電子計算機 **3E**

3E　2003.9.1

本節の授業のテーマ

本日の事業のテーマは、以下のとおりです。

(1)算術演算
- 算術加算(ADDA)
- 算術減算(SUBA)

(2)論理演算
- 論理加算(ADDL)
- 論理減算(SUBL)

本日の授業のゴールは、以下のとおり。
- 算術演算が理解できる。
- 算術演算と論理演算(加算・減算)の違いが理解できる。

教科書と併用して授業を進めます。プリントには、教科書に載っていないことのみを記述します。したがって、授業はプリントと教科書の両方を見ながら進めます。プリントのセクション番号は、前回の続きです。機械語命令が終わるまでその番号は連続します。
しばらく、命令の説明は、このスタイルで授業を進めます。

3. 算術演算

整数(-32768〜32768)の加算と減算命令です。乗除算はCOMET IIのハードウェアーに有りません。そのことについては後で述べます。

3.1 算術加算(ADDA)

2個の16ビットの値を加算する命令です。16進数でも10進数でも正しく計算されます。ただし符号付ということに注意してください。10進数で内容を指定したとき、その16ビットの表現は2の補数です。

例1 レジスタ同士の加算。GR0←GR0+GR1

```
ADDA  GR0,GR1
```

例2 レジスタとメインメモリーの値の加算。Aはメモリーのアドレス(バンチ)を表します。GR0←GR0+(アドレスAの内容)

```
ADDA  GR0,A
```

例3 レジスタとアドレス修飾された実行アドレスの内容の加算。実行アドレスは、(A+GR1の内容)となります。GR0←GR0+(アドレスA+GR1の内容)

```
ADDA  GR0,A,GR1
```

例4 リテラルを使って、加算をします。GR0←GR0+5

```
ADDA  GR0,=5
```

3.2 算術減算(SUBA)

減算の演算で、動作は算術加算と同じです。

例1 レジスタ同士の減算。GR0←GR0-GR1

```
SUBA  GR0,GR1
```

例2 レジスタとメインメモリーの値の減算。Aはメモリーのアドレス(バンチ)を表します。GR0←GR0-(アドレスAの内容)

```
SUBA  GR0,A
```

例3 レジスタとアドレス修飾された実行アドレスの内容の減算。実行アドレスは、(A+GR1の内容)となります。GR0←GR0-(アドレスA+GR1の内容)

```
SUBA  GR0,A,GR1
```

例4 リテラルを使って、減算をします。GR0←GR0-5

```
SUBA  GR0,=5
```

3.3 乗除算はどうするの?

ここで、賢明な諸君は、4 則演算の残りの 2 つ、乗除算はどうしたのという疑問が湧くはずです。湧いて欲しい。

最近の CPU は、これら乗除算のハードウェアーが実装されており、それに対応する機械語命令があります。しかし、単純なハードウェアの COMET II には、そのハードウェアはなく、当然機械語命令も有りません。そのため、ソフトウェアーでその仕組みを実現します。これは、後に学習しますが、ちょっとだけ方法を示します。

3.3.1 乗算

算術加算を使う方法

例えば、10×3 を計算する場合、10+10+10 のように必要な回数だけ足し合わせて乗算を行います。このように算術加算を用いて乗算が出来ます。ただし、この方法はもっとも原始的で効率の悪い方法です。

ビットシフトを使う方法

もう少し効率の良い方法は、ビットシフトを使う方法です。2 進数では、1 ビット左にシフトすると 2 倍に、右にシフトすると 1/2 になります。10×3 を計算する場合、10×(2+1)を考えます。1 ビットシフトしたものと、元の数を足し合わせます。

この方法は、小学生の時に学習した筆算を用いた掛け算と同じ考えです。

3.3.2 除算

算術減算を使う方法

10/3 の計算は 10 からから 3 を引いていきます。そして、負になったら計算は完了です。すると、商と余りが分かります。算術減算を用いて乗除算が出来ます。

この方法は効率が悪いので、もう少しましな方法を考えましょう。小学生の時に学習した筆算のアルゴリズムを適用すれば効率は良くなります。計算は次のようにします。

$(10)_{10}=(1010)_2$　　$(3)_{10}=(11)_2$

以降の計算順序は、割り算を筆算で行うことを頭に入れて考えてください。

① $(1)_2$ から $(11)_2$ を引きます。しかし、引き算はできません。
② $(10)_2$ から $(11)_2$ を引きます。しかし、引き算はできません。
③ $(101)_2$ から $(11)_2$ を引きます。引き算は可能です。引き算の結果は $(10)_2$ で、その桁に $(1)_2$ が立ちます。
④ 先ほどの残りと次の桁を合わせた $(100)_2$ は引き算可能です。引き算の結果は $(1)_2$ で、その桁に $(1)_2$ が立ちます。
⑤ これ以上桁がないので、計算は完了です。商は $(11)_2=(3)_{10}$ 余りは $(1)_2=(1)_{10}$ となります。

ビットシフトを使う方法

2, 4, 8, 16・・・と 2^n で割る場合のみ、ビットシフトが使えます。残念ながら、それ以外の場合は、先に示した方法で計算するほかないです。

ということで、加算と減算ができれば乗除算は可能です。さらに賢明な諸君は、次の疑問が湧くはずです。湧いて欲しい。三角関数や指数関数などは、どうやって計算しているのか?。三角関数や指数関数は、テイラー展開(マクローリン展開)を使うと 4 則演算に分解できる

ことを学習したはずです。したがって、4 則演算ができれば、それらの関数は計算可能です。高級言語のコンパイラーは、これらの関数を 4 則演算に変換して計算しています。

4. 論理演算(加算、減算)

論理演算と言う名前がついていますが、実際は符号無整数の演算です。算術演算が第 15 ビットを符号ビットとして取り扱うのに対して、論理演算では数はすべて正の数です。論理加算は、対象となるデータを 16 ビットの符号無整数として取り扱い計算します。したがって、扱える数字の範囲は、0～65535 までです。

4.1 論理加算(ADDL)

教科書の例(List4-6)を用いて、算術加算(ADDA)と論理加算の違いを説明します。汎用レジスタの値を GR0=#7FFF、GR1=1 とします。それぞれの汎用レジスタの値は、表の 1 のようになります。

それぞれの整数は、符号無整数の場合、

#7FFF $=(7FFFF)_{16}$
$=(0111\ 1111\ 1111\ 1111)_2$
$=(32767)_{10}$

1=$(0000\ 0000\ 0000\ 0001)_2$

符号あり整数の場合、いずれも第 15 ビットが 0 のため、符号無し整数と同一になります。演算結果は、表 1 のとおりです。フラグレジスタの OF が異なるだけです。

表 1 論理加算命令と算術加算命令の比較(その 1)

	論理加算命令	算術加算命令
命令	ADDL GR0,GR1	ADDA GR0,GR1
計算結果 GR0	$(8000)_{16}$	$(8000)_{16}$
フラグレジスタ OF SF ZF	 0 1 0	 1 1 0

次に、もっと違う場合を示します。GR0=#000A、GR1=#FFFA とします。符号無整数は、

#000A $=(000A)_{16}$
$=(0000\ 0000\ 0000\ 1010)_2$
$=(10)_{10}$

#FFFA $=(FFFA)_{16}$
$=(1111\ 1111\ 1111\ 1010)_2$
$=(65530)_{10}$

です。次に、符号有り整数を考えます。#000A は第 15 ビットが 1 なので、符号無しと符号ありは数学の 10 進数にすると同じ値です(10 進数でなくともよい)。一方、#FFFA は負の数を表すため、異なります。負の数を求める方法は、①ビット反転②+1 加算でしたよね。

$(FFFA)_{16}$
1111 1111 1111 1010
0000 0000 0000 0101 ← ビット反転
0000 0000 0000 0110 ← +1 加算
4+2=6 ← 10 進数に変換
$-(6)_{10}$ ← 符号有り整数

加算は、次のように実行されます。いずれも、GR0 の値は(0004)となります。しかし、フラグレジスタの値が異なります。算術加算では 10-6=4 で何も生じませんが、論理加算で

はオーバーフローしてしまいますので、OF が 1 になります。

```
    1111 1111 1111 1010
+   0000 0000 0000 1010
  1 0000 0000 0000 0100
```

表 2 論理加算命令と算術加算命令の比較(その 2)

	論理加算命令	算術加算命令
命令	ADDL GR0,GR1	ADDA GR0,GR1
計算結果 GR0	$(0004)_{16}$	$(0004)_{16}$
フラグレジスタ OF SF ZF	 1 0 0	 0 0 0

これらの例から分かるように、算術加算と論理加算の計算結果の違いはフラグレジスタの違いになります。

4.2 論理減算(SUBL)

算術加算と同じです。計算結果のフラグレジスタの値に気をつけましょう。