

3 演算命令

- 3-1 足し算・引き算の基本
- 3-2 符号つき2進数の足し算・引き算
- 3-3 論理演算
- 3-4 XOR命令は役に立つ
- 3-5 論理シフト命令
- 3-6 算術シフト命令
- 3-7 簡単な掛け算と割り算

82

3-1 足し算・引き算の基本(1)

```

1:PRG0001 START
4: LD GR1,DATA ;GR1にDATA番地の値100を搬入
5: LAD GR2,50 ;GR2に50を搬入
6: ADDL GR1,GR2 ;100 + 50
8: LD GR3,=100 ;GR3にリテラルを使って100を搬入
9: ADDL GR3,=50 ;100 + 50(リテラルを使っている)
11: LAD GR4,DATA2 ;GR4にDATA2のアドレスを搬入
12: LD GR5,0,GR4 ;GR5にDATA2+0番地の値10000を読み込む
13: ADDL GR5,1,GR4 ;GR5にDATA2+1番地の値20000を加算
14: ADDL GR5,2,GR4 ;GR5にDATA2+2番地の値30000を加算
16: LD GR6,DATA ;GR6に100を搬入
17: SUBL GR5,GR6 ;60000 - 100
18: SUBL GR5,=200 ;59900 - 200(リテラルを使っている)
19: SUBL GR5,2,GR4 ;実アドレスDATA2+2の値を減算
22: RET
23:DATA DC 100
24:DATA2 DC 10000,20000,30000
25: END
    
```

83

3-1 足し算・引き算の基本(2)

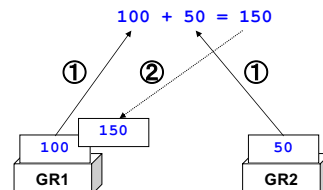
■レジスタ間の加算

【6行目】 ADDL GR1,GR2

- ◆ ADDL命令は足し算を行う**論理加算命令**であり、この命令によって、GR1の値にGR2の値が加算される。
- ◆ ここでは $100 + 50 = 150$ という計算を行っていることになる。

①GR1の値とGR2の値の加算が行われる。(実線の矢印)

②その結果がGR1に上書きされる。(点線の矢印)



84

3-1 足し算・引き算の基本(3)

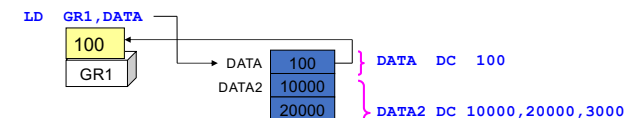
■リテラル【literal】

- ・ プログラムのソースコード中に使用される定数のこと。「255」「casl」など、いわゆる単なる数字や文字列がリテラルの代表例である(IT用語辞典)。

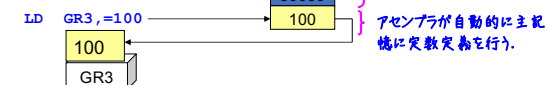
【8行目】 LD GR3,=100

- ◆ オペランド第2項の「=100」の部分は、命令のなかで定数を直接記述するためのもので、**リテラル**と呼ばれる。
- ◆ この命令は、GR3に10進定数値100が設定される[実行のよう(c)]

(a) ラベル



(b) リテラル



85

3-1 足し算・引き算の基本(4)

■リテラルの使う利点

- 定数を読み込むためにDC命令を書いたり、DATAなどのラベルをわざわざ定義したりしなくてもよい。
- DC命令で定義した定数は書き換えることができたが、リテラルで使った定数は書き換えられてしまうことがない。

■リテラルを使った加算命令の例

【9行目】 ADDL GR3,=50

- ➡ オペランド第2項が「=50」となっていて、リテラルが使われている。
- ➡ 9行目は「GR3の値に値50を足してGR3に設定する」という意味となる。
- ➡ この結果、GR3の値は100 + 50 = 150になる。

86

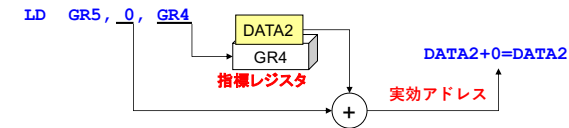
86

3-1 足し算・引き算の基本(5)

■インデックス修飾を用いた加算(1)

【12行目】 LD GR5,0,GR4

- ➡ 12行目はインデックス修飾を用いたLD命令であり、実効アドレスを求める。
- ➡ オペランドの第2項と第3項から、実効アドレスDATA2が得られる。
- ➡ LD命令なので、DATA2番地に格納された値10000(2710₁₆)を読み出して、オペランド第1項で指定されているGR5に設定する。



87

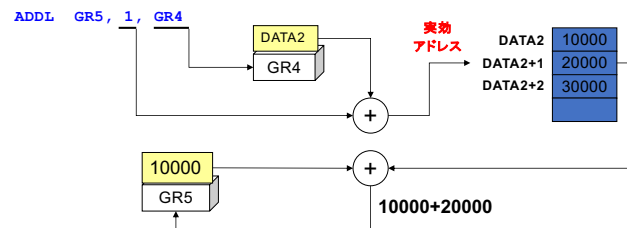
87

3-1 足し算・引き算の基本(6)

■インデックス修飾を用いた加算(2)

【13行目】 ADDL GR5,1,GR4

- ➡ オペランド第3項のGR4の値DATA2に、第2項の値1を加えたDATA2 + 1が実効アドレスである。
- ➡ オペランド第1項で指定されたGR5の値(10000)に実効アドレスで指定されたDATA2 + 1番地の値20000を加えて、結果30000(7530₁₆)をGR5に設定する。



88

88

3-1 足し算・引き算の基本(7)

■3種類の加算の仕方

【1】2つのレジスタの加算

➡ ADDL GR1,GR1

【2】リテラルを用いた加算

➡ ADDL GR3,=50

【3】インデックス修飾を用いた加算

➡ ADDL GR5,2,GR4

命令	書き方		命令の説明
	命令コード	オペランド	
論理加算 ADD Logical	ADDL	r1,r2	r1 ← (r1)+(r2)
		r,adr[,x]	r1 ← (r1)+H(実効アドレス)

89

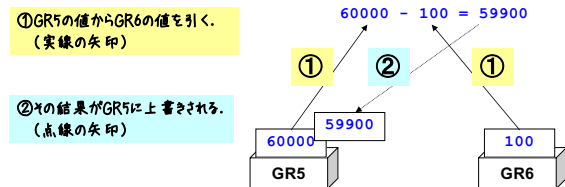
89

3-1 足し算・引き算の基本(8)

■ SUBL 命令(1)

【16,17行目】 SUBA GR5,GR6

- レジスタ間の減算であり、SUBL 命令が減算命令(正確には論理減算命令)である。
- GR5の値60000から、GR6の値100が引かれ、その結果59900(59FC₁₆)がGR5に設定される。
- 注意する点は、オペランド第1項(GR5)が引かれる数であることである。



90

90

3-1 足し算・引き算の基本(9)

■ SUBL 命令(2)

【18行目】 SUBA GR5,#200

- 減算命令でリテラルを使用する場合には、リテラルで指定できる値は、引く数(a-b)となる。

【19行目】 SUBA GR5,2,GR4

- オペランドに項目が3つあるのがインデックス修飾を用いた減算であり、実効アドレスを求めると、GR4にはDATA2のアドレスが設定されているので、(GR4)+2=DATA2+2 となる。

命令	書き方		命令の説明
	命令コード	オペランド	
論理減算 SUBtract Logical	SUBL	r1,r2	r1 ← (r1)-(r2)
		r,adr[,x]	r1 ← (r1)-(実効アドレス)

91

91

問題[3-1]

GR1に30d,GR2に10d,GR3に15d,GR4に5dを設定し、次の加減算を行うプログラムを作成せよ。ただし、命令は、リテラル、ADDL、SUBLを使い、計算して下さい。

$$GR1 + GR2 - GR3 + GR4 + 100$$

92

92

3-2 符号つき2進数の足し算・引き算 (1)

1:PRG0302 START

4: LAD GR1,10

5: LAD GR2,10

6: LAD GR3,-10

7: LAD GR4,65526

8: ADDA GR1,GR3 ; 算術加算 10+(-10) = 0

9: ADDL GR2,GR4 ; 論理加算 10+65526 = 65536(オーバーフロー)

11: LAD GR1,1

12: LAD GR2,1

13: ADDA GR1,#32767 ; 算術加算 1+32767 = 32768(オーバーフロー)

14: ADDL GR2,#32767 ; 論理加算 1+32767 = 32768

16: LD GR1,#100

17: LD GR2,#100

18: SUBA GR1,#103 ; 算術減算 100-103 = -3

19: SUBL GR2,#103 ; 論理減算 100-103 = 65533(オーバーフロー)

22: RET

23: END

93

93

3-2 符号つき2進数の足し算・引き算 (2)

■負の値の扱い方

【6,7行目】 LAD GR3,-10

→ CASL II において、16進数の数値は2通りの解釈があることによるものである。

2進数	16進数	符号なし数値	符号つき数値
1111 1111 1111 1111	FFFF	65535	-1
1111 1111 1111 1110	FFFE	65534	-2
1111 1111 1111 0101	FFF6	65526	-10
1000 0000 0000 0001	8001	32769	-32767
1000 0000 0000 0000	8000	32768	-32768
0111 1111 1111 1111	7FFF	32767	32767
0000 0000 0000 0010	0002	2	2
0000 0000 0000 0001	0001	1	1
0000 0000 0000 0000	0000	0	0

94

94

3-2 符号つき2進数の足し算・引き算 (3)

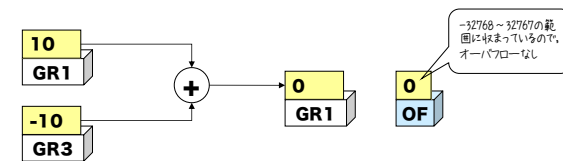
■算術加算と論理加算(1)

【8行目】 ADDA GR1,GR3

→ ADDA命令は、加算する値も符号付きの数値と考慮して、加算を行う算術加算命令である。

→ GR1の値000A16はGR3の値FFFF16を符号付きの数値と考慮して加算を行う。

命令	書き方		命令の説明
	命令コード	オペランド	
算術加算 ADD Arithmetic	ADDA	r1,r2 r,adr[,x]	r1 ← (r1)+(r2) r1 ← (r1)+(実効アドレス)



95

95

3-2 符号つき2進数の足し算・引き算 (4)

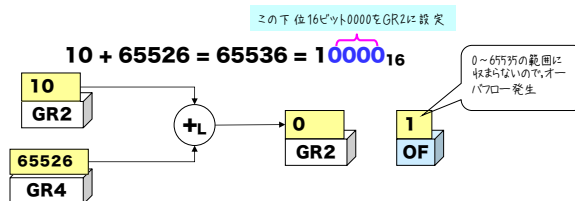
■算術加算と論理加算(2)

【9行目】 ADDL GR2,GR4

→ ADDL命令でGR2の値10とGR4の値65526を加算する。

→ この結果は、65536であり、2進数で表現すると1000000000000000(10000₁₆)である。

→ しかし汎用レジスタは16ビットの大きさしか持たないので、17ビット目は入りきらない。



96

96

3-2 符号つき2進数の足し算・引き算 (5)

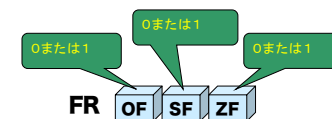
■フラグレジスタとオーバーフロー(1)

→ 演算結果が範囲を超えてしまうことをオーバーフローという。

→ オーバーフローの発生の有無を報知するためにはオーバーフローフラグ(OF)というレジスタが用意されている。

・ COMET II システムには、汎用レジスタの他にフラグレジスタ(FR)と呼ばれるレジスタがある。

・ FRは、演算結果のいろいろな状態を示し、演算命令などの命令が実行されるたびに、結果に応じた値が自動的に設定される。



OF:「演算結果がオーバーフロー」したら1, しなければ0
SF:「演算結果の符号ビットが1」なら1, でなければ0
ZF:「演算結果が0」なら1, でなければ0

97

97

3-2 符号つき2進数の足し算・引き算 (6)

■ フラレジスタとオーバフロー(2)

【13行目】 ADDA GR1,=32767

- ➡ 加算結果 $1 + 32767 = 32768$ は符号つき数値の範囲 $-32768 \sim 32767$ に収まらないので、オーバフローが起き、OFは1に設定される。GR1には32768を16進数に変換した結果である 8000_{16} が設定される。

【14行目】 ADDL GR2,=32767

- ➡ 加算結果 $1 + 32767 = 32768$ は符号なし数値の範囲 $0 \sim 65535$ に収まっているので、オーバフローは起きず、OFは0に設定される。

98

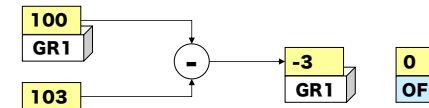
98

3-2 符号つき2進数の足し算・引き算 (7)

■ 算術減算と論理減算(1)

【18行目】 SUBA GR1,=103

- ➡ 値を符号なし数値とみなして、GR1の値からリテラルで指定している103を減算し、結果 $100 - 103 = -3 (= FFFD_{16})$ をGR1に設定する。
- ➡ 演算結果の-3は符号つき数値の範囲内 $(-32768 \sim 32767)$ なので、オーバフローは起らない。



命令	書き方		命令の説明
	命令コード	オペランド	
算術減算 SUBtract Arithmetic	SUBA	$r1, r2$ $r, \text{adr}[x]$	$r1 \leftarrow (r1) - (r2)$ $r \leftarrow (r) - (\text{実効アドレス})$

99

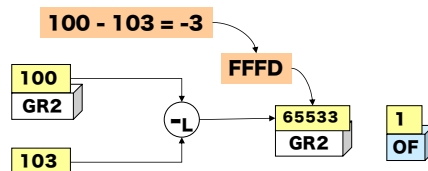
99

3-2 符号つき2進数の足し算・引き算 (8)

■ 算術減算と論理減算(2)

【19行目】 SUBL GR2,=103

- ➡ SUBL 命令によって、 $100 - 103 = -3$ は符号なし数値が表現できる範囲 $(0 \sim 65535)$ を超えているので、オーバフローが発生し、**オーバーフローフラグOF**に1が設定される。
- ➡ GR2には、-3の16進数表現である $FFFD_{16}$ が設定されるが、これは符号なし数値となると 65533 である。



100

100

問題[3-2]

- あらかじめDC命令を使い、ラベルAとBには、各々、16進数の A_{16} と F_{16} を格納する。次の論理加減算を行うプログラムを作成せよ。また、実行結果をプログラムの後に示してください。ただし、ラベル名をANS1とANS2とし、各々、領域を確保し、演算結果を保存する。

① $A + B$ (結果: $ANS1 = \text{??}_{16} = \text{??}_2$)

② $A - B$ (結果: $ANS2 = \text{??}_{16} = \text{??}_2$)

解答例

Program

実行結果

①

②

101

101

3-3 論理演算 (0)

✓ 論理積の命令AND,論理和の命令OR,排他的論理和の命令XORを学ぶ

```
1:PRG0303 START
2::      RPUH
4:      LAD GR1,DATA
5:      LD GR2,0,GR1
6:      AND GR2,#00F0
7:      LD GR3,1,GR1
8:      AND GR3,#000F
9:      OR GR2,GR3
10:     XOR GR2,#0020
12::    RPOP
13:     RET
14:DATA  DC 'a7'
15:     END
```

102

102

文字の符号表

1.3 文字の符号表

- (1) JIS X 0201 ラテン文字・片仮名用 8 ビット符号で規定する文字の符号表を使用する。
- (2) 右に符号表の一部を示す。1 文字は 8 ビットからなり、上位 4 ビットを列で、下位 4 ビットを行で示す。例えば、間隔, 4, H, ¥ のビット構成は, 16 進表示で, それぞれ 20, 34, 48, 5 C である。16 進表示で, ビット構成が 21 ~ 7E (及び表では省略している A1 ~ DF) に対応する文字を図形文字という。図形文字は, 表示 (印刷) 装置で, 文字として表示 (印字) できる。
- (3) この表にない文字とそのビット構成が必要な場合は, 問題中で与える。

a → 0061
7 → 0037

		下位4ビット				上位4ビット			
列	行	02	03	04	05	06	07		
0	間隔	0	@	P			p		
1	1	1	A	Q		a	q		
2	2	2	B	R		b	r		
3	3	3	C	S		c	s		
4	4	4	D	T		d	t		
5	5	5	E	U		e	u		
6	6	6	F	V		f	v		
7	7	7	G	W		g	w		
8	8	8	H	X		h	x		
9	9	9	I	Y		i	y		
10	10	10	J	Z		j	z		
11	11	11	K	[k	{		
12	12	12	L	¥		l	}		
13	13	13	M]		m	~		
14	14	14	N	^		n			
15	15	15	O	_		o			

103

103

3-3 論理演算 (1)

■ 論理演算

- 0と1で表される値である論理値同士の演算を論理演算という。
- 論理演算は, 計算機による処理と相性がよく, 論理演算がマスターできればさまざまな処理を効率よく実行できる。
- 代表的な論理演算に論理積AND, 論理和OR, 排他的論理和XORがある。
- CASL II では, AND命令 (論理積命令), OR命令 (論理和命令), XOR命令 (排他的論理和命令) が用意されている。

A	B	A AND B	A OR B	A XOR B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

104

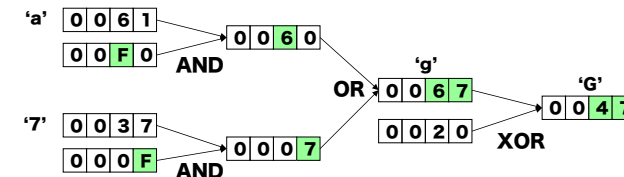
104

3-3 論理演算 (2)

例題プログラムは, 2文字を1文字に変換するプログラムである。

【手順】

- (1) 2つの文字の文字コードを求める。
'a' と '7' の文字コードは, 各々, 0061₁₆ と 0037₁₆ であるとする。
- (2) この2つの文字から 0067₁₆ を求める。
- (3) 0067₁₆ が大文字の場合は小文字に, 小文字の場合は大文字に変換する。
この結果が, 'g' は小文字なので大文字 'G' に変換する。



105

105

3-3 論理演算 (3)

■ AND命令(1)

【6行目】 AND GR2,=#00F0

- AND命令は、論理積演算を行う命令で、GR2の値とリテラルで指定した値00F0₁₆の論理積の演算を行ってGR2に設定する。
- 演算を行う値はどちらも16ビットのビット幅を持っているので、論理積演算は同じビット番号ごとの値で演算が行なわれる。
- 結果として、'a'の符号0061₁₆の「6」の部分(ビット番号7~4の4ビット)を取り出したことになる。



106

106

3-3 論理演算 (4)

■ AND命令(2)

【8行目】 AND GR3,=#000F

- GR3の値とリテラル000F₁₆の論理積の演算を行ってGR3に格納する。
- 結果として、'7'という文字を表す0037₁₆の「7」の部分(ビット番号3~0の4ビット)を取り出したことになる。

命令	書き方		命令の説明
	命令コード	オペランド	
論理積 AND	AND	r1,r2 r,adr[x]	r1 ← (r1) AND (r2) r ← (r) AND (実効アドレス)

107

107

3-3 論理演算 (5)

■ OR命令(1)

【9行目】 OR GR2,GR3

- オペランドは「GR2,GR3」なので、GR2の値とGR3の値の論理和演算を行ってGR2に設定する。
- 結果として、'a'という文字を表す0061₁₆の「6」の部分と'7'という文字を表す0037₁₆の「7」の部分をつなげて、0067₁₆という値を作ったことになる。



108

108

3-3 論理演算 (6)

■ OR命令(2)

- このように、論理積演算と論理和演算をうまく組み合わせることによって、ビット列の一部分どうしをつなげることができる。
- 例では、'a'という文字を表す0061₁₆の「6」の部分と'7'という文字を表す0037₁₆の「7」の部分をつなげて、0067₁₆という値を作ったことになる。
- 結果として、0067₁₆は文字では'g'である。

命令	書き方		命令の説明
	命令コード	オペランド	
論理和 OR	OR	r1,r2 r,adr[x]	r1 ← (r1) OR (r2) r ← (r) OR (実効アドレス)

109

109

3-3 論理演算 (7)

■ XOR命令(1)

【10行目】 XOR GR2,=#0020

- ➡ GR2の値と0020₁₆との排他的論理和をとってGR2に設定している。
- ➡ 結果として、GR2は0067₁₆から0047₁₆となり、これを文字にすると'G'である。つまり、'g'から'G'に変換したことになる。

ビット番号	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
GR2	0	0	0	0	0	0	0	0	1	1	0	0	1	1	1	1	0067 ₁₆
XOR	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0020 ₁₆
GR2	0	0	0	0	0	0	0	0	1	0	0	0	1	1	1	1	0047 ₁₆

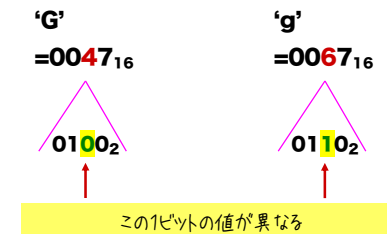
命令	書き方		命令の説明
	命令コード	オペランド	
排他的論理和 XOR		r1, r2 r, adr[x]	r1 ← (r1) XOR (r2) r ← (r) XOR (実効アドレス)

110

3-3 論理演算 (8)

■ アルファベットの英文字と小文字の変換

- ➡ 0020₁₆とのXOR命令によって、どのアルファベットに対しても大文字は小文字に、小文字は大文字に変換できる。(符号表にて確認)



111

問題[3-3]

GR1に1234₁₆, GR2にABCD₁₆を設定し、次の論理演算を行うプログラムを作成せよ。また、実行結果を報告せよ。

- ① GR1とGR2の論理積
- ② GR1とGR2の論理和
- ③ GR1とGR2の排他的論理和
- ④ GR1とFF00₁₆の論理積(上位8ビットだけを取り出している)

■ 注意

- 問題「問3-2」と同様な方法で、プログラムと実行結果をWebclassにて解答せよ。

112

3-4 XOR命令は役に立つ(1)

- ✓ DATA1領域の3語のデータに対してチェックサムを付加し、DATA2領域の3語のデータに対してパリティを付加するプログラムである。

```

1:PRG0004 START
4: LAD GR3,DATA1 ;チェックサムを求める
5: SUBL GR1,GR1 ;ゼロクリア
6: ADDL GR1,0,GR3
7: ADDL GR1,1,GR3
8: ADDL GR1,2,GR3 ;データの和を求める
9: XOR GR1,=#FFFF
10: ADDL GR1,-1 ;データの和の補数を求める
11: ST GR1,3,GR3 ;チェックサムを格納
13: LAD GR4,DATA2 ;パリティを求める
14: XOR GR2,GR2
15: XOR GR2,0,GR4
16: XOR GR2,1,GR4
17: XOR GR2,2,GR4
18: ST GR2,3,GR4 ;各ビットで1が偶数個となるような値を求めて格納
21: RET
22:DATA1 DC 1234,2468,10000,0
23:DATA2 DC 1234,1997,10000,0
24: END
  
```

114

3-4 XOR命令は役に立つ(2)

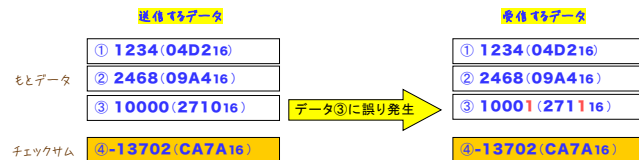
■ チェックサムの仕組み

- ➡ チェックサムとは、送信するデータの合計値が、ゼロなどの決まった値になるように、付加して送信するデータのことである。

【例】データ1234, 2468, 10000を送信するとき、チェックサムとして-13702を付加して送信する。このとき、送信する4語のデータの和は

$$1234 + 2468 + 10000 + (-13702) = 0$$

と0になっている。



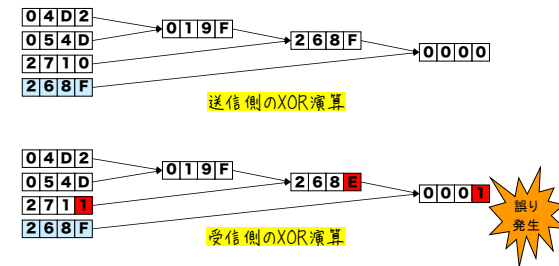
115

115

3-4 XOR命令は役に立つ(3)

■ パリティの仕組み(1)

- データ04D2₁₆, 054D₁₆, 2710₁₆を送信するとき、パリティとして268F₁₆を付加して送信する。全データに対し**TXOR演算**を行うと、すべてのビットが**0**である。
- データを受信した側では、受信した全データに対しTXOR演算を行なうと、すべてのビットが**0ではない**ので、データに誤りが発生したことがわかる。



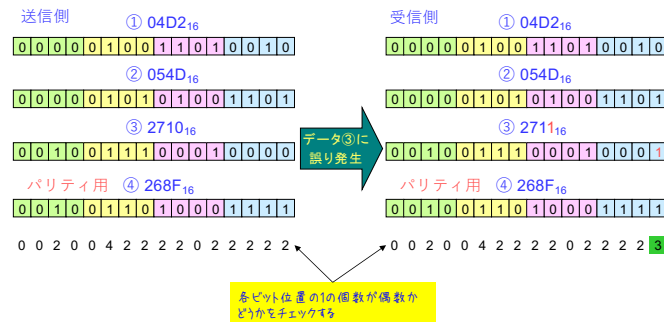
116

116

3-4 XOR命令は役に立つ(4)

■ パリティの仕組み(2)

- ⇒ ビットごとにデータに対して垂直に検査する **垂直パリティ**と、データに対して水平に検査する **水平パリティ**を行う。



117

117

3-4 XOR命令は役に立つ(5)

■ ゼロクリア

【5行目】 SUBL GR1,GR1

- ➡ GR1の値からそれ自身(GR1の値)を減算し、GR1にどのような値が入っていたとしても0になる(ゼロクリア).

- ➡ 他のクリアする方法

LD GR1,=0

LAD GR1.0

AND GR1,=

```
XOR GR1,GR1
```

← 14行目で使用

118

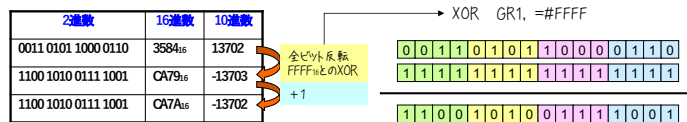
118

3-4 XOR命令は役に立つ(6)

■チェックサムの計算(2の補数の求め方)

【9,10行目】

- ➡ GR1の値13702の正負を反転させた値-13702を求める。正負を反転させた値を求めたのは、2の補数を求めるためである。
- ➡ ある値の2の補数を求めるには、その値の全ビットを反転(0を1に、1を0にする)し、1を足すことによって機械的に求めることができる。



119

119

3-4 XOR命令は役に立つ(7)

■パリティを求める

【15-17行目】 XOR GR2,0,GR4

XOR GR2,1,GR4

XOR GR2,2,GR4

- ➡ GR2を使い、DATA2領域にある連続した3語(順に1234,1357,10000)に対して、次々と排他的論理和演算を行っている。
- ➡ これによって、各ビット位置の、1の個数が偶数個あるか奇数個あるかが簡単に求められる。

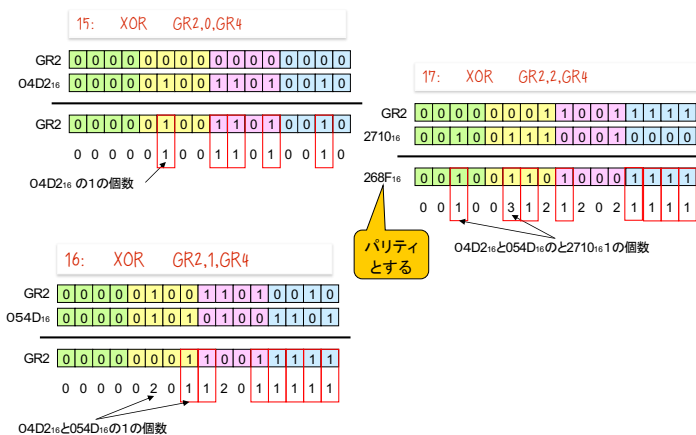
【パリティとは?】

- ➡ 送信するデータのなかの1の数が、偶数または奇数になるように付加して送信するデータのことである。
- ➡ チェックサムが誤りの有無のみを検出出来たのに対して、パリティはどのビットに誤りが発生したかまで知ることが出来る。

120

120

3-4 XOR命令は役に立つ(8)



121

121

補足

■数値から文字へ変換

```
EXMP01 START
LD GR1,NUM
ADDA GR1,='0' ;文字利用
: ADDA GR1,=#0030 ;文字コード+算術演算
: OR GR1,=#0030 ;文字コード+ビット演算
ST GR1,MOJI
OUT MOJI,LEN
RET
NUM DC 7
MOJI DS 1
LEN DC 1
END
```

122

122

問題[3-4]

以下の値に対し、パリティを求めて、その結果を報告せよ。

```

RET
DATA DC 1111.2222.3333
END

```

123

123

3-5 論理シフト命令 (1)

```

1:PRG0305 START
4: LD GR1,DATA
5: SLL GR1,2 :2ビット左シフト
6: SRL GR1,2 :2ビット右シフト
8: LD GR2,DATA
9: LD GR3,=0
10: SRL GR2,0,GR3 :0ビットシフト
12: LD GR2,DATA
13: ADDL GR3,=4
14: SRL GR2,0,GR3 :4ビットシフト
17: RET
18:DATA DC #CDEF
19: END

```

125

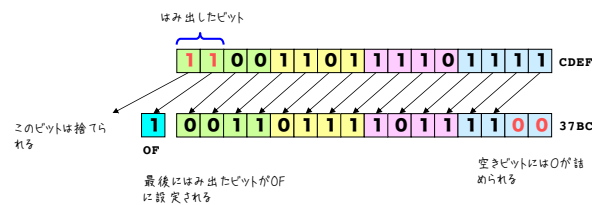
125

3-5 論理シフト命令 (2)

■ 論理左シフト

【5行目】 SLL GR1,2

- ➡ 論理左シフト命令で、オペランド第2項で指定されている2がシフトするビット数である。(もとのビット列CDEF₁₆を2ビット分左にずらす)
- ➡ もとのビット列の上2ビットは左端からはみ出してしまいが、この2ビット分は捨てられる。
- ➡ 最後に捨てられたビットの値(ビット番号14の値1)はオーバフローフラグ(OF)に設定される。
- ➡ もとのビットが左にずれた分だけ、右端に2ビット分の空きができる。ここには値0が設定される。従って、GR1の値は37BC₁₆となる。



126

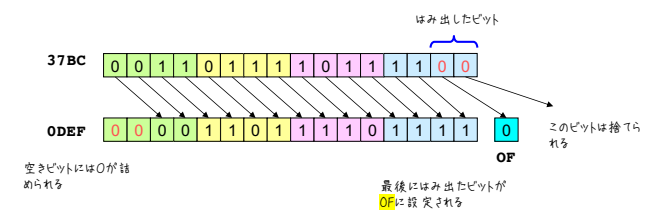
126

3-5 論理シフト命令 (3)

■ 論理右シフト

【6行目】 SRL GR1,2

- ➡ 論理右シフト命令で、オペランド第2項で指定されている2がシフトするビット数である。
- ➡ 右シフトは、左シフトとは逆に、もとのビット列37BC₁₆を2ビット分だけ右にずらして設定する。
- ➡ 右端からはみ出す下2位ビットは捨てられる。ただし、最後に捨てられたビットの値(ビット番号1の値0)はオーバフローフラグ(OF)に設定される。
- ➡ 左端にできた2ビット分の空きには値0が設定される。結果として、GR1の値は0DEF₁₆となる。



127

127

3-5 論理シフト命令 (4)

■ シフトにおける注意点

【1】シフト量は実効アドレスで指定する必要がある。

➡ 2ビットシフトしたい場合

【○】SRL GR1,2

【×】SLL GR1,=2

➡ リテラル2を格納したアドレスがシフト量になってしまうという書き方では正しい結果は得られない。

【2】ゼロクリアとして利用可能である。(5,6行目)

➡ 下位4ビットをゼロクリアしたい場合

➡ SRL GR1,4

➡ SLL GR1,4

命令	書き方		命令の説明
	命令コード	オペランド	
論理左シフト Shift Left Logical	SLL	r,adr[,x]	符号を含み(+)を実効アドレスで指定したビット数だけ左または右にシフトする。シフトの結果、空いたビット位置には0が入る。
論理右シフト Shift Right Logical	SRL	r,adr[,x]	

128

128

3-5 論理シフト命令 (5)

■ インデックス修飾を用いてビット数を指定する

【8-10行目】 SRL GR2,0,GR3

➡ DATAで定義された数値CDEF₁₆をGR2に設定する。9行目ではGR3に0を設定している。

➡ 10行目で、インデックス修飾を用いたシフトを行う。ここでは、「GR3の値(0)にオペランド第2項(0)を足したもの」、つまり0が、シフトすべきビット数として指定されていることになる。

➡ 11行目では、右シフトを実行しているが、シフトするビット数が0なので、GR2の値は結局変化せず、CDEF₁₆のままとなる。

➡ このように、インデックス修飾を用いた場合にも実効アドレスがシフト命令のシフト量となることに注意しよう。

129

129

3-6 算術シフト命令 (1)

```

1: PRG0006 START
4:     LAD GR0, #CDEF
5:     LAD GR1, #OF0F
6:     LAD GR2, #CDEF
7:     LAD GR3, #OF0F
8:     LAD GR4, #CDEF
9:     LAD GR5, #OF0F
10:    LAD GR6, #CDEF
11:    LAD GR7, #OF0F
13:    SLL GR0, 2      ;論理左シフト(符号ビットが0の値)
14:    SLL GR1, 2      ;論理左シフト(符号ビットが0の値)
15:    SRL GR2, 2      ;論理右シフト(符号ビットが0の値)
16:    SRL GR3, 2      ;論理右シフト(符号ビットが0の値)
18:    SLA GR4, 2      ;算術左シフト(符号ビットが0の値)
19:    SLA GR5, 2      ;算術左シフト(符号ビットが0の値)
20:    SRA GR6, 2      ;算術右シフト(符号ビットが0の値)
21:    SRA GR7, 2      ;算術右シフト(符号ビットが0の値)
24:    RET
25:    END

```

130

130

3-6 算術シフト命令(2)

■ 算術シフト

- ➡ 符号ビットが動かず保存される(論理シフトとは異なる)
- ➡ 符号つき数値用である(論理シフトは符号なし数値用)

■ 算術左シフト(1)

【18行目】 SLA GR4,2

➡ GR4の値CDEF₁₆を2ビット分だけ算術左シフトし、結果はB7BC₁₆になっている。

➡ ビット番号15(符号ビット)の値1はそのまま動かず、ビット番号14～0の15ビット分がシフトする。その結果、ビット番号14,13の2ビット分(上位から順に1,0)がはみ出して捨てられ、最後にはみ出したビット番号13の値0がオーバフローフラグ(OF)に設定される。

➡ ビット番号12以下は左にずれ、右端の空いたビットには0が設定される。この結果、GR4の値はB7BC₁₆となったのである。

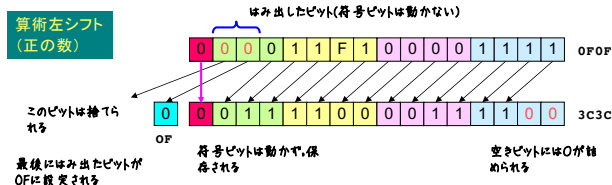
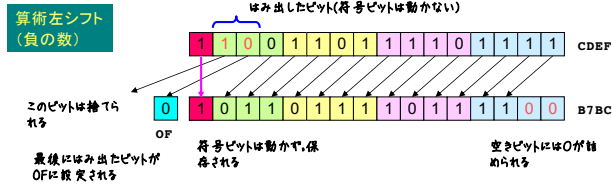
131

131

3-6 算術シフト命令(3)

■算術左シフト(2)

【19行目】 SLA GR5,2



132

132

3-6 算術シフト命令(4)

■算術右シフト(1)

【20行目】 SRA GR6,2

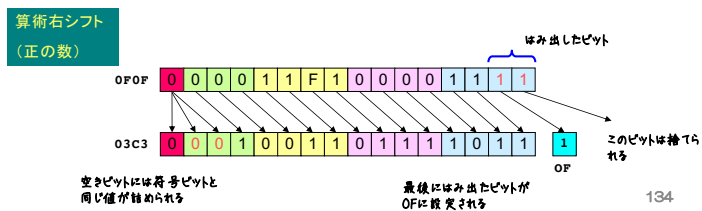
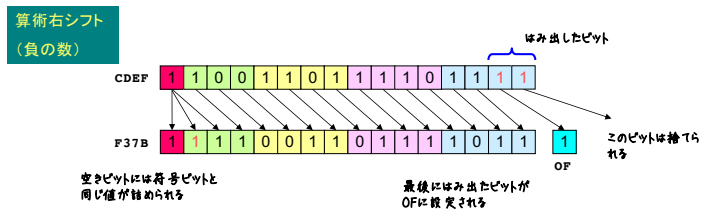
- 2ビット分だけ右にシフトしている。ビット番号1,0の2ビット分がはみ出し、ビット番号2以上が右にずれる。
- 左端には符号ビットと同じ値である1が詰められる。この結果GR6の値はF37B₁₆となる。
- 最後にはみ出たビットは1になるので、OFには1が設定される。
- 算術右シフトの場合には「符号ビットと同じものが詰められる」、つまり、もとの値の最上位ビットが2のまま順に詰められる。

133

133

3-6 算術シフト命令(5)

■算術右シフト(2)



134

134

3-6 算術シフト命令(6)

■算術シフトの仕様

- SLA命令とSRA命令の仕様を表3.10に示す。
- オペランドの書き方は論理シフト命令SLLやSRLとまったく同じである。
- ただし、符号ビットの扱いが論理シフト命令とは異なっている。

命令	書き方		命令の説明
	命令コード	オペランド	
算術左シフト Shift Left Arithmetic	SLA	r,adr[,x]	符号を除き(r)を実効アドレスで指定したビット数だけ左または右にシフトする。シフトの結果、空いたビット位置には、左シフトのときは0、右シフトのときは符号と同じものが入る。
算術右シフト Shift Right Arithmetic	SRA	r,adr[,x]	

135

135

3-7 簡単な掛け算と割り算(0)

```

1:PRG0307 STAR
4:      LAD GR1,10
5:      LAD GR2,10
6:      LAD GR3,10
7:      LAD GR4,10
8:      LAD GR5,10
9:      LAD GR6,10
11:     SLL GR1,1  :10×2
12:     SLL GR2,2  :10×4
13:     SLL GR3,3  :10×8
15:     SRL GR4,1  :10÷2
16:     SRL GR5,2  :10÷4
17:     SRL GR6,3  :10÷8
19:     LAD GR7,21
20:     LAD GR0,21
21:     SLL GR7,1  :21×2
22:     SLL GR0,3  :21×8
23:     ADDA GR0,GR7 :21×10(=21×(2+8)=21×2+21×8)
26:     RET
27:     END

```

136

136

3-7 簡単な掛け算と割り算(2)

■シフト演算と掛け算・割り算の関係(1)

【11行目】 SLL GR1,1

- ▶ GR1を左に1ビット分論理シフトしている。
- ▶ 2進数で考えると、単にビット列を左にずらしただけであるが、10進数で考えると、GR1の値は10進数20(0014₁₆)になっている。
- ▶ 1ビットの左シフトを1回、2回、3回、…と繰り返すと、値は21=2倍、22=4倍、23=8倍、…となる。

【15行目】 SRL GR4,1

- ▶ GR4を右に1ビット分論理シフトしている。
- ▶ GR4の値は5になって、GR4を2で割った値になっている。
- ▶ 実は左シフトは2のべき乗の値の掛け算であったのに対し、逆に右シフトは2のべき乗の値の割り算となる。
- ▶ 1ビットの右シフトを1回、2回、3回、…と繰り返すと、値は21=2、22=4、23=8、…で割られることとなる。

137






137

3-7 簡単な掛け算と割り算(3)






■シフト演算と掛け算・割り算の関係(2)

- ▶ 左シフトによって2のべき乗の掛け算、右シフトによって2のべき乗の割り算が演算できる。
- ▶ 左へ n 個ずらすことはその数値に 10^n を掛けることに相当し、右へ n 個ずらすことは、その数値を 10^n で割ることに相当するのである。
- ▶ 割り切れなかった場合の余りは無視されることに注意しよう。
- ▶ 例えば、16行目のように000A₁₆(10進数で10)を2ビット右シフトすると、0002₁₆となり、 $\lceil 10 \div 22 = 2$ あまり2₂のあまりの部分は無視される。

10進数の場合

$\div 10$  5
 $\div 10$  52
 $\times 10$  520
 $\times 10$  5200
 $\times 10$  52000

2進数の場合

$\div 2$  00000000 00000010 (10進数で2)
 $\div 2$  00000000 00000101 (10進数で5)
 $\times 2$  00000000 00001010 (10進数で10)
 $\times 2$  00000000 00010100 (10進数で20)
 $\times 2$  00000000 00101000 (10進数で40)

138

138

3-7 簡単な掛け算と割り算(4)

■シフト命令の組合せによる掛け算

- ▶ シフト命令を組み合わせることで、2のべき乗の値以外の掛け算も容易に行うことができる。

- ▶ 例えば、 21×10 を計算してみよう。
- ▶ まず、 21×10 の10を、 $10 = 2 + 8$ と、2のべき乗の和に分解する。
- ▶ すると、 21×10 は、

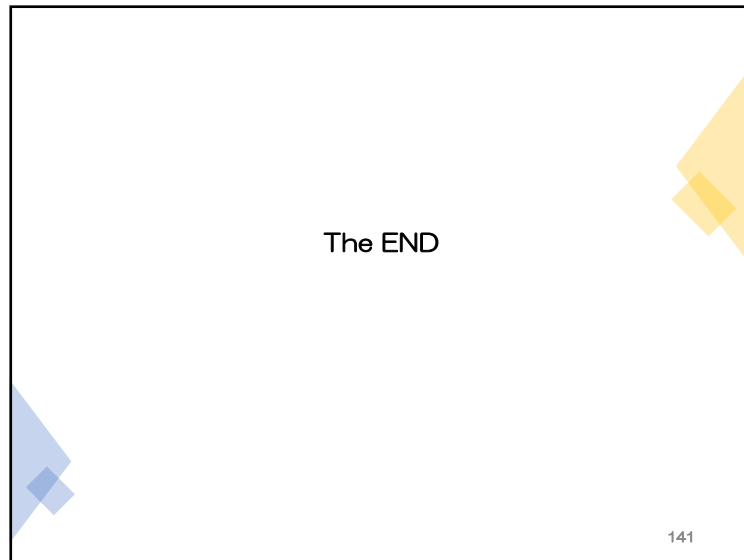
$$21 \times 10 = 21 \times (2 + 8)$$

$$= (21 \times 2) + (21 \times 8)$$
- ▶ 2のべき乗の値の掛け算はシフト演算で実現できるので、

$$21 \times 10 = (21 \text{の1ビット左シフト}) + (21 \text{の3ビット左シフト})$$
と考えることができる。

139

139



141