はなかったのかも知れない.
そうしてみると TAOCP の計算機 MMIX には除算も剰余をとる命令もあるが, この方が特殊なのかな.

`cal` のプログラム

アセンブリ言語で書いてみようとした最初が `cal` である. Unix のコマンドではある年, ある月のカレンダーが出力できて便利なものだ.
例えば今年の 9 月のカレンダーは, `cal 9 2015` とすると, 下のような出力を得る.

ただし cal がグレゴリオ暦になるのは 1752 年 9 月である.

```
% cal 9 2015
   September 2015
Su Mo Tu We Th Fr Sa
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30
```

私は以前, MMIX, Haskell, PostScript, Scheme, Teco, Tex などのプログラム言語で cal を書いた. その話は 2007 年の夏のプログラミング・シンポジウムの報告書にある.[1][2] Raspberry Pi では BCPL でも書いた.

今回はそれを ARM で書くのである. とりあえずグレゴリオ暦の範囲内とする. このプログラムでは除算や剰余の計算が必要なので, 正の被除数を正の除数で割る簡単な除算ルーチンを作ることにした. 簡単なというのは減算を繰り返すのである. 従って商が大きくなるのは嬉しくない.

グレゴリオ暦では 400 年で週日を繰り返すから, 西暦を 400 で割った剰余の年について計算を進めることにした. (以下 R400 という.) 2015 年なら 15 年を対象とする.

R400 を 100 で割ったり 4 で割ったりしなければならない. 4 で割る方は除算ルーチンは必要ないが, これもサブルーチンを呼ぶように書いてある.

R400 を 100 で割った商と剰余を Q100, R100, 4 で割った商と剰余を Q4, R4 と書くと, 閏年は R100=0 なら R400=0, そうでないなら R4=0 で判定する.

400 年周期の最初から R400 年の最後までの日数は $\mathtt{Q400}\times 365-\mathtt{Q100}+\mathtt{Q4}$ だがその翌日 (R400+1 年 1 月 1 日) の週日は 7 で割った剰余. $365 \bmod 7 = 1$ だから ($\mathtt{Q400}-\mathtt{Q100}+\mathtt{Q4}+$ 定数) mod 7 でよい. その年の m 月 1 日の週日は, 12 月なら $31 \bmod 7 = 3$ を引く, 11 月なら $(31+30) \bmod 7 = 5$ を引く, ... のように得られる. 1 月と 2 月の引く数は閏年の修正が必要だ.

こうして R400 年 m 月 1 日の週日が決まったら, その直前の日曜に相当する日付とその 6 週間後の日付を計算し, それを 1 行に 7 日づつ出力する. ただし日付 d が $d < 1$ と $d > n$(n は月末の日付) なら空白を出力するという方針のプログラムを書く.

```
00 .data
01 base:   .word 0          /*year*/
02         .word 0          /*month*/
03         .byte 6,2,2,5,0,3,5,1,4,6,2,4
04         .byte 31,28,31,30,31,30,31,31,30,31,30,31
05 s0:     .asciz " Su Mo Tu We Th Fr Sa\n"
06 s1:     .asciz "   "
07 s2:     .asciz " %2d"
08 s3:     .asciz "\n"
09 s4:     .asciz "%d %d"
10 .text
11 divmod: mov r2,#0        /*r0<-r0/r1,r1<-r0%r1*/
12         add r2,r2,#1
13         subs r0,r0,r1
14         bge divmod+4
15         add r1,r0,r1
16         sub r0,r2,#1
17         bx lr
18         .global main
19 main:   mov r8,lr        /*save lr*/
20         ldr r1,ba
21         add r2,r1,#4
22         ldr r0,s4a
23         bl scanf         /*read year month*/
24         ldr r3,ba
25         ldr r0,[r3]
26         mov r1,#400
27         bl divmod
28         mov r4,r1        /*R400*/
29         mov r0,r1
30         mov r1,#4
31         bl divmod
32         mov r7,r0        /*Q4*/
33         mov r5,r1        /*R4*/
34         mov r0,r4
35         mov r1,#100
36         bl divmod
37         mov r6,r0        /*Q100*/
38         cmp r1,#0        /*R100*/
39         moveq r1,r4      /*R400*/
40         movne r1,r5      /*R4*/
41         cmp r1,#0        /*leap?*/
42         mov r0,#5
43         streqb r0,[r3,+#8]
44         mov r0,#1
45         streqb r0,[r3,+#9]
46         mov r0,#29
47         streqb r0,[r3,+#21]
48         mov r0,#1
```

```
49          add r0,r0,r4    /*R400*/
50          sub r0,r0,r6    /*Q100*/
51          add r0,r0,r7    /*Q4*/
52          ldr r6,[r3,+#4] /*month*/
53          add r1,r6,#7
54          ldrb r1,[r3,r1]
55          add r0,r0,r1
56          mov r1,#7
57          bl divmod
58          mov r4,r1       /*day of week of 1st*/
59          rsb r5,r4,#1
60          rsb r7,r4,#43
61          add r6,r6,#19   /*month leng*/
62          ldrb r6,[r3,r6]
63          ldr r0,s0a      /*week names*/
64          bl printf
65          mov r4,#7
66 l0:      cmp r5,#1       /*first day*/
67          rsbgts r0,r5,r6 /*last day*/
68          mov r1,r5       /*r1<-day*/
69          ldrpl r0,s2a    /*print day format*/
70          ldrmi r0,s1a    /*print blank format*/
71          bl printf
72          add r5,r5,#1
73          subs r4,r4,#1
74          ldreq r0,s3a    /*end of a week*/
75          bleq printf     /*newline*/
76          moveq r4,#7
77          cmp r5,r7       /*end of cal?*/
78          bmi l0
79          mov lr,r8       /*restore lr*/
80          mov r0,#1
81          bx lr
82 ba:      .word base
83 s0a:     .word s0
84 s1a:     .word s1
85 s2a:     .word s2
86 s3a:     .word s3
87 s4a:     .word s4
```

行 00 .data ここからデータ領域という指示.
行 01,02 年と月を格納する場所 4 バイトずつ.
行 03 1 月から順に次の年初の曜日とその月の 1 日の曜日の差 12 月は 31 日なので 4 週と 3 日余るが, −3 mod 7=4 のような値のリスト. 以下 rs という.
行 04 各月の長さ. 以下 ms という.
行 05〜09 scanf, printf の書式.
行 10 プログラム部分の開始.
行 11〜17 除算サブルーチン r0 を r1 で割り商を r0, 剰余を r1 に置く.
行 19 主ルーチンの入り口, 帰り番地を r8 へ退避する.
行 20 year を入れる場所を r1 へ.
行 21 month を入れる場所を r2 へ.
行 22 scanf の書式の番地を r0 へ.
行 23 scanf を呼ぶ.
行 24 データ領域のベースアドレスを r3 へ.
行 25 year を r0 へ.
行 26 400 を r1 へ.
行 27 除算.
行 28 400 で割った剰余 R400 を r4 へ.
以下 R400 を 100 と 4 で割った商と剰余を求める.
行 38 100 で割った剰余を 0 と比較.
それが 0 か否かにより, 400 と 4 との剰余を r1 に置く (条件つき mov.)
行 41 r1 が 0 なら閏年 (閏年の判定には 100 で割った剰余 R100=0 なら 400 で割った剰余 R400=0, 0 でないなら 4 で割った剰余 R4=0 かで判定できる.)
行 42〜47 r1=0 なら閏年なので, rs の 1 月 2 月と ms の 2 月を修正する.
行 48〜58 その月の 1 日の週日は (1+R400−Q100+Q4+rs[月])mod 7. 2015 年 9 月は 2.
行 59 1 日以前の日曜の日付. 9 月は −1.
行 60 6 週後の日曜の日付. 9 月は 41.
行 62 その月の最後の日付. 9 月は 30.
出力用の値が設定できたので出力に移る.
行 63,64 曜日の名を出力.
行 65 r4 は 1 週間を数えるカウンタ.
行 66 日付が 1 より大きいか.
行 67 そうなら最後の日付より小さいか.
行 68 日付を r1 に置く.
行 69,70 比較の結果でいづれかの書式を r0 に置く.
行 72 日付を 1 増やす.
行 73 週のカウンタを減らす.
行 74〜76 1 週間が終わったら改行しカウンタをリセット.
行 77,78 最後になっていなければ l0 へ戻る.
行 79 lr を戻す.
行 80,81 プログラムの戻り値を入れて終わる.
行 82〜87 定数の番地.

このプログラムを Raspberry Pi で動かすには,

```
as -o cal.o cal.s
gcc -o cal cal.o
./cal
```

で起動し

```
2015 9
```

のように年と月を入力する.

このプログラムでは, データの場所にはラベルがあるが, プログラムの部分には殆どラベルが見られない. divmod はサブルーチンの入り口, main はプログラム全体の入り口, あとカレンダー出力のループの l0 があるだけである. 従ってジャンプ命令もほとんどない. そういうことでは ARM のアーキテクチャは気持ちがいいといえる.
このプログラムは `http://www.iijlab.net/~ew/cal.s` に置いてある.

引き放し除算

除算にはいろいろな哲学がある. 私の好きなのは, 剰余を除数と同符号にとるものである. 負の除数で割るときは, 剰余が 0 にならないが, 負で割ることはそんなにないからこれでいいであろう. この除算が嬉しいのは多倍長の被除数を単精度の除数で割るときである. つまり先頭の商の後は正の商を得なければならないからだ. 私が以前実装した iPhone 用の電卓の除算もそうしてある [3].
l ビットレジスタでの除数 d, 剰余 r, 商 q, それらに対する被除数 n の範囲と, $l = 4$ での値を次に示す.
$0 < d$ の時, $0 \le d < 2^{l-1}$, $0 \le r < d$, $-2^{l-1} \le q < 2^{l-1}$, $-2^{l-1} \times d \le n < 2^{l-1} \times d$.
$l = 4$ の時, $0 \le d < 8$, $0 \le d < 8$, $-8 \le q < 8$, $-8 \times d \le n < 8 \times d$.
$d < 0$ の時, $-2^{l-1} \le d < 0$, $d \le r < 0$, $-2^{l-1} \le q < 2^{l-1}$, $2^{l-1} \times d \le n < -2^{l-1} \times d$.
$l = 4$ の時, $-8 \le d < 0$, $d \le r < 0$, $-8 \le q < 8$, $8 \times d \le n < -8 \times d$.
$l = 4$ での最大最小あたりの n に対する q, r は次のようだ.

	n, q, r	n, q, r	n, q, r	$n, 1, r$
$d = 7$	$-56, -8, 0$	$-55, -8, 1$	$54, 7, 5$	$55, 7, 6$
$d = 6$	$-48, -8, 0$	$-47, -8, 1$	$46, 7, 4$	$47, 7, 5$
$d = 2$	$-16, -8, 0$	$-15, -8, 1$	$14, 7, 0$	$15, 7, 1$
$d = 1$	$-8, -8, 0$	$-7, -7, 0$	$6, 6, 0$	$7, 7, 0$
$d = -1$	$-8, 7, -1$	$-7, 6, -1$	$6, -7, -1$	$7, -8, -1$
$d = -2$	$-16, 7, -2$	$-15, 7, -1$	$14, -8, -2$	$15, -8, -1$
$d = -7$	$-56, 7, -7$	$-55, 7, -6$	$54, -8, -2$	$55, -8, -1$
$d = -8$	$-64, 7, -8$	$63, 7, -7$	$62, -8, -2$	$63, -8, -1$

引き放し除算の方法はこうだ. ARM のレジスタは 32 ビットだが, 4 ビットだと思って書いた図が下だ. 左は 55 割る 7, 右は 47 割る 7.

```
0←[0 0 1 1][0 1 1 1]      0←[0 0 1 0][1 1 1 1]
1  [0 1 1 0][1 1 1 0]      1  [0 1 0 1][1 1 1 0]
2 -[0 1 1 1]              2 -[0 1 1 1]
3←[1 1 1 1][1 1 1 0]      3←[1 1 1 0][1 1 1 0]
4  [1 1 1 1][1 1 0 0]      4  [1 1 0 1][1 1 0 0]
5 +[0 1 1 1]              5 +[0 1 1 1]
6←[0 1 1 1][1 1 0 0]      6←[0 1 0 0][1 1 0 0]
7  [1 1 0 1][1 0 0 0]      7  [1 0 0 1][1 0 0 0]
8 -[0 1 1 1]              8 -[0 1 1 1]
9←[0 1 1 0][1 0 0 1]      9←[0 0 1 0][1 0 0 1]
10 [1 1 0 1][0 0 1 0]     10 [0 1 0 1][0 0 1 0]
11-[0 1 1 1]              11-[0 1 1 1]
12 [0 1 1 1][0 0 1 1]     12 [1 1 1 0][0 0 1 1]
13*[0 1 1 0][0 1 1 1]     13*[0 1 1 1][0 1 1 1]
                          14*[0 1 0 1][0 1 1 0]
```

行 0. 被除数が二進法で置いてある. それを左へ 1 ビットシフト ← する.
行 1. 被除数と行 2 にある除数の符号を較べる.
行 2. 符号が同じなら被除数から除数を引き, 異なれば足す.
行 3〜12. 被除数を左へ 1 ビットシフトする.
シフトする前の被除数と除数の符号が同じなら被除数から除数を引き −, 被除数の最下位に 1 を置く. 異なれば足す +.
行 13. 右のレジスタを 2 倍して 1 を足す. 左のレジスタに剰余. 右のレジスタに商がえられる. 剰余と除数の符号が異なれば, 剰余に除数を足し, 商から 1 を引く (補正*).
これを 32 ビットレジスタで実行するのが下のプログラムである. 除算サブルーチンは 08 行から 33 行まで.

```
00 .data
01         .balign 4
```

```
02 m0:      .word 0x7fffffff        /*dividend lo*/
03 m1:      .word 0x3fffffff        /*dividend hi*/
04 m2:      .word 0x7fffffff        /*divisor*/
05 s1:      .asciz "quotient %8x remainder %8x\n"
06 .text
07          .balign 4
08 div:     adds r0,r0,r0   /*left shift*/
09          adcs r3,r1,r1
10          bvs eret        /*overflow?*/
11          eors r1,r3,r2
12          subpl r1,r3,r2
13          addmi r1,r3,r2
14          eors r3,r1,r3
15          bpl eret        /*overflow*/
16          mov r4,#30
17 loop:    adds r0,r0,r0   /*left shift r1r0*/
18          adc r3,r1,r1
19          eors r1,r1,r2   /*comp sign*/
20          subpl r1,r3,r2
21          addpl r0,r0,#1
22          addmi r1,r3,r2
23          subs r4,r4,#1
24          bpl loop        /*end loop?*/
25          add r0,r0,r0
26          add r0,r0,#1
27          eors r3,r1,r2
28          addmi r1,r1,r2
29          submi r0,r0,#1
30          bx lr           /*return*/
31 eret:    mov r1,r2,LSR #31/*overflow error*/
32          sub r1,r1,#1
33          bx lr           /*return*/
34          .global main
35 main:    str lr,[sp,#-4]!/*save link*/
36          ldr r0,m0a
37          ldr r0,[r0]
38          ldr r1,m1a
39          ldr r1,[r1]     /*nhi<-0*/
40          ldr r2,m2a
41          ldr r2,[r2]     /*divisor*/
42          bl div          /*call div*/
43          mov r2,r1       /*remainder r1*/
44          mov r1,r0       /*quotienti r0*/
45          ldr r0,s1a
46          bl printf
47          ldr lr,[sp],#4  /*restore link*/
48          mov r0,#1
49          bx lr
50 m0a:     .word m0
51 m1a:     .word m1
52 m2a:     .word m2
53 s1a:     .word s1
```

行 08 ここから除算サブルーチン. 被除数は上位が r1, 下位が r0, 除数が r2 にある.

行 08,09 r1,r0 を繋げて左シフト. r0+r0 を r0 に置き, adds でキャリーを cpsr に置く. adc でそのキャリーを足しながら r1+r1 を r3 に置く. adcs なのでオーバーフローがあればセットする.

行 10 被除数の左端の 2 ビットが 01, 10 だとオーバーフローが起き, bvs でジャンプ.

行 11 除数と被除数の符号を較べる.

行 12,13 符号が同じなら被除数から除数を引く, 異なれば足す.

行 14 その加減算の前後の被除数の符号を較べる. 変っていなければオーバーフロー.

行 16 カウンタ r4 に 30 をセット.

行 17,18 被除数を 1 ビット左シフト.

行 19 符号を較べる.

行 20〜22 同じなら除数を引き, r0 に 1 を足す. 異なれば足す.

行 23,24 カウンタを減らす.

行 25,26 補正

行 27〜29 補正

行 30 サブルーチンから帰る.

行 31〜33 オーバーフローのとき, 除数が正なら −1, 負なら 0 を持って帰る.

以下は除算のテスト. 各行 左から被除数上位 被除数下位 除数 商 剰余. 最上段は r1 が `3fffffff`, r0 が `7fffffff` で, r2 の除数が `7fffffff` のとき, 商が `7fffffff`, 剰余が `7ffffffe` と読む.

```
3fffffff 7fffffff 7fffffff 7fffffff 7ffffffe
c0000000 80000000 7fffffff 80000000        0
c0000000 7fffffff 80000000 7fffffff ffffffff
3fffffff 80000000 80000000 80000000 80000000
```

参考文献

[0] http://www.cl.cam.ac.uk/~mr10/bcpl4raspi.pdf

[1] 和田英一, MMIX cal ハック, 2007 年夏のプログラミング・シンポジウム「First Programming Language プログラミング言語の実力と美学」報告集 http://www.iijlab.net/~ew/cal.pdf

[2] 和田英一, いろいろなプログラム言語による cal のプログラム, 2007 年夏のプログラミング・シンポジウム「First Programming Language プログラミング言語の実力と美学」報告集 http://www.iijlab.net/~ew/prog2.pdf

[3] http://www.iijlab.net/~ew/hhcmanualjp.pdf