Tutorial #3
Hilary Term Weeks 8 and 9
Floating Point

Work in groups of 6 students using one of the whiteboards.
Complete the exercises in your own time if necessary.
Revise the exercises to prepare for the next lab.

1 Floating-Point Numbers

(a) Normalise the following decimal floating point numbers:

(i) 456.789×10^3

(ii) 0.000425×10^{-2}

(b) Convert the following binary floating point numbers to decimal:

(i) 1.1

(ii) 1000.0101

(iii) 0.1111

(c) Convert the following decimal floating point numbers to binary:

(i) 15.25

(ii) 12.1875

(iii) 8.9

(iv) 0.08

(d) Normalise each of the binary floating point numbers in part (c).

(e) Show how you would store each of the normalised floating point numbers from part (d) as a 32-bit word using the IEEE 754 standard.

2 Floating-Point Arithmetic

Decode, align, add, normalise and re-encode each of the following floating point numbers encoded using the IEEE 754 standard, using the approach outlined in lectures.

(i) a=`0x3FA00000`, b=`0x3F400000`

(ii) a=`0x41C40000`, b=`0x41960000`

(iii) a=`0x41A60000`, b=`0x3E400000`

3 Floating-Point Decoding

Write an ARM Assembly Language subroutine which, given an IEEE-754 value in R0, returns the fraction in R0 and the exponent in R1. The fraction and exponent should be returned as signed two's complement values.

4 Floating-Point Encoding

Write an ARM Assembly Language subroutine which, given a a floating point value expressed as a fraction in R0 and an exponent in R1, returns the IEEE-754 representation of the value. The fraction will be passed as two's complement value and will not necessarily be normalised. The exponent will also be passed as a two's complement value.

Complete the exercises in your own time if necessary!

CS1022 Tutorial #3 SOLUTION
Hilary Term Weeks 8 and 9
Floating Point

1 Floating-Point Numbers

(a) (i) 4.56789×10^5

(ii) 4.25×10^{-6}

(b) (i) 1.5

(ii) 8.3125

(iii) 0.9375

(c) (i) 1111.01

(ii) 1100.0011

(iii) 1000.111001100110011

(iv) 0.00010100011110101110 ...

(d) (i) 1.11101×2^3

(ii) 1.0100011×2^3

(iii) $1.00011100110011001100110011 \ldots \times 2^3$

(iv) $00001.0100011110101110 \ldots \times 2^{-4}$

(e) (i) 0x41740000

(ii) 0x41430000

(iii) 0x410e6666

(iv) 0x3da3d70a

2 Floating-Point Arithmetic

(i) $1.25 + 0.75 = 2$ (0x40000000)

(ii) $24.5 + 18.75 = 43.25$ (0x422d0000)

(iii) $20.75 + 0.1875 = 20.9375$ (0x41a78000)

3 Floating Point Decode

```
1  ;
2  ; fpdecode
3  ; decodes an IEEE 754 floating point value to the signed (2's complement)
4  ; fraction and a signed 2's complement (unbiased) exponent
5  ; parameters:
6  ;       r0 - ieee 754 float
7  ; return:
8  ;       r0 - fraction (signed 2's complement word)
9  ;       r1 - exponent (signed 2's complement word)
10 ;
11 fpdecode
12         STMFD   sp!, {r4-r5, lr}
13
14         LDR     r4, =0x7F800000
15         AND     r1, r0, r4              ; e = value & 0x7F800000
16         MOV     r1, r1, LSR #23         ; e = e >> 23
17         SUB     r1, r1, #127            ; e = e - bias
18
19         AND     r5, r0, #0x80000000     ; s = value & 0x80000000
20
21         LDR     r4, =0x007FFFFF
22         AND     r0, r0, r4              ; f = value & 0x007FFFFF
23         ORR     r0, r0, #0x00800000     ; f = f | 0x00800000 // hiddden bit
24
25         CMP     r5, #0x80000000         ; if (s == 0x80000000) {
26         BNE     elfNeg1                 ;
27         RSB     r0, r0, #0              ;   f = 0 - f
28 elfNeg1                                 ; }
29
30         LDMFD   sp!, {r4-r5, pc}
```

4 Floating Point Encode

```
1  ;
2  ; fpencode
3  ; encodes an IEEE 754 value using a specified fraction and exponent
4  ; parameters:
5  ;       r0 - fraction (signed 2's complement word)
6  ;       r1 - exponent (signed 2's complement word)
7  ; result:
8  ;       r0 - ieee 754 float
9  ;
10 fpencode
11         STMFD   sp!, {r4-r6, lr}
12
13         ; handle signed fraction
14
15         CMP     r0, #0                  ; if (fr < 0) {
16         BGE     elfNeg2                 ;
17         MOV     r4, #1                  ;   sign = 1
18         RSB     r0, r0, #0              ;   fr = 0 - fr
19 elfNeg2                                 ; }
20
21         ; count leading zeros for normalisation
22         ; note: no CLZ instruction on the ARM v4 architecture
23
24         MOV     r5, r0                  ; tmp = fr
25         MOV     r6, #0                  ; clz = 0
26 doClz
27         MOVS    r5, r5, LSL #1          ; while ( (tmp = tmp << 1) does not carry out)
28         BCS     eDoClz                  ; {
29         ADD     r6, r6, #1              ;  clz++;
```

```
30          B       doClz                   ; }
31  eDoClz
32
33          ; normalise
34
35          SUBS    r6, r6, #8              ; shift = clz - 8
36          BLO     elsNormRt
37          BEQ     elfNorm
38
39          ; normalise left                ; if (shift > 0) {
40
41          MOV     r0, r0, LSL r6          ;   fr = fr << shift
42          SUB     r1, r1, r6              ;   er = er - shift
43          B       elfNorm                 ; }
44
45  elsNormRt
46
47          ; normalise right               ; else {
48
49          RSB     r6, r6, #0              ;   shift = -shift
50          MOV     r0, r0, ASR r6          ;   fr = fr >> shift
51          ADD     r1, r1, r6              ;   er = er + shift
52
53  elfNorm                                 ; }
54
55          ; encode parts
56
57          ; remove hidden bit
58
59          BIC     r0, r0, #0x00800000     ; result = fr & 0xFF7FFFFF
60
61          ; insert sign
62
63          ORR     r0, r0, r4, LSL #31     ; result = result | sign << 31
64
65          ; insert biased exponent
66
67          ADD     r1, r1, #127            ; er = er + bias
68          ORR     r0, r0, r1, LSL #23     ; result = result | er << 23
69
70          LDMFD   sp!, {r4-r6, pc}
```