論理加算・減算と論理演算

山本昌志*

2004 年 9 月 6 日

1　前回の復習と本日の学習

1.1　復習

先週は、算術加算・減算命令を学習した。

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。

併せて、加算と減算命令を使えば、乗除算の演算ができることを示した。

1.2　本日の学習内容

本日の学習内容は、

- 整数の演算である論理加算・減算命令を説明する。ただし、算術加算とは異なり、これらは、1 語を符号なし整数として取り扱う。

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。

2　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。前回学習した算術加算 (ADDA) と算術減算 (SUBA) は、符号付き整数[1] 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

*国立秋田工業高等専門学校　電気工学科

[1]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。

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。

表 1: 符号有り整数と負号無し整数

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

2.1 論理加算 (ADDL)

2.1.1 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。

2.1.2 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								

2.2 論理減算 (SUBL)

2.2.1 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。

2.2.2 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								

3 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

3.1 論理積 (AND)

3.1.1 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。

3.1.2 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、AA 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。

3.2 論理和 (OR)

3.2.1 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。

3.2.2 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。

3.3 排他的論理和 (XOR)

3.3.1 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。

3.3.2 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。

4 練習問題

来週から試験なので、ここの練習問題は課題としませんが、これくらいはできるようになってください。

4.1 算術加算

次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								

4.2 算術減算

次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								

4.3 論理積

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。

4.4 論理和

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。

4.5 排他的論理和

次のプログラムの動作をするプログラムを作成せよ。

- 3322 を-1 倍する。
- 結果を、メモリーに格納する。

  [ヒント]　-1 倍するためには、ビット反転と+1 加算すればよい。