

浮動小数点形式に関する課題

1. はじめに

講義では、5ビット（符号1ビット、指数部2ビット、仮数部2ビット）の浮動小数点形式を例に解説した。本課題では、16ビット（符号1ビット、指数部7ビット、仮数部8ビット）の浮動小数点形式に対するアセンブリ・プログラミングを通して、浮動小数点演算 [1] に関する理解を深める。

課題は3グループに分かれている。課題1から5までがFloat16の出力に関する課題、課題6から8までがFloat16の入力に関する課題、課題9が総括した課題である。Cプログラムを作成してから課題に取り掛かるとよいだろう。

2. 浮動小数点形式 Float16

この課題で扱う浮動小数点形式 Float16 を図1に示す。バイアスは64する。符号ビット S 、指数部 e 、仮数部 m とすると、Float16 形式で表現できる数値は式 (1) のとおり。IEEE754 の単精度で指数部が255の場合は無限大もしくは非数であるが、Float16 では指数部が127であっても正規化数として扱うことにする。

$$\left. \begin{aligned} e=0, m=0 & \quad N=0 \\ e=0, m \neq 0 & \quad N=(-1)^S \times 0.m \times 2^{63} \\ 0 < e < 128 & \quad N=(-1)^S \times 1.m \times 2^{e-64} \end{aligned} \right\} \quad (1)$$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	指数部 e (7ビット)							仮数部 m (8ビット)							

図1 Float16 形式

3. Float16 を扱う C プログラム

数値を扱うアセンブリコードでは、テストデータ^{*1}を用意するのが面倒である。このためテストデータを作成するためのヘッダファイルとプログラムを示す。

3.1 ヘッダファイル float16.h

Float16 形式を C 言語で扱うためのヘッダファイルをリスト1に示す。Float16 形式はビットフィールドで定義してある。バイアスはF16BIASと定義してある。このヘッダファイルを使った関数PrintF16Fmtをリスト2に示す。このプログラムは、Float16 形式を16進数4桁で表示する。

リスト1の12行目に関数PrintF16Fmtのプロトタイプ宣言がある。このようにFloat16を使う関数は、リスト1の13行以降にプロトタイプ宣言を追加すればよい。

関数PrintF16Fmtでは、符号ビットSign、指数部Epart、仮数部Mpartをそれぞれシフト操作と論理和で、変数iに16ビットのFloat16形式を格納している。ヘッダファイルに定義してある各部のビットフィールドと見比べて理解せよ。

次のように各フィールドに値を代入して、PrintF16Formatを呼び出せば、16進数4桁で8324と表示される。

```
Float16 x;
x.Sign=1;
x.Epart=3;
x.Mpart=0x24;
PrintF16Format(x);
```

リスト1 float16.h

```
1 #ifndef float16_h
2 #define float16_h
3
4 #include <stdio.h>
5 #define F16BIAS 64
6 typedef struct Float16 {
7     unsigned int Sign : 1;
8     unsigned int Epart : 7;
9     unsigned int Mpart : 8;
10 } Float16;
11
12 void PrintF16Fmt (Float16);
13
14 #endif /* float16_h */
```

リスト2 PrintF16Fmt.c

```
1 #include "float16.h"
2
3 void PrintF16Fmt (Float16 x)
4 {
5     unsigned int i;
6     i = 0;
7     i = i | (x.Sign << 15);
8     i = i | (x.Epart << 8);
9     i = i | x.Mpart;
10     printf("%04X\n", i);
11 }
```

^{*1} この課題では、動作検証のための入力データと出力データをまとめたものとする。

3.2 テストデータ作成用関数 `FtoF()`

IEEE754 の単精度 (float 型) を Float16 に変換するプログラムを右に示す。IEEE754 と Float16 は浮動小数点数形式の考え方は同じであり、大きな違いは指数部のサイズとバイアスである。仮数部はサイズだけが違う。指数部は Float16 のバイアスに変換し、仮数部は 16 ビットだけ切り出している。C プログラムで `PrintF16Format (FtoF (3.14))` とすれば 3.14 を Float16 形式 4192_{16} に変換できる。ちなみに、IEEE754 で、3.14 は $d00f4940_{16}$ である。式 (1) で示したとおり、IEEE754 にある無限大は考慮していない。

リスト 3 ftof.c

```

1 #include "float16.h"
2 // Convert IEEE754 to F16Float
3 Float16 FtoF(float f)
4 {
5     Float16 f16;
6     int i, e;
7     i = *(int *)&f;
8     f16.Sign = (i >> 31) & 0x00000001L;
9     f16.Mpart = (i & 0x007FFFFFFF) >> 15;
10
11     e = ((i & 0x7F800000L) >> 23);
12     if (e == 0) f16.Epart = 0;
13     else f16.Epart = e - 127 + F16BIAS;
14     return f16;
15 }
```

4. 課題

課題で作成するアセンブリ・コードは、QtSPIM で実行できるコードとすること。

課題 1 実数 2.71828 を IEEE754 の単精度と Float16 に手計算で変換せよ。手計算の結果は、`PrintF16Format` 関数と `FtoF` 関数を使って検証せよ。

課題 2 Float16 形式で正数の最大値と、0 でない最小値を 10 進数と 2 進数で示せ。

課題 3 右に示した Float16 形式の仮数部だけを 10 進数に変換する C プログラムをアセンブリ・コードに変換せよ。アセンブリでの関数名は `MpartP` とせよ。関数 `printf()` による数字の出力については、配布した `fact-loop.s` を参照せよ。詳細は、QtSPIM の Help で System Calls に記載されている。

引数 Float16 `x` については、レジスタ `$a0` の下位 16 ビットに Float16 形式で浮動小数点が格納されているとせよ。4 行目は、変数 `m` を `$s0` に割り当てたとすれば、`andi $s0, $a0, 0x00ff` とすればよい。6 行目はシフト操作だけで 10 倍を計算している。

リスト 4 MpartPrint.c

```

1 void MpartPrint(Float16 x)
2 {
3     unsigned int m;
4     m = x.Mpart;
5     while (m != 0) {
6         m = (m << 1) + (m << 3);
7         printf("%d", m >> 8);
8         m = m & 0x00ff;
9     }
10 }
```

課題 4 Float16 形式を 10 進数で出力するアセンブリ・プログラムを作成せよ。アセンブリでの関数名は `F16P` とせよ。仮数部の出力は、課題 3 の解答を使用すればよい。出力例を右表に示す。指数部は、例のように 2 のべき乗で出力すればよい。文字列 `"*2^("&"")` の出力は配布した `fact-loop.s` を参照せよ。詳細は、QtSPIM の Help で System Calls に記載されている。

表 1 出力例

Float16 形式	出力
0000	+0
0027	+0.15234375*2 ⁽⁻⁶³⁾
AAAA	-1.6640625*2 ⁽⁻²²⁾
8080	-0.5*2 ⁽⁻⁶³⁾
8888	-1.53125*2 ⁽⁻⁵⁶⁾

課題 5 課題 4 で作成したアセンブリ・コードの動作を検証するためにテストデータを作成せよ。テストデータの例を右表に示す。テストデータには、正しい結果が得られる入力データと、その出力データを含める。ゼロ、課題 2 の解答、非正規化数など、プログラムの動作を検証する上で適切な入力データとその出力データも含めること。

なお、本来ならば、テストデータは課題 4 と一緒に作成する。

表 2 テストデータ (例)

入力	出力	コメント
0000	+0	正のゼロ
8000	-0	負のゼロ
		正の最大値
		0 でない最小値
		正の非正規化
		負の非正規化
以下略		

課題 6 符号なし整数 (32 ビット) を 10 で除算する C プログラムを右に示す。この C プログラムをアセンブリ・コードに変換せよ。アセンブリでの関数名は `div10` とせよ。

除算命令は遅いので、シフト命令と加算命令で実装する方法がある。簡単な例として、205 倍して 11 ビット左シフト ($2^{11} = 2048$) すれば $0.1 (\approx \frac{205}{2048})$ 倍することができる。この C プログラムでは、32 ビットの精度で $0.1 (= \frac{858993459}{8589934592})$ 倍している [2]。

リスト 5 Div10.c

```

1 unsigned int Div10(unsigned int k)
2 {
3     unsigned int j, m, n, p, q, r;
4     j = (k + 1) << 1;
5     m = j + (j << 1);
6     n = m + (m >> 4);
7     p = n + (n >> 8);
8     r = p + (p >> 16);
9     q = r >> 6;
10    return q;
11 }
```

課題 7 文字列を 32 ビットの固定小数点形式に変換する方法について考える。例えば、文字列 “27.75” を入力すると、符号なし整数として $(001BC000)_{16}$ を出力する。中央に小数点があり、上位 16 ビットが整数、下位 16 ビットが少数である。右のプログラムは、整数部 `x` を 13 行目の `return` で 16 ビット左シフトしている。

数字の文字列を 32 ビットの固定小数点形式に変換するアセンブリ・コードを作成せよ。課題 6 で作成した `div10` を使うこと。アセンブリでの関数名は、`StoF32` とせよ。

QtSPIM での入出力については、QtSPIM の Help の System Calls に記載されている。

リスト 6 StoFix32.c

```

1 unsigned int StoFix32(char *s)
2 {
3     // Integer part
4     int x = 0;
5     while (*s != '.') {
6         x = (x << 1) + (x << 3) + *s - '0';
7         s = s + 1;
8     }
9     // Fractional part
10    int y = 0;
11    // your code
12
13    return (x << 16) + (y & 0x00ff);
14 }
```

課題 8 課題 7 で変換した 32 ビットの固定小数点形式を、式 (1) に従うように、正規化して Float16 形式に変換するアセンブリ・コードを作成せよ。また、このアセンブリ・コードに対するテストデータを示せ。テストデータを示すことにより、作成したアセンブリ・コードが正しく動作する引数の範囲が分かるようにせよ。

課題 9 Float16 同士を加算し、結果を出力するアセンブリ・コードを作成せよ。教科書の 195 頁の「浮動小数点加算」と 198 頁の例題「2 進の浮動小数点加算」が参考になる。

5. おわりに

課題レポートの提出については次回の授業で伝える。課題をとおして、QtSPIM の使い方に習熟しておくこと。課題について、不明な点があれば、UNIPA の Q&A を使って問い合わせること。

参考文献

- [1] ジョン・L. ヘネシー、デイビッド・A. パターソン：コンピュータの構成と設計 第 5 版上, pp. 190–198, 日経 BP 社 (2015).
- [2] Vowels, R. A.: Divide by 10, *Australian Computer Journal*, Vol. 24, No. 3, pp. 81–85 (1992).