機械語命令 (算術・論理演算)

山本昌志*

2006 年 1 月 20 日

1　前回の復習と本日の学習

1.1　復習

1.1.1　アセンブラ命令

アセンブラーは，アセンブラ言語を機械語に変換するプログラムである．アセンブラ命令は，アセンブラーにその変換方法を指示する．CASL II には，次の 4 つのアセンブラ命令がある．

START	プログラムの先頭に，必ず書く必要がある．プログラムの実行開始番地を指示する．
END	プログラムの最後に，必ず書く必要がある．プログラムの記述の最後を示す．プログラムの実行の終了を示すものではない．
DC	プログラムで処理すべきデータを定義する．メモリーの初期値を与えると解釈してもよい．
DS	プログラムの実行に必要なメモリーを確保する．

プログラムは命令とデータから構成されると以前に述べたが，データ部を構成するために，DC と DS の命令がある．

1.1.2　機械語命令 (データ転送)

実際の CPU の動作を示す命令が機械語命令である．要するにこれは，CPU というハードウェアーができることを示している．プログラムを構成するデータと命令のうち，後者は機械語命令から構成される．

前回の授業では，データ転送に関する 3 つの機械語命令を学習した．

LD	メインメモリーや他の汎用レジスターのデータ (内容) を汎用レジスターにコピーする．
ST	汎用レジスターのデータ (内容) をメインメモリーにコピーする．
LAD	メインメモリーの実効番地を汎用レジスターにコピーする．指標レジスターの使い方によっては，汎用レジスターの値を操作できる．

*国立秋田工業高等専門学校　電気工学科

1.2 本日の学習内容

本日は，CASL II の整数の演算である算術加算・減算と論理加算・減算，論理演算について，説明する．教科書 [1] の p.45-55 である．学習のゴールは，以下の命令の動作を理解することである．

- 1 語を符号付き整数として取り扱い，加算と減算を行う．

算術加算	ADDA	1 語のデータを符号付き整数と見なし，加算を行う．
算術減算	SUBA	1 語のデータを符号付き整数と見なし，減算を行う．

- 算術加算とは異なり，これらは 1 語を符号無し整数として取り扱い加算・減算を行う．

論理加算	ADDL	1 語のデータを符号無し整数と見なし，加算を行う．
論理減算	SUBL	1 語のデータを符号無し整数と見なし，減算を行う．

- 2 年生のとき学習したブール代数の演算である，論理積と論理和，排他的論理和の命令について説明する．これらは，1 語を各ビットごとに演算を行う．

論理積	AND	1 語のデータを各ビット毎に論理積を計算する．
論理和	OR	1 語のデータを各ビット毎に論理和を計算する．
排他的論理和	XOR	1 語のデータを各ビット毎に排他的論理和を計算する．

理解のポイントは，

- 演算するデータは 16 ビットで，それを符号付 2 進数，あるいはブール代数の真偽として取り扱う．負の数は 2 の補数という表現が使われる．
- 計算結果，フラグレジスタがどのようになるかを考える．

である．

2 演算とは

2.1 コンピューターでの演算

コンピューターの仕事は，データの加工でる．与えられたデータを，目的のデータに加工する．例えば，

- 整数の 5 と 8 のデータが与えられると，それを加工して和である 13 を表示する．
- CD-ROM のビット列のデータを加工して，音に変換する．
- 飛行機のスロットルレバーの角度とエンジンの回転数などのデータから，燃料噴射弁の角度を与えるデータを作る．

等である．どのような場合でも，図 1 のようになっている．この入出力データのことを情報と言い，それをプログラムの命令により処理を行い，出力データを作る．情報処理とはこのようなことを言う．

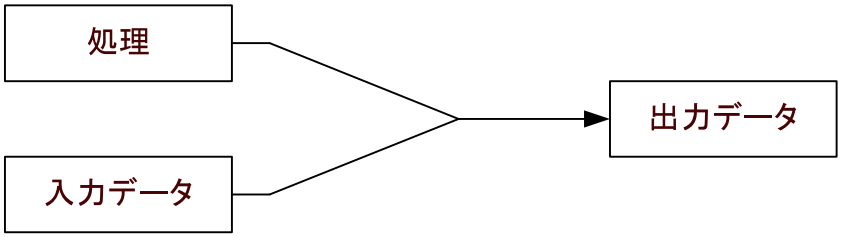

図 1: 情報処理

コンピューターの内部，とくにデータの処理を行う CPU では，入力と出力のデータはビット列である．CPU では命令に従い，ビット列の操作を行っている．このビット列を操作する処理のことを演算といい，プログラムでは演算命令がそれを担う．

数学でも演算と言う言葉を使うが，その内容は非常に似ている．例えば，数学との対比では，

- $\sin \pi/2$ を計算して，その結果として 1 を得る．この場合，入力データは $\pi/2$ で，演算は sin で，出力データは 1 である．

のように考えられる．

ポイント

コンピューターは，入力データのビット列から出力ビット列を作る処理を行っている．そのビット列の変換の処理を演算と言う．

2.2 CASL II の演算

先ほど述べたように，CASL II でも演算命令を使ってデータのビットパターンを変化させる．ただし，これらの命令で変化させることができるのは，汎用レジスターに格納されたデータだけである．汎用レジスターのデータが変化すると，それにつられて他のレジスターの値も変化するが，それは二次的なものである．あくまで，演算の結果，すなわちデータの変化は汎用レジスターに蓄えられるのである．

CASL II で用意されている演算は，

- 算術・論理演算命令
- 比較演算命令
- シフト演算命令

である[1]．たったこれだけで，かなり広範囲のデータ処理が可能である．命令の動作については，今後，学習する．諸君は，数学的な準備は出来ているので，これらがそんなに難しく思う必要が無い．例えば，＋と言う数学の記号の代わりに，`ADDA` と書くことを覚えればよい．

[1] 教科書の p.208 に全ての演算命令が書かれている

3　算術加算・減算

算術加算と減算は，データを 16 ビットの符号付整数として計算する．CASL II では加算と減算の命令が用意されているが乗除算は無い．

3.1　算術加算 (ADDA)

3.1.1　内容

命令語　ADDA:　**ADD A**rithmetic　(add:加える　arithmetic:算術)

役割　符号有り整数の加算を行う命令

書式と内容

ラベル欄	命令コード 欄	オペランド 欄	
label	ADDA	r1,r2	r1←r1+r2
label	ADDA	r,*adr*[,x]	r←r+(*adr*+[x]) の内容

FR　計算結果に応じて，変化する．

Flag	bit	計算結果	ビット パターン
OF	1	結果 ＜-32768 または 32767＜ 結果	結果が 16 ビットを超えたとき
	0	-32768≦ 結果 ≦32767	結果が 16 ビット以内
SF	1	結果が負の場合	第 15 ビットが 1
	0	結果が正の場合	第 15 ビットが 0
ZF	1	結果がゼロの場合	全てのビットが 0
	0	結果がゼロ以外の場合	いずれかのビットが 1

2 個の 16 ビットの値を加算する命令である．16 進数でも 10 進数でも正しく計算できる．ただし符号付で計算を行うので，最上位の第 15 ビットは符号ビットを表し，2 の補数で表現される．

3.1.2　例

```
ADDA  GR0,GR1       ;GR0 ← GR0+GR1
ADDA  GR0,A         ;GR0 ← GR0+(アドレス A の内容)
ADDA  GR0,A,GR1     ;GR0 ← GR0+(アドレス [A+GR1] の内容)
ADDA  GR0,=5        ;GR0 ← GR0+5
```

3.2 算術減算 (SUBA)

3.2.1 内容

命令語 SUBA: **SUB**tract **A**rithmetic (subtract:引き算 arithmetic:算術)

役割 符号有り整数の減算 (引き算) を行う命令

書式と内容

ラベル欄	命令コード 欄	オペランド 欄	
label	SUBA	r1,r2	r1←r1-r2
label	SUBA	r,*adr*[,x]	r←r-(*adr*+[x]) の内容

FR 計算結果に応じて，変化する．

Flag	bit	計算結果	ビットパターン
OF	1	結果 ＜-32768 または 32767＜ 結果	結果が 16 ビットを超えたとき
	0	-32768≦ 結果 ≦32767	結果が 16 ビット以内
SF	1	結果が負の場合	第 15 ビットが 1
	0	結果が正の場合	第 15 ビットが 0
ZF	1	結果がゼロの場合	全てのビットが 0
	0	結果がゼロ以外の場合	いづれかのビットが 1

二つのデータの引き算を行う．その他は，加算命令と同じ．

3.2.2 例

```
SUBA  GR0,GR1      ;GR0 ← GR0-GR1
SUBA  GR0,A        ;GR0 ← GR0-(アドレス A の内容)
SUBA  GR0,A,GR1    ;GR0 ← GR0-(アドレス [A+GR1] の内容)
SUBA  GR0,=5       ;GR0 ← GR0-5
```

4 論理加算・減算

論理加算と減算は，データを 16 ビットの符号なし整数として計算する．先ほど示した算術加算 (ADDA) と算術減算 (SUBA) は，符号付き整数[2] として取り扱う．これらの違いに注意が必要である．たとえば，同じ 16 ビットのデータでも，10 進数の整数として取り扱うとき，次にようになる．なぜこのようになるか，忘れた者は以前のノートを見よ．

[2]第 15 ビットを符号ビットと見なし，2 の補数で表現される．

表 1: 符号付き整数と負号無し整数

ビットパターン	符号無整数	符号付き整数
0000000001010011	$(83)_{10}$	$(83)_{10}$
1000000001010011	$(32851)_{10}$	$(-32685)_{10}$

4.1 論理加算 (ADDL)

4.1.1 内容

命令語 ADDL: **ADD L**ogical (add:加える logical:論理的な)
役割 符号無し整数の足し算を行う命令.
書式 教科書 (p.48) の通り.
機能 教科書 (p.48) の通り.
フラグレジスタ 教科書 (p.49) の通り.

符号無し整数の演算を行うということは，全て正として取り扱う．それなのに，フラグレジスタの SF が 1 になることに対して，疑問に思うだろう．それについては，教科書の例題で説明する．

4.1.2 使用例

```
ADDL  GR0,GR1       ;GR0 ← GR0+GR1
ADDL  GR0,A         ;GR0 ← GR0+(アドレス A の内容)
ADDL  GR0,A,GR1     ;GR0 ← GR0+(アドレス [A+GR1] の内容)
ADDL  GR0,=5        ;GR0 ← GR0+5
```

教科書の例題を実行したときのメモリーとレジスターの内容を表 2 示す．それらの値は，各行の実行の終了時点での値である．ただし，アスタリスク (∗) の箇所は，未定であることを示す．物理的にそれらが存在するので，値は未定ではあるが，ビットパターンはある．1, 6, 7, 8, 9 行はアセンブラ命令なので実行されない．そのため，メモリーやレジスタの値は空白としている．

この例題で重要なことは，フラグレジスタの SF の値である．ここでの計算は，

$$\begin{aligned}(7FFF)_{16}0(1)_{10} &= (0111111111111111)_2 + (0000000000000001)_2 \\ &= (1000000000000000)_2 \\ &= (8000)_{16}\end{aligned}$$

を行っている．計算結果が正であっても，第 15 ビットが 1 なので，Sign flag(SF) が 1 となる．

もし，3 行目の ADDL の代わりに，ADDA を使うと，GR1 の内容は同じ #8000 となるが，フラグレジスターは OV=1，SF=1，ZF=0 となる．オーバーフローフラグが異なる．理由は明らかであろう．

表 2: 教科書 List4-6(p.49) の実行例.

行	プログラム			GR1	OF	SF	ZF	AA	BB	CC
1	PGM	START								
2		LD	GR1,AA	#7FFF	0	0	0	#7FFF	1	*
3		ADDL	GR1,BB	#8000	0	1	0	#7FFF	1	*
4		ST	GR1,CC	#8000	0	1	0	#7FFF	1	#8000
5		RET		#8000	0	1	0	#7FFF	1	#8000
6	AA	DC	#7FFF							
7	BB	DC	1							
8	CC	DS	1							
9		END								

4.2 論理減算 (SUBL)

4.2.1 内容

命令語 SUBL: **SUBtract L**ogical (subtract:引き算 logical:論理的な)

役割 符号無し整数の引き算を行う命令.

書式 教科書 (p.50) の通り.

機能 教科書 (p.50) の通り.

フラグレジスタ 教科書 (p.49) の通り.

4.2.2 使用例

```
SUBL  GR0,GR1       ;GR0 ← GR0-GR1
SUBL  GR0,A         ;GR0 ← GR0-(アドレス A の内容)
SUBL  GR0,A,GR1     ;GR0 ← GR0-(アドレス [A+GR1] の内容)
SUBL  GR0,=5        ;GR0 ← GR0-5
```

教科書の例題を実行したときのメモリーとレジスターの内容を表 3 に示す．この例題で重要なことは，フラグレジスタの SF の値である．ここでの計算は,

$$\begin{aligned}(FFFF)_{16} - (1)_{10} &= (1111111111111111)_2 - (0000000000000001)_2 \\ &= (1111111111111110)_2 \\ &= (FFFE)_{16}\end{aligned}$$

を行っている．計算結果が正であっても，第 15 ビットが 1 なので，Sign flag(SF) が 1 となる．

表 3: 教科書 List4-7(p.50) の実行例．メモリーとレジスターの値は，各行の実行の終了時点での値である．アスタリスク (∗) の箇所は未定を表し，実行されない行の場合は空白としている．

行	プログラム			GR1	OF	SF	ZF	AA	BB	CC
1	PGM	START								
2		LD	GR1,AA	#FFFF	0	1	0	#FFFF	1	∗
3		SUBL	GR1,BB	#FFFE	0	1	0	#FFFF	1	∗
4		ST	GR1,CC	#FFFE	0	1	0	#FFFF	1	#FFFE
5		RET		#FFFE	0	1	0	#FFFF	1	#FFFE
6	AA	DC	#FFFF							
7	BB	DC	1							
8	CC	DS	1							
9		END								

5 論理演算命令

CASL II の場合，論理積と論理和，排他的論理和の論理演算命令が用意されている．COMET II は，1 語である 16 ビットの各ビット毎の論理演算を行う．論理演算については，2 年生のブール代数で学習したが，忘れた人もいると思うので，表に載せておく．

表 4: 論理積 (AND)

A	B	$A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

表 5: 論理和 (OR)

A	B	$A + B$
0	0	0
0	1	1
1	0	1
1	1	1

表 6: 排他的論理和 (XOR)

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

5.1 論理積 (AND)

5.1.1 内容

命令語 AND: **AND** (and:かつ)
役割 ビット毎の論理積を計算する．
書式 教科書 (p.51) の通り．
機能 教科書 (p.51) の通り．
フラグレジスタ 教科書 (p.51) の通り．

5.1.2 使用例

```
AND  GR0,GR1        ;GR0 ← (GR0 の各ビット).AND.(GR1 の各ビット)
```

```
AND  GR0,A          ;GR0 ← (GR0 の各ビット).AND.(アドレス A の内容の各ビット)
AND  GR0,A,GR1      ;GR0 ← GR0.AND.(アドレス [A+GR1] の内容)
AND  GR0,=5         ;GR0 ← GR0.AND.(0000000000000101)
```

AND 命令の使われ方として多いのは，マスク処理への応用である．たとえば，GR0 の最下位のビットが 0 か 1 かを調べる場合である．この場合，AA の内容を，#0001 として，

```
AND  GR0,AA     ;AA=#0001
```

を実行する．すると，もし GR1 の最下位ビット (第 0 ビット) が 0 の場合，フラグレジスタの ZF=1 になる．このような使われ方は非常に多い．この処理には，第 0 ビット以外は関係ないので，隠している．このことをマスクと言う．このようにして特定のビットを調べることができる．次に述べる OR の命令でも，これと似た処理ができそうに思えるが，大変面倒である．なぜか考えてみよ．

この応用として，特定のビットを 0 にすることができる．たとえば，A の内容を#5555 として，

```
AND  GR0,A     ;#5555
```

を実行する．すると，GR1 の奇数番目のビットが 0 に設定される．

5.2 論理和 (OR)

5.2.1 内容

命令語	OR: **OR** (or:または)
役割	ビット毎の論理和を計算する．
書式	教科書 (p.53) の通り．
機能	教科書 (p.53) の通り．
フラグレジスタ	教科書 (p.51) の通り．

5.2.2 使用例

```
OR   GR0,GR1        ;GR0 ← (GR0 の各ビット).OR.(GR1 の各ビット)
OR   GR0,A          ;GR0 ← (GR0 の各ビット).OR.(アドレス A の内容の各ビット)
OR   GR0,A,GR1      ;GR0 ← GR0.OR.(アドレス [A+GR1] の内容)
OR   GR0,=5         ;GR0 ← GR0.OR.(0000000000000101)
```

AND とは反対に，OR 命令は，特定のビットを 1 にすることができる．たとえば，AA の内容を#5555 として，

```
OR   GR0,AA    ;AA=#5555
```

を実行する．すると，GR1 の偶数番目のビットが 1 に設定される．

5.3 排他的論理和 (XOR)

5.3.1 内容

命令語	XOR: e**X**clusive **OR** (exclusive:排他的な or:または)
役割	ビット毎の排他的論理和を計算する.
書式	教科書 (p.54) の通り.
機能	教科書 (p.54) の通り.
フラグレジスタ	教科書 (p.51) の通り.

5.3.2 使用例

```
XOR  GR0,GR1      ;GR0 ← (GR0 の各ビット).XOR.(GR1 の各ビット)
XOR  GR0,A        ;GR0 ← (GR0 の各ビット).XOR.(アドレス A の内容の各ビット)
XOR  GR0,A,GR1    ;GR0 ← GR0.XOR.(アドレス [A+GR1] の内容)
XOR  GR0,=5       ;GR0 ← GR0.XOR.(0000000000000101)
```

XOR 命令は，ビットを反転することができる．すなわち，論理否定 (NOT) の動作をすることができる．これは，AA の内容を#FFFF として，

```
XOR  GR0,AA   ;AA=#FFFF
```

を実行する．すると，GR1 のすべてのビットが反転される．

特定のビットを反転することも可能である．たとえば，AA の内容を#5555 として，

```
XOR  GR0,AA   ;AA=#5555
```

を実行する．すると，GR1 の偶数番目のビットが反転される．

6 レポート

6.1 課題

[**問 1**] 次の論理加算を行うプログラムを作成し，プログラムの各実効段階のフラグレジスタの値を示せ.

(ア) **10+123** (イ) **#0FFF+#0001** (ウ) **#8001+32767**

[**問 2**] 次の論理減算を行うプログラムを作成し，プログラムの各実効段階のフラグレジスタの値を示せ.

(ア) **10-123** (イ) **#7FFF-#8FFE** (ウ) **#8001-32767**

[**問 3**] 次の論理加算のプログラムの各実効段階でのメモリーとフラグレジスタの値を示せ.

行	プログラム			GR1	OF	SF	ZF	AA	BB	CC
1	PGM	START								
2		LD	GR1,AA							
3		ADDL	GR1,BB							
4		ST	GR1,CC							
5		RET								
6	AA	DC	#FFF1							
7	BB	DC	#000F							
8	CC	DS	1							
9		END								

[**問 4**] 次の論理減算のプログラムの各実効段階でのメモリーとフラグレジスタの値を示せ.

行	プログラム			GR1	OF	SF	ZF	AA	BB	CC
1	PGM	START								
2		LD	GR1,AA							
3		SUBL	GR1,BB							
4		ST	GR1,CC							
5		RET								
6	AA	DC	#8000							
7	BB	DC	#8001							
8	CC	DS	1							
9		END								

[**問 5**] 次のプログラムの動作をするプログラムを作成せよ.

- #ABCD の奇数番目のビットを 0 に設定する.
- 結果を，メモリーに格納する.

[**問 6**] 次のプログラムの動作をするプログラムを作成せよ.

- #ABCD の第 0,1,2,3 と第 8,9,10,11 番目のビットを 1 に設定する.
- 結果を，メモリーに格納する.

6.2 レポート提出要領

提出方法は，次の通りとする.

期限　　1 月 27 日 (金) PM 1:00
用紙　　A4
提出場所　山本研究室の入口のポスト
表紙　　表紙を 1 枚つけて，以下の項目を分かりやすく記述すること．
　　授業科目名「電子計算機」
　　課題名「課題　算術・論理演算」
　　3E　学籍番号　氏名
　　提出日
内容　　2 ページ以降に問いに対する答えを分かりやすく記述すること．

7　付録

7.1　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ある．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る．しかし，単純な COMET II には，そのハードウェアはなく，当然機械語命令も無い．そのため，ソフトウェアーでその仕組みを実現させなくてはならない．詳細については，後で学習するが，ちょっとだけ方法を示す．興味深い方法である．

7.1.1　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う．このように算術加算を用いて乗算を行うことができる．ただし，この方法はもっとも原始的で効率の悪い方法である．人間がこの計算を行うのは非現実的であるが，単純作業を非常に高速で行うコンピューター向きの方法である．

この例でもわかるように加算ができれば，乗算はできるのである．

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ある．これは，小学生のときに学習をした筆算を用いた掛け算と同じである．たとえば，34×24 を計算する場合，筆算は $34 \times (2 \times 10^1 + 4 \times 10^0)$ と分解したはずである．そうして，次の手順でこの除算を行ったはずである．

1. 34×2 を計算し，1 桁ずらす (10 倍する).
2. 34×4 を計算する．
3. 先の計算結果を合計する．この合計 816 が 34×24 の計算結果である．

同じことを 2 進数で行う．これがコンピューターによる乗算である．先ほどと同じ計算 (32×24) を行う．これを 2 進数で表現すると，

$$(100010)_2 \times (11000)_2 = (100010)_2 \times (1 \times 2^4 + 1 \times 2^3)$$

となる．これを先ほど同様の手順で計算する．

1. 掛け算は 1 倍なので計算する必要が無く，最初に $(100010)_2$ を 4 桁左にずらす (ビットシフト)．すると，$(1000100000)_2$ となる．
2. 次に $(100010)_2$ を 3 桁左にずらす．すると，$(100010000)_2$ となる．
3. 先の計算結果を合計すると，$(1100110000)_2$ となる．これは，10 進数の 816 である．

ここでは，ビットシフトと加算命令を使った．これらの命令が用意されていれば乗算ができることがわかるであろう．実際，CASL II にはビットシフトの命令は用意されている．

7.1.2 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．この例でもわかるように減算ができれば，除算はできるのである．この方法は効率が悪いので，もう少しましな方法を考える．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る．計算は次のようにする．計算の準備として，10 と 3 を 2 進数で表す．

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

次に示すように計算すれば，計算効率は上がるであろう．計算順序は，筆算での割り算と同じである．

1. $(1)_2$ から $(11)_2$ を減算したいが，負になるのでそれは不可とする．
2. $(10)_2$ から $(11)_2$ を減算したいが，これも負になるので不可とする．
3. $(101)_2$ から $(11)_2$ を減算する．それは可能で，結果は $(10)_2$ で，その桁に $(1)_2$ が立つ．
4. 先ほどの残りと次の桁を合わせた $(100)_2$ は減算可能である．減算の結果は $(1)_2$ で，その桁に $(1)_2$ が立つ．
5. これ以上桁がないので，計算は完了である．商は $(11)_2 = (3)_{10}$ 余りは $(1)_2 = (1)_{10}$ となる．

ビットシフトを使う方法 直ちに，ビットシフトが適用できるのは，割る数が 2^n になっている場合である．ただ，先ほどの筆算のアルゴリズムでもビットシフトは使ってはいる．

7.1.3 まとめ (乗除算)

ということで，加算と減算ができれば乗除算は可能である．さらに賢明な諸君は，次の疑問が湧くはずである．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) を使うと四則演算に分解できることを学習したはずである．したがって，四則演算ができれば，それらの関数は計算可能である．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するようにしている．

参考文献

[1] 東田幸樹, 山本芳人, 広瀬啓雄. アセンブラ言語　CASL II. 工学図書株式会社, 2002 年.